

# 1 Pinning Down the Strong Wilber 1 Bound for 2 Binary Search Trees

3 **Parinya Chalermsook**

4 Aalto University, Finland

5 chalermsook@gmail.com

6 **Julia Chuzhoy**

7 Toyota Technological Institute at Chicago

8 cjulia@ttic.edu

9 **Thatchaphol Saranurak**

10 Toyota Technological Institute at Chicago

11 saranurak@ttic.edu

## 12 — Abstract —

---

13 The dynamic optimality conjecture, postulating the existence of an  $O(1)$ -competitive online algorithm  
14 for binary search trees (BSTs), is among the most fundamental open problems in dynamic data  
15 structures. Despite extensive work and some notable progress, including, for example, the Tango  
16 Trees (Demaine et al., FOCS 2004), that give the best currently known  $O(\log \log n)$ -competitive  
17 algorithm, the conjecture remains widely open. One of the main hurdles towards settling the  
18 conjecture is that we currently do not have approximation algorithms achieving better than an  
19  $O(\log \log n)$ -approximation, even in the offline setting. All known non-trivial algorithms for BST's so  
20 far rely on comparing the algorithm's cost with the so-called Wilber's first bound (WB-1). Therefore,  
21 establishing the worst-case relationship between this bound and the optimal solution cost appears  
22 crucial for further progress, and it is an interesting open question in its own right.

23 Our contribution is two-fold. First, we show that the gap between the WB-1 bound and the optimal  
24 solution value can be as large as  $\Omega(\log \log n / \log \log \log n)$ ; in fact, we show that the gap holds even  
25 for several stronger variants of the bound. Second, we provide a simple algorithm, that, given an  
26 integer  $D > 0$ , obtains an  $O(D)$ -approximation in time  $\exp\left(O\left(n^{1/2^{\Omega(D)}} \log n\right)\right)$ . In particular,  
27 this yields a constant-factor approximation algorithm with sub-exponential running time. Moreover,  
28 we obtain a simpler and cleaner efficient  $O(\log \log n)$ -approximation algorithm that can be used  
29 in an online setting. Finally, we suggest a new bound, that we call the *Guillotine Bound*, that is  
30 stronger than WB-1, while maintaining its algorithm-friendly nature, that we hope will lead to  
31 better algorithms. All our results use the geometric interpretation of the problem, leading to cleaner  
32 and simpler analysis.

33 **2012 ACM Subject Classification** Theory of computation → Data structures design and analysis

34 **Keywords and phrases** Binary search trees, Dynamic optimality, Wilber bounds

35 **Digital Object Identifier** 10.4230/LIPIcs.APPROX/RANDOM.2020.33

36 **Related Version** A full version of the paper is available at <https://arxiv.org/abs/1912.02900>.

37 **Funding** *Parinya Chalermsook*: Supported by European Research Council (ERC) under the  
38 European Union's Horizon 2020 research and innovation programme (grant agreement No. 759557)  
39 and by Academy of Finland Research Fellows, under grant No. 310415.

40 *Julia Chuzhoy*: Supported in part by NSF grant CCF-1616584. Part of the work was done while the



© Parinya Chalermsook, Julia Chuzhoy, and Thatchaphol Saranurak;  
licensed under Creative Commons License CC-BY

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques  
(APPROX/RANDOM 2020).

Editors: Jarosław Byrka and Raghu Meka; Article No. 33; pp. 33:1–33:22

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

41 second author was a Weston visiting professor at the Department of Computer Science and Applied  
 42 Mathematics, Weizmann Institute of Science.

## 43 **1** Introduction

44 Binary search trees (BST's) are a fundamental data structure that has been extensively  
 45 studied for many decades. Informally, suppose we are given as input an **online** access  
 46 sequence  $X = \{x_1, \dots, x_m\}$  of keys from  $\{1, \dots, n\}$ , and our goal is to maintain a binary  
 47 search tree  $T$  over the set  $\{1, \dots, n\}$  of keys. The algorithm is allowed to modify the tree  
 48  $T$  after each access; the tree obtained after the  $i$ th access is denoted by  $T_{i+1}$ . Each such  
 49 modification involves a sequence of *rotation* operations that transform the current tree  $T_i$  into  
 50 a new tree  $T_{i+1}$ . The cost of the transformation is the total number of rotations performed  
 51 plus the depth of the key  $x_i$  in the tree  $T_i$ . The total cost of the algorithm is the total cost  
 52 of all transformations performed as the sequence  $X$  is processed. We denote by  $\text{OPT}(X)$  the  
 53 smallest cost of any algorithm for maintaining a BST for the access sequence  $X$ , when the  
 54 whole sequence  $X$  is known to the algorithm in advance.

55 Several algorithms for BST's, whose costs are guaranteed to be  $O(m \log n)$  for any access  
 56 sequence, such as AVL-trees [1] and red-black trees [2], are known since the 60's. Moreover,  
 57 it is well known that there are length- $m$  access sequences  $X$  on  $n$  keys, for which  $\text{OPT}(X) =$   
 58  $\Omega(m \log n)$ . However, such optimal worst-case guarantees are often unsatisfactory from both  
 59 practical and theoretical perspectives, as one can often obtain better results for “structured”  
 60 inputs. Arguably, a better notion of the algorithm's performance to consider is *instance*  
 61 *optimality*, where the algorithm's performance is compared to the optimal cost  $\text{OPT}(X)$  for  
 62 the specific input access sequence  $X$ . This notion is naturally captured by the algorithm's  
 63 *competitive ratio*: we say that an algorithm for BST's is  $\alpha$ -*competitive*, if, for every online  
 64 input access sequence  $X$ , the cost of the algorithm's execution on  $X$  is at most  $\alpha \cdot \text{OPT}(X)$ .  
 65 Since for every length- $m$  access sequence  $X$ ,  $\text{OPT}(X) \geq m$ , the above-mentioned algorithms  
 66 that provide worst-case  $O(m \log n)$ -cost guarantees are also  $O(\log n)$ -competitive. However,  
 67 there are many known important special cases, in which the value of the optimal solution  
 68 is  $O(m)$ , and for which the existence of an  $O(1)$ -competitive algorithm would lead to a  
 69 much better performance, including some interesting applications, such as, for example,  
 70 adaptive sorting [23, 6, 19, 22, 13, 20, 12, 8, 7, 3, 5, 4]. A striking conjecture of Sleator  
 71 and Tarjan [21] from 1985, called the *dynamic optimality conjecture*, asserts that the *Splay*  
 72 *Trees* provide an  $O(1)$ -competitive algorithm for BST's. This conjecture has sparked a long  
 73 line of research, but despite the continuing effort, and the seeming simplicity of BST's, it  
 74 remains widely open. In a breakthrough result, Demaine et al. [10] proposed the Tango Trees  
 75 algorithm, that achieves an  $O(\log \log n)$ -competitive ratio, and has remained the best known  
 76 algorithm for the problem, for over 15 years. A natural avenue for overcoming this barrier  
 77 is to first consider the “easier” task of designing (offline) approximation algorithms, whose  
 78 approximation factor is below  $O(\log \log n)$ . Designing better approximation algorithms is  
 79 often a precursor to obtaining better online algorithms, and it is a natural stepping stone  
 80 towards this goal.

81 The main obstacle towards designing better algorithms, both in the online and the offline  
 82 settings, is obtaining tight lower bounds on the value  $\text{OPT}(X)$ , that can be used in algorithm  
 83 design. If the input access sequence  $X$  has length  $m$ , and it contains  $n$  keys, then it is easy  
 84 to see that  $\text{OPT}(X) \geq \Omega(m)$ , and, by using any balanced BST's, such as AVL-trees, one  
 85 can show that  $\text{OPT}(X) = O(m \log n)$ . This trivially implies an  $O(\log n)$ -approximation for

86 both offline and online settings. However, in order to obtain better approximation, these  
 87 simple bounds do not seem sufficient. Wilber [25] proposed two new bounds, that we refer  
 88 to as the first Wilber Bound (WB-1) and the second Wilber Bound (WB-2). He proved that,  
 89 for every input sequence  $X$ , the values of both these bounds on  $X$  are at most  $\text{OPT}(X)$ .  
 90 The breakthrough result of Demaine et al. [10], that gives an  $O(\log \log n)$ -competitive online  
 91 algorithm, relies on the WB-1 bound. In particular, they show that the cost of the solution  
 92 produced by their algorithm is within an  $O(\log \log n)$ -factor from the WB-1 bound on the  
 93 given input sequence  $X$ , and hence from  $\text{OPT}(X)$ . This in turn implies that, for every input  
 94 sequence  $X$ , the value of the WB-1 bound is within an  $O(\log \log n)$  factor from  $\text{OPT}(X)$ .  
 95 Follow-up work [24, 14] improved several aspects of Tango Trees, but it did not improve  
 96 the approximation factor. Additional lower bounds on  $\text{OPT}$ , that subsume both the WB-1  
 97 and the WB-2 bounds, were suggested in [9, 11, 15], but unfortunately it is not clear how  
 98 to exploit them in algorithm design. To this day, the only method we have for designing  
 99 non-trivial online or offline approximation algorithms for BST's is by relying on the WB-1  
 100 bound, and this seems to be the most promising approach for obtaining better algorithms.  
 101 In order to make further progress on both online and offline approximation algorithms for  
 102 BST's, it therefore appears crucial that we better understand the relationship between the  
 103 WB-1 bound and the optimal solution cost.

104 Informally, the WB-1 bound relies on recursive partitioning of the input key sequence, that  
 105 can be represented by a partitioning tree. The standard WB-1 bound (that we refer to  
 106 as the *weak* WB-1 bound) only considers a single such partitioning tree. It is well-known  
 107 (see e.g. [10, 24, 16]), that the gap between  $\text{OPT}(X)$  and the weak WB-1 bound for an  
 108 access sequence  $X$  may be as large as  $\Omega(\log \log n)$ . However, the “bad” access sequence  
 109  $X$  used to obtain this gap is highly dependent on the fixed partitioning tree  $T$ . It is then  
 110 natural to consider a stronger variant of WB-1, that we refer to as *strong* WB-1 bound and  
 111 denote by  $\text{WB}(X)$ , that maximizes the weak WB-1 bound over all such partitioning trees. As  
 112 suggested by Iacono [16], and by Kozma [17], this gives a promising approach for improving  
 113 the  $O(\log \log n)$ -approximation factor.

114 In this paper, we show that, even for this strong variant of Wilber Bound, the gap between  
 115  $\text{OPT}(X)$  and  $\text{WB}(X)$  may be as large as  $\Omega(\log \log n / \log \log \log n)$ . This negative result  
 116 extends to an even stronger variant of the Wilber Bound that we discuss below.

117 Our second set of results is algorithmic. We show an (offline) algorithm that, given an input  
 118 sequence  $X$  and a positive integer  $D$ , obtains an  $O(D)$ -approximation, in time  $\text{poly}(m) \cdot$   
 119  $\exp\left(n^{1/2^{\Omega(D)}} \log n\right)$ . When  $D$  is constant, the algorithm obtains an  $O(1)$ -approximation  
 120 in sub-exponential time. When  $D$  is  $\Theta(\log \log n)$ , it matches the best current efficient  
 121  $O(\log \log n)$ -approximation algorithm. In the latter case, we can also adapt the algorithm to  
 122 the online setting, obtaining an  $O(\log \log n)$ -competitive online algorithm.

123 All our results use the geometric interpretation of the problem, introduced by Demaine  
 124 et al. [9], leading to clean divide-and-conquer-style arguments that avoid, for example, the  
 125 notion of pointers and rotations. We feel that this approach, in addition to providing a cleaner  
 126 and simpler view of the problem, is more natural to work with in the context of approximation  
 127 algorithms, and should be more amenable to the powerful geometric techniques in the field.

128 **Independent Work.** Independently from our work, Lecomte and Weinstein [18] showed that  
 129 second Wilber Bound (WB-2) dominates the WB-1 bound, and moreover, they show an access  
 130 sequence  $X$  for which the two bounds have a gap of  $\Omega(\log \log n)$ . In particular, their result

131 implies that the gap between  $\text{WB}(X)$  and  $\text{OPT}(X)$  is  $\Omega(\log \log n)$  for that access sequence.  
 132 We note that the access sequence  $X$  that is used in our negative results also provides a  
 133 gap of  $\Omega(\log \log n / \log \log \log n)$  between the WB-2 and the WB-1 bounds, although we only  
 134 realized this after hearing the statement of the results of [18]. Additionally, Lecomte and  
 135 Weinstein show that the WB-2 bound is invariant under rotations, and use this to show that,  
 136 when the WB-2 bound is constant, then the Independent Rectangle bound of [9] is linear.  
 137 We now provide a more detailed description of our results.

## 138 Our Results and Techniques

139 We use the geometric interpretation of the problem, introduced by Demaine et al. [9], that  
 140 we refer to as the Min-Sat problem. Let  $P$  be any set of points in the plane. We say that two  
 141 points  $p, q \in P$  are *collinear* iff either their  $x$ -coordinates are equal, or their  $y$ -coordinates  
 142 are equal. If  $p$  and  $q$  are non-collinear, then we let  $\square_{p,q}$  be the smallest closed rectangle  
 143 containing both  $p$  and  $q$ ; notice that  $p$  and  $q$  must be diagonally opposite corners of this  
 144 rectangle. We say that the pair  $(p, q)$  of points is *satisfied* in  $P$  iff there is some additional  
 145 point  $r \neq p, q$  in  $P$  that lies in  $\square_{p,q}$  (the point may lie on the boundary of the rectangle).  
 146 Lastly, we say that the set  $P$  of points is satisfied iff for every pair  $p, q \in P$  of distinct points,  
 147 either  $p$  and  $q$  are collinear, or they are satisfied in  $P$ .

148 In the Min-Sat problem, the input is a set  $P$  of points in the plane with integral  $x$ - and  
 149  $y$ -coordinates; we assume that all  $x$ -coordinates are between 1 and  $n$ , and all  $y$ -coordinates  
 150 are between 1 and  $m$  and distinct from each other, and that  $|P| = m$ . The goal is to find a  
 151 minimum-cardinality set  $Y$  of points, such that the set  $P \cup Y$  of points is satisfied.

152 An access sequence  $X$  over keys  $\{1, \dots, n\}$  can be represented by a set  $P$  of points in the  
 153 plane as follows: if a key  $x$  is accessed at time  $y$ , then add the point  $(x, y)$  to  $P$ . Demaine  
 154 et al. [9] showed that, for every access sequence  $X$ , if we denote by  $P$  the corresponding set  
 155 of points in the plane, then the value of the optimal solution to the Min-Sat problem on  $P$  is  
 156  $\Theta(\text{OPT}(X))$ . They also showed that, in order to obtain an  $O(\alpha)$ -approximation algorithm  
 157 for BST's, it is sufficient to obtain an  $\alpha$ -approximation algorithm for the Min-Sat problem.  
 158 In the online version of the Min-Sat problem, at every time step  $t$ , we discover the unique  
 159 input point whose  $y$ -coordinate is  $t$ , and we need to decide which points with  $y$ -coordinate  $t$   
 160 to add to the solution. Demaine et al. [9] also showed that an  $\alpha$ -competitive online algorithm  
 161 for Min-Sat implies an  $O(\alpha)$ -competitive online algorithm for BST's. For convenience, we do  
 162 not distinguish between the input access sequence  $X$  and the corresponding set of points in  
 163 the plane, that we also denote by  $X$ .

164 **Negative Results for WB-1.** We say that an input access sequence  $X$  is a *permutation* if  
 165 each key in  $\{1, \dots, n\}$  is accessed exactly once. Equivalently, in the geometric view, every  
 166 column with an integral  $x$ -coordinate contains exactly one input point.

167 Informally, the WB-1 bound for an input sequence  $X$  is defined as follows. Let  $B$  be the  
 168 bounding box containing all points of  $X$ , and consider any vertical line  $L$  drawn across  $B$ ,  
 169 that partitions it into two vertical strips, separating the points of  $X$  into two subsets  $X_1$  and  
 170  $X_2$ . Assume that the points of  $X$  are ordered by their  $y$ -coordinates from smallest to largest.  
 171 We say that a pair  $(x, x') \in X$  of points *cross* the line  $L$ , iff  $x$  and  $x'$  are consecutive points  
 172 of  $X$ , and they lie on different sides of  $L$ . Let  $C(L)$  be the number of all pairs of points in  $X$   
 173 that cross  $L$ . We then continue this process recursively with  $X_1$  and  $X_2$ , with the final value

174 of the WB-1 bound being the sum of the two resulting bounds obtained for  $X_1$  and  $X_2$ , and  
 175  $C(L)$ . This recursive partitioning process can be represented by a binary tree  $T$  that we call  
 176 a *partitioning tree* (we note that the partitioning tree is not related to the BST tree that the  
 177 BST algorithm maintains). Every vertex  $v$  of the partitioning tree is associated with a vertical  
 178 strip  $S(v)$ , where for the root vertex  $r$ ,  $S(r) = B$ . If the partitioning algorithm uses a vertical  
 179 line  $L$  to partition the strip  $S(v)$  into two sub-strips  $S_1$  and  $S_2$ , then vertex  $v$  has two children,  
 180 whose corresponding strips are  $S_1$  and  $S_2$ . Note that every sequence of vertical lines used in  
 181 the recursive partitioning procedure corresponds to a unique partitioning tree and vice versa.  
 182 Given a set  $X$  of points and a partitioning tree  $T$ , we denote by  $\text{WB}_T(X)$  the WB-1 bound  
 183 obtained for  $X$  while following the partitioning scheme defined by  $T$ . Wilber [25] showed  
 184 that, for every partitioning tree  $T$ ,  $\text{OPT}(X) \geq \Omega(\text{WB}_T(X))$  holds. Moreover, Demaine et  
 185 al. [10] showed that, if  $T$  is a balanced tree, then  $\text{OPT}(X) \leq O(\log \log n) \cdot \text{WB}_T(X)$ . These  
 186 two bounds are used to obtain the  $O(\log \log n)$ -competitive algorithm of [10]. We call this  
 187 variant of WB-1, that is defined with respect to a fixed tree  $T$ , the *weak* WB-1 bound.

188 Unfortunately, it is well-known (see e.g. [10, 24, 16]), that the gap between  $\text{OPT}(X)$  and  
 189 the weak WB-1 bound on an input  $X$  may be as large as  $\Omega(\log \log n)$ . In other words,  
 190 for any fixed partitioning tree  $T$ , there exists an input  $X$  (that depends on  $T$ ), with  
 191  $\text{WB}_T(X) \leq O(\text{OPT}(X)/\log \log n)$ . However, the construction of this “bad” input  $X$  depends  
 192 on the fixed partitioning tree  $T$ . We consider a stronger variant of WB-1, that we refer to as  
 193 *strong* WB-1 bound and denote by  $\text{WB}(X)$ , that maximizes the weak WB-1 bound over all  
 194 such partitioning trees, that is,  $\text{WB}(X) = \max_T \{\text{WB}_T(X)\}$ . Using this stronger bound as an  
 195 alternative to weak WB-1 in order to obtain better approximation algorithms was suggested  
 196 by Iacono [16], and by Kozma [17].

197 Our first result rules out this approach: we show that, even for the strong WB-1 bound, the  
 198 gap between  $\text{WB}(X)$  and  $\text{OPT}(X)$  may be as large as  $\Omega(\log \log n / \log \log \log n)$ , even if the  
 199 input  $X$  is a permutation.

200 ► **Theorem 1.** *For every integer  $n'$ , there is an integer  $n \geq n'$ , and an access sequence  $X$  on*  
 201  *$n$  keys with  $|X| = n$ , such that  $X$  is a permutation,  $\text{OPT}(X) \geq \Omega(n \log \log n)$ , but  $\text{WB}(X) \leq$*   
 202  *$O(n \log \log \log n)$ . In other words, for every partitioning tree  $T$ ,  $\frac{\text{OPT}(X)}{\text{WB}_T(X)} \geq \Omega\left(\frac{\log \log n}{\log \log \log n}\right)$ .*

203 We note that it is well known (see e.g. [5]), that any  $c$ -approximation algorithm for per-  
 204 mutation input can be turned into an  $O(c)$ -approximation algorithm for any input sequence.  
 205 However, the known instances that achieve an  $\Omega(\log \log n)$ -gap between the weak WB-1 bound  
 206 and  $\text{OPT}$  are not permutations. Therefore, our result is the first to provide a super-constant  
 207 gap between WB-1 and  $\text{OPT}$  for permutations, even for the case of weak WB-1.

208 **Extension of WB-1.** We consider several generalizations of the WB-1 bound that allow  
 209 partitioning the plane both horizontally and vertically. We call the new bounds the *consistent*  
 210 *Guillotine Bound* and the *Guillotine Bound*. Our negative result extends to the consistent  
 211 Guillotine Bound but *not* to the Guillotine Bound. The Guillotine Bound seems to maintain  
 212 the algorithm-friendly nature of WB-1, and in particular it naturally fits into the algorithmic  
 213 framework that we propose. We hope that this bound can lead to improved algorithms, both  
 214 in the offline and the online settings

215 **Separating the Two Wilber Bounds.** The sequence  $X$  given by Theorem 1 not only  
 216 provides a separation between WB-1 and  $\text{OPT}$ , but it also provides a separation between the  
 217 WB-1 bound and the WB-2 bound (also called the *funnel* bound). The latter can be defined  
 218 in the geometric view as follows. Recall that, for a pair of points  $x, y \in X$ ,  $\square_{x,y}$  is the smallest

219 closed rectangle containing both  $x$  and  $y$ . For a point  $x$  in the access sequence  $X$ , the *funnel*  
 220 of  $x$  is the set of all points  $y \in X$ , for which  $\square_{x,y}$  does not contain any point of  $X \setminus \{x, y\}$ , and  
 221  $\text{alt}(x)$  is the number of alterations between the left of  $x$  and the right of  $x$  in the funnel of  $x$ .  
 222 The second Wilber Bound for sequence  $X$  is then defined as:  $\text{WB}^{(2)}(X) = |X| + \sum_{x \in X} \text{alt}(x)$ .  
 223 We show that, for the sequence  $X$  given by Theorem 1,  $\text{WB}^{(2)}(X) \geq \Omega(n \log \log n)$  holds, and  
 224 therefore  $\text{WB}^{(2)}(X)/\text{WB}(X) \geq \Omega(\log \log n / \log \log \log n)$  for that sequence, implying that  
 225 the gap between  $\text{WB}(X)$  and  $\text{WB}^{(2)}(X)$  may be as large as  $\Omega(\log \log n / \log \log \log n)$ . We  
 226 note that we only realized that our results provide this stronger separation between the two  
 227 Wilber bounds after hearing the statements of the results from the independent work of  
 228 Lecomte and Weinstein [18] mentioned above.

229 **Algorithmic Results.** We provide new simple approximation algorithms for the problem,  
 230 that rely on its geometric interpretation, namely the Min-Sat problem.

231 **► Theorem 2.** *There is an offline algorithm for Min-Sat, that, given any integral parameter*  
 232  *$D \geq 1$ , and an access sequence  $X$  to  $n$  keys of length  $m$ , produces a solution of cost at*  
 233 *most  $O(D \cdot \text{OPT}(X))$  and has running time at most  $\text{poly}(m) \cdot \exp\left(O\left(n^{1/2^{\Theta(D)}} \log n\right)\right)$ . For*  
 234  *$D = O(\log \log n)$ , the algorithm's running time is polynomial in  $n$  and  $m$ , and it can be*  
 235 *adapted to the online setting, achieving an  $O(\log \log n)$ -competitive ratio.*

236 Our results show that the problem of obtaining a constant-factor approximation for Min-Sat  
 237 cannot be NP-hard, unless  $\text{NP} \subseteq \text{SUBEXP}$ , where  $\text{SUBEXP} = \bigcap_{\epsilon > 0} \text{DTime}[2^{n^\epsilon}]$ . This, in  
 238 turn, provides a positive evidence towards the dynamic optimality conjecture, as one natural  
 239 avenue to disproving it is to show that obtaining a constant-factor approximation for BST's is  
 240 NP-hard. Our results rule out this possibility, unless  $\text{NP} \subseteq \text{SUBEXP}$ . While the  $O(\log \log n)$ -  
 241 approximation factor achieved by our algorithm in time  $\text{poly}(mn)$  is similar to that achieved  
 242 by other known algorithms [10, 14, 24], this is the first algorithm that relies solely on the  
 243 geometric formulation of the problem, which is arguably cleaner, simpler, and better suited  
 244 for exploiting the rich toolkit of algorithmic techniques developed in the areas of online and  
 245 approximation algorithms.

246 **Organization.** We start with preliminaries in Section 2. In Section 3, we state decomposition  
 247 theorems which are useful for both of our negative and positive results. In Section 4, we  
 248 provide the proof of Theorem 1, our main negative result. We discuss extensions of the  
 249 Wilber Bound in Section 5. Lastly, we show our main positive result – the proof of Theorem 2  
 250 – in Section 6. Due to lack of space, many of the proofs are deferred to the full version.

## 251 **2 Preliminaries**

252 All our results only use the geometric interpretation of the problem, that we refer to as the  
 253 Min-Sat problem. We include the formal definition of algorithms for BST's and formally  
 254 state their equivalence to Min-Sat in the full version.

### 255 **2.1 The Min-Sat Problem**

256 For a point  $p \in \mathbb{R}^2$  in the plane, we denote by  $p.x$  and  $p.y$  its  $x$ - and  $y$ -coordinates, respectively.  
 257 Given any pair  $p, p'$  of points, we say that they are *collinear* if  $p.x = p'.x$  or  $p.y = p'.y$ . If  $p$

258 and  $p'$  are not collinear, then we let  $\square_{p,p'}$  be the smallest closed rectangle containing both  $p$   
 259 and  $p'$ ; note that  $p$  and  $p'$  must be diagonally opposite corners of the rectangle.

260 ► **Definition 3.** We say that a non-collinear pair  $p, p'$  of points is satisfied by a point  $p''$   
 261 if  $p''$  is distinct from  $p$  and  $p'$  and  $p'' \in \square_{p,p'}$  (where  $p''$  may lie on the boundary of the  
 262 rectangle). We say that a set  $S$  of points is satisfied iff for every non-collinear pair  $p, p' \in S$   
 263 of points, there is some point  $p'' \in S$  that satisfies this pair.

264 We refer to horizontal and vertical lines as *rows* and *columns* respectively. For a collection  
 265 of points  $X$ , the *active rows* of  $X$  are the rows that contain at least one point in  $X$ . We  
 266 define the notion of *active columns* analogously. We denote by  $r(X)$  and  $c(X)$  the number  
 267 of active rows and active columns of the point set  $X$ , respectively. We say that a point set  
 268  $X$  is a *semi-permutation* if every active row contains exactly one point of  $X$ . Note that,  
 269 if  $X$  is a semi-permutation, then  $c(X) \leq r(X)$ . We say that  $X$  is a *permutation* if it is a  
 270 semi-permutation, and additionally, every active column contains exactly one point of  $X$ .  
 271 Clearly, if  $X$  is a permutation, then  $c(X) = r(X) = |X|$ . We denote by  $B$  the smallest closed  
 272 rectangle containing all points of  $X$ , and call  $B$  the *bounding box*.

273 We are now ready to define the Min-Sat problem. The input to the problem is a set  $X$  of  
 274 points that is a semi-permutation, and the goal is to compute a minimum-cardinality set  $Y$   
 275 of points, such that  $X \cup Y$  is satisfied. We say that a set  $Y$  of points is a *feasible solution*  
 276 for  $X$  if  $X \cup Y$  is satisfied. We denote by  $\text{OPT}(X)$  the minimum value  $|Y|$  of any feasible  
 277 solution  $Y$  for  $X$ .<sup>1</sup> In the online version of the Min-Sat problem, at every time step  $t$ , we  
 278 discover the unique input point from  $X$  whose  $y$ -coordinate is  $t$ , and we need to decide which  
 279 points with  $y$ -coordinate  $t$  to add to the solution  $Y$ . The Min-Sat problem is equivalent to  
 280 the BST problem, in the following sense:

281 ► **Theorem 4 ([9]).** Any efficient  $\alpha$ -approximation algorithm for Min-Sat can be trans-  
 282 formed into an efficient  $O(\alpha)$ -approximation algorithm for BST's, and similarly any online  
 283  $\alpha$ -competitive algorithm for Min-Sat can be transformed into an online  $O(\alpha)$ -competitive  
 284 algorithm for BST's.

## 285 2.2 Basic Geometric Properties

286 The following observation is well known (see, e.g. Observation 2.1 from [9]).

287 ► **Observation 5.** Let  $Z$  be any satisfied point set. Then for every pair  $p, q \in Z$  of distinct  
 288 points, there is a point  $r \in \square_{p,q} \setminus \{p, q\}$  such that  $r.x = p.x$  or  $r.y = p.y$ .

289 **Collapsing Sets of Columns or Rows.** Assume that we are given any set  $X$  of points, and  
 290 any collection  $\mathcal{C}$  of consecutive active columns for  $X$ . In order to collapse the set  $\mathcal{C}$  of columns,  
 291 we replace  $\mathcal{C}$  with a single representative column  $C$  (for concreteness, we use the column  
 292 of  $\mathcal{C}$  with minimum  $x$ -coordinate). For every point  $p \in X$  that lies on a column of  $\mathcal{C}$ , we  
 293 replace  $p$  with a new point, lying on the column  $C$ , whose  $y$ -coordinate remains the same.  
 294 Formally, we replace point  $p$  with point  $(x, p.y)$ , where  $x$  is the  $x$ -coordinate of the column

<sup>1</sup> We remark that in the original paper that introduced this problem [9], the value of the solution is defined as  $|X \cup Y|$ , while our solution value is  $|Y|$ . It is easy to see that for any semi-permutation  $X$  and solution  $Y$  for  $X$ ,  $|Y| \geq \Omega(|X|)$  must hold, so the two definitions are equivalent to within factor 2.

295  $C$ . We denote by  $X|_C$  the resulting new set of points. We can similarly define collapsing set  
 296 of rows. The following useful observation is easy to verify.

297 ► **Observation 6.** *Let  $S$  be any set of points, and let  $C$  be any collection of consecutive active*  
 298 *columns (or rows) with respect to  $S$ . If  $S$  is a satisfied set of points, then so is  $S|_C$ .*

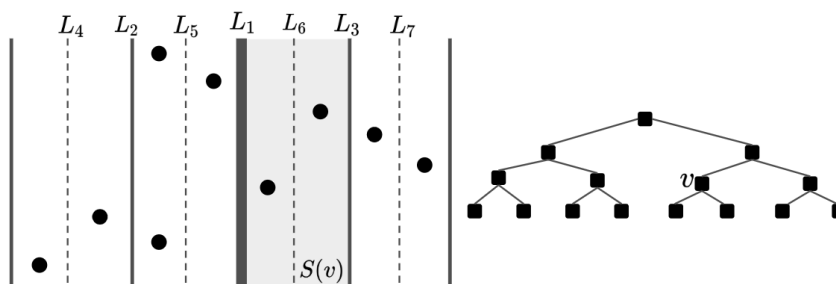
299 **Canonical Solutions.** We say that a solution  $Y$  for input  $X$  is *canonical* iff every point  
 300  $p \in Y$  lies on an active row and an active column of  $X$ . It is easy to see that *any* solution  
 301 can be transformed into a canonical solution, without increasing its cost (see the full version  
 302 of the paper for the proof).

303 ► **Observation 7.** *There is an efficient algorithm, that, given an instance  $X$  of Min-Sat and*  
 304 *any feasible solution  $Y$  for  $X$ , computes a feasible canonical solution  $\hat{Y}$  for  $X$  with  $|\hat{Y}| \leq |Y|$ .*

### 305 2.3 Partitioning Trees

306 We now turn to define partitioning trees, that are central to both defining the WB-1 bound  
 307 and to describing our algorithm.

308 Let  $X$  be the a set of points that is a semi-permutation. We can assume without loss of  
 309 generality that every column with an integral  $x$ -coordinate between 1 and  $c(X)$  inclusive  
 310 contains at least one point of  $X$ . Let  $B$  be the bounding box of  $X$ . Assume that the set of  
 311 active columns is  $\{C_1, \dots, C_a\}$ , where  $a = c(X)$ , and that for all  $1 \leq i \leq a$ , the  $x$ -coordinate  
 312 of column  $C_i$  is  $i$ . Let  $\mathcal{L}$  be the set of all vertical lines with half-integral  $x$ -coordinates between  
 313  $1 + 1/2$  and  $a - 1/2$  (inclusive). Throughout, we refer to the vertical lines in  $\mathcal{L}$  as *auxiliary*  
 314 *columns*. Let  $\sigma$  be an arbitrary ordering of the lines of  $\mathcal{L}$  and denote  $\sigma = (L_1, L_2, \dots, L_{a-1})$ .  
 315 We define a hierarchical partitioning of the bounding box  $B$  into vertical strips using  $\sigma$ , as  
 316 follows. We perform  $a - 1$  iterations. In the first iteration, we partition the bounding box  
 317  $B$ , using the line  $L_1$ , into two vertical strips,  $S_L$  and  $S_B$ . For  $1 < i \leq a - 1$ , in iteration  $i$   
 318 we consider the line  $L_i$ , and we let  $S$  be the unique vertical strip in the current partition  
 319 that contains the line  $L_i$ . We then partition  $S$  into two vertical sub-strips by the line  $L_i$ .  
 320 When the partitioning algorithm terminates, every vertical strip contains exactly one active  
 321 column.



■ **Figure 1** An Illustration of partitioning tree and the corresponding sequence  $\sigma = (L_1, \dots, L_7)$ . Strip  $S(v)$  corresponds to node  $v$  that owns line  $L_6$ .

322 This partitioning process can be naturally described by a binary tree  $T = T(\sigma)$ , that we  
 323 call a *partitioning tree* associated with the ordering  $\sigma$  (see Figure 1). Each node  $v \in V(T)$   
 324 is associated with a vertical strip  $S(v)$  of the bounding box  $B$ . The strip  $S(r)$  of the root



325 vertex  $r$  of  $T$  is the bounding box  $B$ . For every inner vertex  $v \in V(T)$ , if  $S = S(v)$  is the  
 326 vertical strip associated with  $v$ , and if  $L \in \mathcal{L}$  is the first line in  $\sigma$  that lies strictly in  $S$ , then  
 327 line  $L$  partitions  $S$  into two sub-strips, that we denote by  $S_L$  and  $S_R$ . Vertex  $v$  then has two  
 328 children, whose corresponding strips are  $S_L$  and  $S_R$  respectively. We say that  $v$  *owns* the  
 329 line  $L$ , and we denote  $L = L(v)$ . For each leaf node  $v$ , the corresponding strip  $S(v)$  contains  
 330 exactly one active column of  $X$ , and  $v$  does not own any line of  $\mathcal{L}$ . For each vertex  $v \in V(T)$ ,  
 331 let  $N(v) = |X \cap S(v)|$  be the number of points from  $X$  that lie in  $S(v)$ , and let  $\text{width}(v)$  be  
 332 the width of the strip  $S(v)$ . Given a partition tree  $T$  for point set  $X$ , we refer to the vertical  
 333 strips in  $\{S(v)\}_{v \in T}$  as  $T$ -strips.

## 334 2.4 The WB-1 Bound

335 The WB-1 bound<sup>2</sup> is defined with respect to an ordering (or a permutation)  $\sigma$  of the auxiliary  
 336 columns, or, equivalently, with respect to the partitioning tree  $T(\sigma)$ . It will be helpful to  
 337 keep both these views in mind. In this paper, we will make a clear distinction between a  
 338 weak variant of the WB-1 bound, as defined by Wilber himself in [25] and a strong variant,  
 339 as mentioned in [16].

340 Let  $X$  be a semi-permutation, and let  $\mathcal{L}$  be the corresponding set of auxiliary columns.  
 341 Consider an arbitrary fixed ordering  $\sigma$  of columns in  $\mathcal{L}$  and its corresponding partition tree  
 342  $T = T(\sigma)$ . For each inner node  $v \in V(T)$ , consider the set  $X' = X \cap S(v)$  of input points that  
 343 lie in the strip  $S(v)$ , and let  $L(v) \in \mathcal{L}$  be the line that  $v$  owns. We denote  $X' = \{p_1, p_2, \dots, p_k\}$ ,  
 344 where the points are indexed in the increasing order of their  $y$ -coordinates; since  $X$  is a  
 345 semi-permutation, no two points of  $X$  may have the same  $y$ -coordinate. For  $1 \leq j < k$ , we  
 346 say that the ordered pair  $(p_j, p_{j+1})$  of points form a *crossing* of  $L(v)$  iff  $p_j, p_{j+1}$  lie on the  
 347 opposite sides of the line  $L(v)$ . We let  $\text{cost}(v)$  be the total number of crossings of  $L(v)$  by  
 348 the points of  $X \cap S(v)$ . When  $L = L(v)$ , we also write  $\text{cost}(L)$  to denote  $\text{cost}(v)$ . If  $v$  is a  
 349 leaf vertex, then its cost is set to 0.

350 ► **Definition 8** (WB-1 bound). *For any semi-permutation  $X$ , an ordering  $\sigma$  of the auxiliary*  
 351 *columns in  $\mathcal{L}$ , and the corresponding partitioning tree  $T = T_\sigma$ , the (weak) WB-1 bound of  $X$*   
 352 *with respect to  $\sigma$  is:  $\text{WB}_\sigma(X) = \text{WB}_T(X) = \sum_{v \in V(T)} \text{cost}(v)$ . The strong WB-1 bound of*  
 353  *$X$  is  $\text{WB}(X) = \max_\sigma \text{WB}_\sigma(X)$ , where the maximum is taken over all permutations  $\sigma$  of the*  
 354 *lines in  $\mathcal{L}$ .*

355 It is well known that the WB-1 bound is a lower bound on the optimal solution cost:

356 ▷ **Claim 9.** For any semi-permutation  $X$ ,  $\text{WB}(X) \leq 2 \cdot \text{OPT}(X)$ .

357 The original proof of this fact is due to Wilber [25], which was later presented in the geometric  
 358 view by Demaine et al. [9], via the notion of *independent rectangles*.

## 359 3 Geometric Decomposition Theorems

360 In this section, we develop several technical tools that will allow us to decompose a given  
 361 instance into a number of sub-instances. We then analyze the optimal solution costs and the

<sup>2</sup> Also called Interleaving bound [10], the first Wilber bound, “interleave lower bound” [25], or alternation bound [16]

362 Wilber bound values for the resulting subinstances.

363 **Split Instances.** Consider a semi-permutation  $X$  and its partitioning tree  $T$ . Let  $U \subseteq V(T)$   
 364 be a collection of vertices of the tree  $T$ , such that the strips  $\{S(v)\}_{v \in U}$  partition the bounding  
 365 box. In other words, every root-to-leaf path in  $T$  must contain exactly one vertex of  $U$ . We  
 366 now define splitting an instance  $X$  via the set  $U$  of vertices of  $T$ .

367 ► **Definition 10 (A Split).** A split of  $(X, T)$  at  $U$  is a collection of instances  $\{X^c, \{X_v^s\}_{v \in U}\}$ ,  
 368 defined as follows.

- 369 ■ For each vertex  $v \in U$ , instance  $X_v^s$  is called a **strip instance**, and it contains all points  
 370 of  $X$  that lie in the interior of the strip  $S(v)$ .
- 371 ■ Instance  $X^c$  is called a **compressed instance**, and it is obtained from  $X$  by collapsing,  
 372 for every vertex  $v \in U$ , all active columns in the strip  $S(v)$  into a single column.

373 We also partition the tree  $T$  into sub-trees that correspond to the new instances: for every  
 374 vertex  $v \in U$ , we let  $T_v$  be the sub-tree of  $T$  rooted at  $v$ . Observe that  $T_v$  is a partitioning  
 375 tree for instance  $X_v^s$ . The tree  $T^c$  is obtained from  $T$  by deleting from it, for all  $v \in U$ , all  
 376 vertices of  $V(T_v) \setminus \{v\}$ . It is easy to verify that  $T^c$  is a valid partitioning tree for instance  $X^c$ .  
 377 The following observation, whose proof appears in the full version of the paper, establishes  
 378 several basic properties of a split. Recall that, given an instance  $X$ ,  $r(X)$  and  $c(X)$  denote  
 379 the number of active rows and active columns in  $X$ , respectively.

380 ► **Observation 11.** If  $X$  is a semi-permutation, then the following properties hold for any  
 381  $(X, T)$ -split at  $U$ :

- 382 ■  $\sum_{v \in U} r(X_v^s) = r(X)$
- 383 ■  $\sum_{v \in U} c(X_v^s) = c(X)$
- 384 ■  $c(X^c) \leq |U|$
- 385 ■  $\sum_{v \in U} \text{WB}_{T_v}(X_v^s) + \text{WB}_{T^c}(X^c) = \text{WB}_T(X)$ .

386 The first property holds since  $X$  is a semi-permutation. In order to establish the last property,  
 387 consider any vertex  $x \in V(T)$ , and let  $T' \in \{T^c\} \cup \{T_v\}_{v \in U}$  be the new tree to which  $v$   
 388 belongs; if  $x \in U$ , then we set  $T' = T_x$ . It is easy to see that the cost of  $v$  in tree  $T'$  is the  
 389 same as its cost in the tree  $T$  (recall that the cost of a leaf vertex is 0). The last property  
 390 can be viewed as a “perfect decomposition” property of the weak WB-1 bound. We will show  
 391 below an (approximate) decomposition property of strong WB-1 bound.

392 **Splitting by Lines.** We can also define the splitting with respect to any subset  $\mathcal{L}' \subseteq \mathcal{L}$  of  
 393 the auxiliary columns for  $X$ , analogously: Notice that the lines in  $\mathcal{L}'$  partition the bounding  
 394 box  $B$  into a collection of internally disjoint strips, that we denote by  $\{S'_1, \dots, S'_k\}$ . We can  
 395 then define the strip instances  $X_i^s$  as containing all vertices of  $X \cap S_i$  for all  $1 \leq i \leq k$ , and  
 396 the compressed instance  $X^c$ , that is obtained by collapsing, for each  $1 \leq i \leq k$ , all active  
 397 columns that lie in strip  $S_i$ , into a single column. We also call these resulting instances a  
 398 split of  $X$  by  $\mathcal{L}'$ .

399 We can also consider an arbitrary ordering  $\sigma$  of the lines in  $\mathcal{L}$ , such that the lines of  $\mathcal{L}'$   
 400 appear at the beginning of  $\sigma$ , and let  $U \subseteq V(T(\sigma))$  contain all vertices  $u$  for which the strip  
 401  $S(u)$  is in  $\{S_i\}_{1 \leq i \leq k}$ . If we perform a split of  $(X, T)$  at  $U$ , we obtain exactly the same strip  
 402 instances  $X_1^s, \dots, X_k^s$ , and the same compressed instance  $X^c$ .

## 33:10 Pinning Down the Strong Wilber 1 Bound

403 **Decomposition Theorem for OPT.** The following theorem gives a crucial decomposition  
404 property of OPT. The theorem is used in our algorithm for Min-Sat, and its proof appears  
405 in the full version of the paper.

406 ► **Theorem 12.** *Let  $X$  be a semi-permutation, let  $T$  be any partitioning tree for  $X$ , let  
407  $U \subseteq V(T)$  be a subset of vertices of  $T$  such that the strips in  $\{S(v) \mid v \in U\}$  partition the  
408 bounding box, and let  $\{X^c, \{X_v^s\}_{v \in U}\}$  be an  $(X, T)$ -split at  $U$ . Then:*

$$\sum_{v \in U} \text{OPT}(X_v^s) + \text{OPT}(X^c) \leq \text{OPT}(X).$$

409 **Decomposition Theorem for the Strong WB-1 bound.** We also prove, in the full version  
410 of the paper, the following theorem about the strong WB-1 bound, that we use several times  
411 in our negative result.

412 ► **Theorem 13.** *Let  $X$  be a semi-permutation and let  $T$  be a partitioning tree for  $X$ . Let  
413  $U \subseteq V(T)$  be a set of vertices of  $T$  such that the strips in  $\{S(v) \mid v \in U\}$  partition the  
414 bounding box. Let  $\{X^c, \{X_v^s\}_{v \in U}\}$  be the split of  $(X, T)$  at  $U$ . Then:*

$$\text{WB}(X) \leq 4\text{WB}(X^c) + 8 \sum_{v \in U} \text{WB}(X_v^s) + O(|X|).$$

415 This result is somewhat surprising. One can think of the expression  $\text{WB}(X^c) + \sum_{v \in U} \text{WB}(X_v^s)$   
416 as a WB-1 bound obtained by first cutting along the lines that serve as boundaries of the  
417 strips  $S(v)$  for  $v \in U$ , and then cutting the individual strips. However,  $\text{WB}(X)$  is the  
418 maximum of  $\text{WB}_T(X)$  obtained over all trees  $T$ , including those that do not obey this  
419 partitioning order. The proofs of both Theorems 12 and 13 are given in the full version.

## 420 **4 Separation of OPT and the Strong Wilber Bound**

421 In this section we present our negative results, proving Theorem 1. We start by defining  
422 several basic tools used in our construction in Section 4.1. From Section 4.2 onward, we  
423 describe our construction and its analysis.

### 424 **4.1 Basic Tools**

425 **Monotonically Increasing Sequence.** We say that an input set  $X$  of points is *monotonically*  
426 *increasing* iff  $X$  is a permutation, and moreover for every pair  $p, p' \in X$  of points, if  $p.x < p'.x$ ,  
427 then  $p.y < p'.y$  must hold. It is well known that the value of the optimal solution of  
428 monotonically increasing sequences is low, and we exploit this fact in our negative results.

429 ► **Observation 14.** *If  $X$  is a monotonically increasing set of points, then  $\text{OPT}(X) \leq |X| - 1$ .*

430 **Bit Reversal Sequence (BRS).** We use the geometric variant of BRS, which is more intuitive  
431 and easier to argue about. Let  $\mathcal{R} \subseteq \mathbb{N}$  and  $\mathcal{C} \subseteq \mathbb{N}$  be sets of integers (which are supposed to  
432 represent sets of active rows and columns.) The instance  $\text{BRS}(i, \mathcal{R}, \mathcal{C})$  is only defined when  
433  $|\mathcal{R}| = |\mathcal{C}| = 2^i$ . It contains  $2^i$  points, and it is a permutation, whose sets of active rows

434 and columns are exactly  $\mathcal{R}$  and  $\mathcal{C}$  respectively; so  $|\mathcal{R}| = |\mathcal{C}| = 2^i$ . We define the instance  
 435 recursively. The base of the recursion is instance  $\text{BRS}(0, \{C\}, \{R\})$ , containing a single point  
 436 at the intersection of row  $R$  and column  $C$ . Assume now that we have defined, for all  
 437  $1 \leq i' \leq i$ , and any sets  $\mathcal{R}', \mathcal{C}'$  of  $2^{i'}$  integers, the corresponding instance  $\text{BRS}(i', \mathcal{R}', \mathcal{C}')$ . We  
 438 define instance  $\text{BRS}(i+1, \mathcal{R}, \mathcal{C})$ , where  $|\mathcal{R}| = |\mathcal{C}| = 2^{i+1}$ , as follows.

439 Consider the columns in  $\mathcal{C}$  in their natural left-to-right order, and define  $\mathcal{C}_{left}$  to be the  
 440 first  $2^i$  columns and  $\mathcal{C}_{right} = \mathcal{C} \setminus \mathcal{C}_{left}$ . Denote  $\mathcal{R} = \{R_1, \dots, R_{2^{i+1}}\}$ , where the rows  
 441 are indexed in their natural bottom to top order, and let  $\mathcal{R}_{even} = \{R_2, R_4, \dots, R_{2^{i+1}}\}$   
 442 and  $\mathcal{R}_{odd} = \{R_1, R_3, \dots, R_{2^{i+1}-1}\}$  be the sets of all even-indexed and all odd-indexed  
 443 rows, respectively. Notice that  $|\mathcal{C}_{left}| = |\mathcal{C}_{right}| = |\mathcal{R}_{even}| = |\mathcal{R}_{odd}| = 2^i$ . The instance  
 444  $\text{BRS}(i+1, \mathcal{R}, \mathcal{C})$  is defined to be  $\text{BRS}(i, \mathcal{R}_{odd}, \mathcal{C}_{left}) \cup \text{BRS}(i, \mathcal{R}_{even}, \mathcal{C}_{right})$ .

445 For  $n = 2^i$ , we denote by  $\text{BRS}(n)$  the instance  $\text{BRS}(i, \mathcal{C}, \mathcal{R})$ , where  $\mathcal{C}$  contains all columns  
 446 with integral  $x$ -coordinates from 1 to  $n$ , and  $\mathcal{R}$  contains all rows with integral  $y$ -coordinates  
 447 from 1 to  $n$ ; see Figure 2 for an illustration.

448 It is well-known that, if  $X$  is a bit-reversal sequence on  $n$  points, then  $\text{OPT}(X) \geq \Omega(n \log n)$ .

449  $\triangleright$  **Claim 15.** Let  $X = \text{BRS}(i, \mathcal{C}, \mathcal{R})$ , for any  $i \geq 0$  and any sets  $\mathcal{C}$  and  $\mathcal{R}$  of columns and rows,  
 450 respectively, with  $|\mathcal{R}| = |\mathcal{C}| = 2^i$ . Then  $|X| = 2^i$ , and  $\text{OPT}(X) \geq \frac{\text{WB}(X)}{2} \geq \frac{|X|(\log |X| - 2) + 1}{2}$ .

451 Next, we present two additional technical tools that we use in our construction.

452 **Exponentially Spaced Columns.** Recall that we defined the bit reversal instance  $\text{BRS}(\ell, \mathcal{R}, \mathcal{C})$ ,  
 453 where  $\mathcal{R}$  and  $\mathcal{C}$  are sets of  $2^\ell$  rows and columns, respectively, that serve in the resulting  
 454 instance as the sets of active rows and columns; the instance contains  $n = 2^\ell$  points. In  
 455 the Exponentially-Spaced BRS instance  $\text{ES-BRS}(\ell, \mathcal{R})$ , we are still given a set  $\mathcal{R}$  of  $2^\ell$  rows  
 456 that will serve as active rows in the resulting instance, but we define the set  $\mathcal{C}$  of columns in  
 457 a specific way. For an integer  $i$ ,  $C_i$  be the column whose  $x$ -coordinate is  $i$ . We then let  $\mathcal{C}$   
 458 contain, for each  $0 \leq i < 2^\ell$ , the column  $C_{2^i}$ . Denoting  $N = 2^n = 2^{2^\ell}$ , the  $x$ -coordinates of  
 459 the columns in  $\mathcal{C}$  are  $\{1, 2, 4, 8, \dots, N/2\}$ . The instance is then defined to be  $\text{BRS}(\ell, \mathcal{R}, \mathcal{C})$  for  
 460 this specific set  $\mathcal{C}$  of columns. Notice that the instance contains  $n = \log N = 2^\ell$  input points.

461 It is easy to see that any point set  $X = \text{ES-BRS}(\ell, \mathcal{R})$  satisfies  $\text{OPT}(X) = \Omega(n \log n)$ . We  
 462 remark that this idea of exponentially spaced columns is inspired by the instance used by  
 463 Iacono [16] to prove a gap between the weak WB-1 bound and  $\text{OPT}(X)$ . However, Iacono's  
 464 instance is tailored to specific partitioning tree  $T$ , and it is clear that there is another  
 465 partitioning tree  $T'$  with  $\text{OPT}(X) = \Theta(\text{WB}_{T'}(X))$ . Therefore, this instance does not give a  
 466 separation result for the strong WB-1 bound, and in fact it does not provide negative results  
 467 for the weak WB-1 bound when the input point set is a permutation.

468 **Cyclic Shift of Columns.** Suppose we are given a point set  $X$ , and let  $\mathcal{C}' = \{C_0, \dots, C_{N-1}\}$   
 469 be any set of columns indexed in their natural left-to-right order, such that all points of  $X$   
 470 lie on columns of  $\mathcal{C}'$  (but some columns may contain no points of  $X$ ). Let  $0 \leq s < N$  be any  
 471 integer. We denote by  $X^s$  a cyclic shift of  $X$  by  $s$  units, obtained as follows. For every point  
 472  $p \in X$ , we add a new point  $p^s$  to  $X^s$ , whose  $y$ -coordinate is the same as that of  $p$ , and whose  
 473  $x$ -coordinate is  $p.x + s \pmod N$ . In other words, we shift the point  $p$  by  $s$  steps to the right  
 474 in a circular manner. Equivalently, we move the last  $s$  columns of  $\mathcal{C}'$  to the beginning of the  
 475 instance. The following claim, whose proof appears in the full version of the paper, shows  
 476 that the value of the optimal solution does not decrease significantly in the shifted instance.

477  $\triangleright$  Claim 16. Let  $X$  be any point set that is a semi-permutation. Let  $0 \leq s < N$  be a shift  
 478 value, and let  $X' = X^s$  be the instance obtained from  $X$  by a cyclic shift of its points by  $s$   
 479 units to the right. Then  $\text{OPT}(X') \geq \text{OPT}(X) - |X|$ .

480 **4.2 Construction of the Bad Instance**

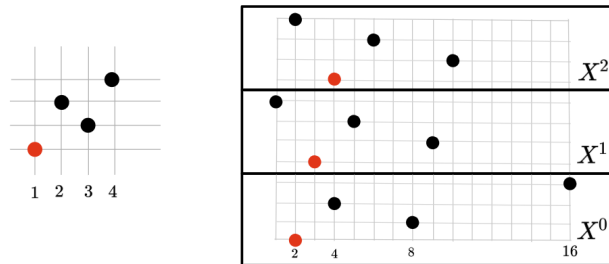
481 We construct two instances: instance  $\hat{X}$  on  $N^*$  points, that is a semi-permutation (but is  
 482 somewhat easier to analyze), and instance  $X^*$  in  $N^*$  points, which is a permutation; the  
 483 analysis of instance  $X^*$  heavily relies on the analysis of instance  $\hat{X}$ . We will show that the  
 484 optimal solution value of both instances is  $\Omega(N^* \log \log N^*)$ , but the cost of the Wilber  
 485 Bound is at most  $O(N^* \log \log \log N^*)$ . Our construction uses the following three parameters.  
 486 We let  $\ell \geq 1$  be an integer, and we set  $n = 2^\ell$  and  $N = 2^n$ .

487 **First Instance.** We now construct our first final instance  $\hat{X}$ , which is a semi-permutation  
 488 containing  $N$  columns. Intuitively, we create  $N$  instances  $X^0, X^1, \dots, X^{N-1}$ , where instance  
 489  $X^s$  is an exponentially-spaced BRS instance that is shifted by  $s$  units. We then stack these  
 490 instances on top of one another in this order.

491 Formally, for all  $0 \leq j \leq N - 1$ , we define a set  $\mathcal{R}_j$  of  $n$  consecutive rows with integral  
 492 coordinates, such that the rows of  $\mathcal{R}_0, \mathcal{R}_1, \dots, \mathcal{R}_{N-1}$  appear in this bottom-to-top order.  
 493 Specifically, set  $\mathcal{R}_j$  contains all rows whose  $y$ -coordinates are in  $\{jn + 1, jn + 2, \dots, (j + 1)n\}$ .

494 For every integer  $0 \leq s \leq N - 1$ , we define a set of points  $X^s$ , which is a cyclic shift of  
 495 instance  $\text{ES-BRS}(\ell, \mathcal{R}_s)$  by  $s$  units. Recall that  $|X^s| = 2^\ell = n$  and that the points in  $X^s$   
 496 appear on the rows in  $\mathcal{R}_s$  and a set  $\mathcal{C}_s$  of columns, whose  $x$ -coordinates are in  $\{(2^j + s)$   
 497  $\text{mod } N : 0 \leq j < n\}$ . We then let our final instance be  $\hat{X} = \bigcup_{s=0}^{N-1} X^s$ . From now on, we  
 498 denote  $N^* = |\hat{X}|$ . Recall that  $|N^*| = N \cdot n = N \log N$ .

499 Observe that the number of active columns in  $\hat{X}$  is  $N$ . Since the instance is symmetric  
 500 and contains  $N^* = N \log N$  points, every column contains exactly  $\log N$  points. Each row  
 501 contains exactly one point, so  $\hat{X}$  is a semi-permutation. (See Figure 2 for an illustration).



■ **Figure 2** An illustration of our construction. The figure on the left shows the instance  $\text{BRS}(2, \{1, 2, 3, 4\}, \{1, 2, 3, 4\})$ . The figure on the right combines three copies  $X^0, X^1, X^2$  of the corresponding exponentially-spaced instance, with horizontal shifts of 0, 1, and 2, respectively. The red points show how copies of the same point in different sub-instances.

502 Lastly, we need the following bound on the value of the optimal solution of instance  $\hat{X}$ .

503  $\blacktriangleright$  **Observation 17.**  $\text{OPT}(\hat{X}) = \Omega(N^* \log \log N^*)$

504 **Proof.** From Claims 15 and 16, for each  $0 \leq s \leq N - 1$ , each sub-instance  $X^s$  has

505  $\text{OPT}(X^s) \geq \Omega(n \log n) = \Omega(\log N \log \log N)$ . Therefore,  $\text{OPT}(\hat{X}) \geq \sum_{s=0}^{N-1} \text{OPT}(X^s) =$   
 506  $\Omega(N \log N \log \log N) = \Omega(N^* \log \log N^*)$  (we have used the fact that  $N^* = N \log N$ ). ◀

507 **Second Instance.** We now construct our second and final instance,  $X^*$ , that is a permutation.  
 508 In order to do so, we start with the instance  $\hat{X}$ , and, for every active column  $C$  of  $\hat{X}$ , we  
 509 create  $n = \log N$  new columns (that we view as copies of  $C$ ),  $C^1, \dots, C^{\log N}$ , which replace  
 510 the column  $C$ . We denote this set of columns by  $\mathcal{B}(C)$ , and we refer it as the *block of*  
 511 *columns representing*  $C$ . Recall that the original column  $C$  contains  $\log N$  input points of  
 512  $\hat{X}$ . We place each such input point on a distinct column of  $\mathcal{B}(C)$ , so that the points form a  
 513 monotonically increasing sequence (see the definition in Section 4.1). This completes the  
 514 definition of the final instance  $X^*$ . We obtain the following immediate bound on the optimal  
 515 solution cost of instance  $X^*$ .

516 ▷ **Claim 18.**  $\text{OPT}(X^*) \geq \text{OPT}(\hat{X}) = \Omega(N^* \log \log N^*)$ .

### 517 4.3 Upper Bound for $\text{WB}(\hat{X})$

518 In this section we prove the following theorem.

519 ▶ **Theorem 19.**  $\text{WB}(\hat{X}) \leq O(N^* \log \log \log N^*)$ .

520 In order to prove the theorem, consider again the instance  $\hat{X}$ . Recall that it consists of  $N$   
 521 instances  $X^0, X^1, \dots, X^{N-1}$  that are stacked on top of each other vertically in this order. We  
 522 rename these instances as  $X_1, X_2, \dots, X_N$ , so  $X_j$  is exactly  $\text{ES-BRS}(\log N)$ , that is shifted  
 523 by  $(j - 1)$  units to the right. Recall that  $|\hat{X}| = N^* = N \log N$ , and each instance  $X_s$   
 524 contains exactly  $\log N$  points. We denote by  $\mathcal{C}$  the set of  $N$  columns, whose  $x$ -coordinates  
 525 are  $1, 2, \dots, N$ . All points of  $\hat{X}$  lie on the columns of  $\mathcal{C}$ . For convenience, for  $1 \leq j \leq N$ , we  
 526 denote by  $C_j$  the column of  $\mathcal{C}$  whose  $x$ -coordinate is  $j$ .

527 Let  $\sigma$  be any ordering of the auxiliary columns in  $\mathcal{L}$ , and let  $T = T_\sigma$  be the corresponding  
 528 partitioning tree. It is enough to show that, for any such ordering  $\sigma$ , the value of  $\text{WB}_\sigma(\hat{X})$   
 529 is bounded by  $O(N^* \log \log \log N^*)$ . Recall that  $\text{WB}_\sigma(\hat{X})$  is the sum, over all vertices  
 530  $v \in V(T)$ , of  $\text{cost}(v)$ . The value of  $\text{cost}(v)$  is defined as follows. If  $v$  is a leaf vertex, then  
 531  $\text{cost}(v) = 0$ . Otherwise, let  $L = L(v)$  be the line of  $\mathcal{L}$  that  $v$  owns. Index the points  
 532 in  $X \cap S(v)$  by  $q_1, \dots, q_z$  in their bottom-to-top order. A consecutive pair  $(q_j, q_{j+1})$  of  
 533 points is a *crossing* iff they lie on different sides of  $L(v)$ . We distinguish between the two  
 534 types of crossings that contribute towards  $\text{cost}(v)$ . We say that the crossing  $(q_j, q_{j+1})$  is of  
 535 *type-1* if both  $q_j$  and  $q_{j+1}$  belong to the same shifted instance  $X_s$  for some  $0 \leq s \leq N - 1$ .  
 536 Otherwise, they are of *type-2*. Note that, if  $(q_j, q_{j+1})$  is a crossing of type 2, with  $q_j \in X_s$   
 537 and  $q_{j+1} \in X_{s'}$ , then  $s, s'$  are not necessarily consecutive integers, as it is possible that for  
 538 some indices  $s''$ ,  $X_{s''}$  has no points that lie in the strip  $S(v)$ . We now let  $\text{cost}_1(v)$  be the  
 539 total number of type-1 crossings of  $L(v)$ , and  $\text{cost}_2(v)$  the total number of type-2 crossings.  
 540 Note that  $\text{cost}(v) = \text{cost}_1(v) + \text{cost}_2(v)$ . We also define  $\text{cost}_1(\sigma) = \sum_{v \in V(T)} \text{cost}_1(v)$  and  
 541  $\text{cost}_2(\sigma) = \sum_{v \in V(T)} \text{cost}_2(v)$ . Clearly,  $\text{WB}_\sigma(\hat{X}) = \text{cost}_1(\sigma) + \text{cost}_2(\sigma)$ . In the full version of  
 542 the paper, we prove the following two theorems:

543 ▶ **Theorem 20.** For every ordering  $\sigma$  of the auxiliary columns in  $\mathcal{L}$ ,  $\text{cost}_1(\sigma) \leq O(N^* \log \log \log N^*)$ .

544 ▶ **Theorem 21.** For every vertex  $v \in V(T)$ ,  $\text{cost}_2(v) \leq O(\log N) + O(\text{cost}_1(v))$ .

### 33:14 Pinning Down the Strong Wilber 1 Bound

545 Notice that from the latter theorem, we get that  $\text{cost}_2(\sigma) \leq O(\text{cost}_1(\sigma)) + O(|V(T)| \cdot \log N) =$   
546  $O(N^* \log \log \log N^*) + O(N \log N) = O(N^* \log \log \log N^*)$ . Combining the two theorems  
547 together completes the proof of Theorem 19.

#### 548 4.4 Upper Bound for $\text{WB}(X^*)$

549 In this section we show that  $\text{WB}(X^*) = O(N^* \log \log \log N^*)$ , completing the proof of  
550 Theorem 1. Recall that instance  $X^*$  is obtained from instance  $\hat{X}$  by replacing every active  
551 column  $C$  of  $X^*$  with a block  $\mathcal{B}(C)$  of columns, and then placing the points of  $C$  on the  
552 columns of  $\mathcal{B}(C)$  so that they form a monotone increasing sequence, while preserving their  
553  $y$ -coordinates. The resulting collection of all blocks  $\mathcal{B}(C)$  partitions the set of all active  
554 columns of  $X^*$ . We denote this set of blocks by  $\mathcal{B}_1, \dots, \mathcal{B}_N$ . The idea is to use Theorem 13  
555 in order to bound  $\text{WB}(X^*)$ .

556 Consider a set of lines  $\mathcal{L}'$  (with half-integral  $x$ -coordinates) that partition the bounding box  
557  $B$  into strips, where the  $i$ th strip contains the block  $\mathcal{B}_i$  of columns, so  $|\mathcal{L}'| = (N - 1)$ . We  
558 consider a split of instance  $X^*$  by  $\mathcal{L}'$ : This gives us a collection of strip instances  $\{X_i^s\}_{1 \leq i \leq N}$   
559 and the compressed instance  $X^c$ . Notice that the compressed instance is precisely  $\hat{X}$ , and  
560 each strip instance  $X_i^s$  is a monotone increasing point set.

561 Since each strip instance  $X_i^s$  is monotonously increasing, from Observation 14 and Claim  
562 9, for all  $i$ ,  $\text{WB}(X_i^s) \leq O(\text{OPT}(X_i^s)) \leq O(|X_i^s|)$ . From Theorem 13, we then get that:  
563  $\text{WB}(X^*) \leq 4\text{WB}(\hat{X}) + 8 \sum_i \text{WB}(X_i^s) + O(|X^*|) \leq 4\text{WB}(\hat{X}) + O(|X^*|) \leq O(N^* \log \log \log N^*)$ .

## 564 5 Guillotine Bounds

565 In this section we consider an extension of the Wilber bound which we call the Guillotine  
566 bound. The *Guillotine bound*  $\text{GB}(X)$  extends  $\text{WB}(X)$  by allowing both vertical and horizontal  
567 partitioning lines. Specifically, given the bounding box  $B$ , we let  $L$  be any vertical or  
568 horizontal line crossing  $B$ , that separates  $X$  into two subsets  $X_1$  and  $X_2$ . We define the  
569 number of crossings of  $L$  exactly as before, and then recurse on both sides of  $L$  as before.  
570 This partitioning scheme can be represented by a binary tree  $T$ , where every vertex of the tree  
571 is associated with a rectangular region of the plane. We denote the resulting bound obtained  
572 by using the partitioning tree  $T$  by  $\text{GB}_T(X)$ , and we define  $\text{GB}(X) = \max_T \text{GB}_T(X)$ . We  
573 show that  $\text{GB}$  is a lower bound on the optimal solution cost in the following lemma, whose  
574 proof is deferred to the full version.

575 ► **Lemma 22.** *For any point set  $X$  that is a permutation,  $\text{GB}(X) \leq 2\text{OPT}(X)$ .*

576 The *Consistent Guillotine bound* restricts the Guillotine bound by maximizing only over  
577 partitioning schemes that are “consistent” in the following sense: suppose that the current  
578 partition of the bounding box  $B$ , that we have obtained using previous partitioning lines, is  
579 a collection  $\{R_1, \dots, R_k\}$  of rectangular regions. We need to choose a vertical or a horizontal  
580 line  $L$  that spans the whole bounding box  $B$ , that is,  $L$  intersects the boundary of  $B$  in two  
581 points. Once line  $L$  is chosen, for **every** rectangular region  $R_i$  that intersects  $L$ , we must  
582 partition  $R_i$  into two sub-regions using the line  $L$ , and then count the number of consecutive  
583 pairs of points in  $X \cap R_i$  that cross the line  $L$ . In other words, we must partition all rectangles  
584  $R_1, \dots, R_k$  consistently with respect to the line  $L$ . In contrast, in the Guillotine bound, we  
585 are allowed to partition each area  $R_i$  independently. From the definitions, the value of the

586 Guillotine bound  $\text{GB}(X)$  is always at least as large as the value of the Consistent Guillotine  
 587 bound, denoted by  $\text{cGB}(X)$ , on any input sequence  $X$ , which is at least as large as  $\text{WB}(X)$ .  
 588 We generalize our negative result to the Consistent Guillotine bound in the following theorem,  
 589 whose proof appears in the full version of the paper.

590 ► **Theorem 23.** *For every integer  $n'$ , there is an integer  $n \geq n'$ , and a set  $X$  of points*  
 591 *that is a permutation with  $|X| = n$ , such that  $\text{OPT}(X) \geq \Omega(n \log \log n)$  but  $\text{cGB}(X) \leq$*   
 592  *$O(n \log \log \log n)$ .*

593 Our negative results do not extend to the general GB bound, while our divide-and-conquer  
 594 framework can naturally be adapted to work with GB. We leave open an interesting question  
 595 of establishing the worst-case gap between the value of OPT and that of the Guillotine bound,  
 596 and we hope that combining the Guillotine bound with our algorithmic framework will lead  
 597 to better online and offline approximation algorithms.

## 598 6 The Algorithms

599 In this section we provide the high level intuition for the proof of Theorem 2. A more detailed  
 600 description appears in the Appendix. Both the polynomial time and the sub-exponential  
 601 time algorithms follow the same framework. We start with a high-level overview of this  
 602 framework. For simplicity, assume that the number of active columns in the input instance  
 603  $X$  is an integral power of 2. The key idea is to decompose the input instance into smaller  
 604 sub-instances, using the split instances defined in Section 3. We solve the resulting instances  
 605 recursively and then combine the resulting solutions.

606 Suppose we are given an input point set  $X$  that is a semi-permutation, with  $|X| = m$ , such  
 607 that the number of active columns is  $n$ . We consider a *balanced* partitioning tree  $T$ , where  
 608 for every vertex  $v \in V(T)$ , the line  $L(v)$  that  $v$  owns splits the strip  $S(v)$  in the middle,  
 609 with respect to the active columns that are contained in  $S(v)$ . Therefore, the height of the  
 610 partitioning tree is  $\log n$ .

611 Consider now the set  $U$  of vertices of  $T$  that lie in the middle layer of  $T$ . We consider the  
 612 split of  $(X, T)$  at  $U$ , obtaining a new collection of instances  $(X^c, \{X_i^s\}_{i=1}^k)$  where  $k = \Theta(\sqrt{n})$ .  
 613 Note that each resulting strip instance  $X_i^s$  contains  $\Theta(\sqrt{n})$  active columns, and so does the  
 614 compressed instance  $X^c$ .

615 We recursively solve each such instance and then combine the resulting solutions. The key to  
 616 the algorithm and its analysis is to show that there is a collection  $Z$  of  $O(|X|)$  points, such  
 617 that, if we are given any solution  $Y^c$  to instance  $X^c$ , and, for all  $1 \leq i \leq k$ , any solution  $Y_i$   
 618 to instance  $X_i^s$ , then  $Z \cup Y^c \cup \left(\bigcup_{i=1}^k Y_i\right)$  is a feasible solution to instance  $X$ . We also show  
 619 that the total number of input points that appear in all instances that participate in the  
 620 same recursive level is bounded by  $O(\text{OPT}(X))$ . This ensures that in every recursive level  
 621 we add at most  $O(\text{OPT}(X))$  points to the solution, and the total solution cost is at most  
 622  $O(\text{OPT}(X))$  times the number of the recursive levels, which is bounded by  $O(\log \log n)$ .

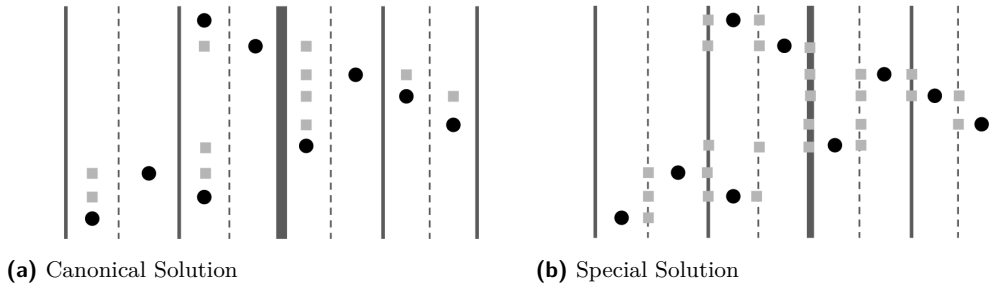
623 In order to obtain the sub-exponential time algorithm, we restrict the recursion to  $D$  levels,  
 624 and then solve each resulting instance directly in time  $r(X)c(X)^{O(c(X))}$ . This approach gives  
 625 an  $O(D)$ -approximation algorithm with running time at most  $\text{poly}(m) \cdot \exp\left(n^{1/2^{\Omega(D)}} \log n\right)$   
 626 as desired. A more detailed description of the algorithm appears in the Appendix.



## 627 — References

- 628 1 G. M. Adelson-Velskiĭ and E. M. Landis. An algorithm for organization of information. *Dokl.*  
629 *Akad. Nauk SSSR*, 146:263–266, 1962.
- 630 2 Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta Inf.*,  
631 1:290–306, 1972. URL: <https://doi.org/10.1007/BF00289509>, doi:10.1007/BF00289509.
- 632 3 Prosenjit Bose, Karim Douïeb, John Iacono, and Stefan Langerman. The power and lim-  
633 itations of static binary search trees with lazy finger. In *Algorithms and Computation -*  
634 *25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Pro-*  
635 *ceedings*, pages 181–192, 2014. URL: [http://dx.doi.org/10.1007/978-3-319-13075-0\\_15](http://dx.doi.org/10.1007/978-3-319-13075-0_15),  
636 doi:10.1007/978-3-319-13075-0\_15.
- 637 4 P. Chalermsook, M. Goswami, L. Kozma, K. Mehlhorn, and T. Saranurak. Greedy is an  
638 almost optimal deque. *WADS*, 2015.
- 639 5 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol  
640 Saranurak. Pattern-avoiding access in binary search trees. In *IEEE 56th Annual Symposium*  
641 *on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*,  
642 pages 410–423, 2015.
- 643 6 R. Chaudhuri and H. Höft. Splaying a search tree in preorder takes linear time. *SIGACT*  
644 *News*, 24(2):88–93, April 1993. URL: <http://doi.acm.org/10.1145/156063.156067>, doi:  
645 10.1145/156063.156067.
- 646 7 R. Cole. On the dynamic finger conjecture for splay trees. part ii: The proof.  
647 *SIAM Journal on Computing*, 30(1):44–85, 2000. URL: <http://dx.doi.org/10.1137/S009753979732699X>,  
648 arXiv:<http://dx.doi.org/10.1137/S009753979732699X>, doi:  
649 10.1137/S009753979732699X.
- 650 8 Richard Cole, Bud Mishra, Jeanette Schmidt, and Alan Siegel. On the dynamic finger  
651 conjecture for splay trees. part i: Splay sorting log n-block sequences. *SIAM J. Comput.*,  
652 30(1):1–43, April 2000. URL: <http://dx.doi.org/10.1137/S0097539797326988>, doi:10.  
653 1137/S0097539797326988.
- 654 9 Erik D. Demaine, Dion Harmon, John Iacono, Daniel M. Kane, and Mihai Pătraşcu. The  
655 geometry of binary search trees. In *SODA 2009*, pages 496–505, 2009. URL: [http://dl.acm.](http://dl.acm.org/citation.cfm?id=1496770.1496825)  
656 [org/citation.cfm?id=1496770.1496825](http://dl.acm.org/citation.cfm?id=1496770.1496825).
- 657 10 Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Pătraşcu. Dynamic optimality  
658 - almost. *SIAM J. Comput.*, 37(1):240–251, 2007. Announced at FOCS’04. URL: <http://dx.doi.org/10.1137/S0097539705447347>, doi:10.1137/S0097539705447347.
- 660 11 Jonathan Derryberry, Daniel Dominic Sleator, and Chengwen Chris Wang. A lower bound  
661 framework for binary search trees with rotations. 2005.
- 662 12 Jonathan C. Derryberry and Daniel D. Sleator. Skip-splay: Toward achieving the unified  
663 bound in the BST model. In Frank Dehne, Marina Gavrilova, Jörg-Rüdiger Sack, and CsabaD.  
664 Toth, editors, *Algorithms and Data Structures*, volume 5664 of *Lecture Notes in Computer*  
665 *Science*, pages 194–205. Springer Berlin Heidelberg, 2009. URL: [http://dx.doi.org/10.1007/](http://dx.doi.org/10.1007/978-3-642-03367-4_18)  
666 [978-3-642-03367-4\\_18](http://dx.doi.org/10.1007/978-3-642-03367-4_18), doi:10.1007/978-3-642-03367-4\_18.
- 667 13 Amr Elmasry. On the sequential access theorem and deque conjecture for splay trees. *Theoretical*  
668 *Computer Science*, 314(3):459 – 466, 2004. URL: [http://www.sciencedirect.com/science/](http://www.sciencedirect.com/science/article/pii/S0304397504000696)  
669 [article/pii/S0304397504000696](http://www.sciencedirect.com/science/article/pii/S0304397504000696), doi:<http://dx.doi.org/10.1016/j.tcs.2004.01.019>.
- 670 14 George F. Georgakopoulos. Chain-splay trees, or, how to achieve and prove loglogn-  
671 competitiveness by splaying. *Inf. Process. Lett.*, 106(1):37–43, 2008.
- 672 15 Dion Harmon. *New Bounds on Optimal Binary Search Trees*. PhD thesis, Massachusetts  
673 Institute of Technology, 2006.
- 674 16 John Iacono. In pursuit of the dynamic optimality conjecture. In *Space-Efficient Data*  
675 *Structures, Streams, and Algorithms*, volume 8066 of *Lecture Notes in Computer Sci-*  
676 *ence*, pages 236–250. Springer Berlin Heidelberg, 2013. URL: [http://dx.doi.org/10.1007/](http://dx.doi.org/10.1007/978-3-642-40273-9_16)  
677 [978-3-642-40273-9\\_16](http://dx.doi.org/10.1007/978-3-642-40273-9_16), doi:10.1007/978-3-642-40273-9\_16.

- 678 17 László Kozma. *Binary search trees, rectangles and patterns*. PhD thesis, Saarland University, Saarbrücken, Germany, 2016. URL: [http://scidok.sulb.uni-saarland.de/volltexte/](http://scidok.sulb.uni-saarland.de/volltexte/2016/6646/)  
679 2016/6646/.
- 680 18 Victor Lecomte and Omri Weinstein. Settling the relationship between wilber’s bounds for  
681 dynamic optimality. *arXiv preprint arXiv:1912.02858*, 2019.
- 682 19 Joan M. Lucas. On the competitiveness of splay trees: Relations to the union-find problem.  
683 *On-line Algorithms, DIMACS Series in Discrete Mathematics and Theoretical Computer*  
684 *Science*, 7:95–124, 1991.
- 685 20 Seth Pettie. Splay trees, Davenport-Schinzel sequences, and the deque conjecture. In *Proceed-*  
686 *ings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA ’08,  
687 pages 1115–1124, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.  
688 URL: <http://dl.acm.org/citation.cfm?id=1347082.1347204>.
- 689 21 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*,  
690 32(3):652–686, 1985. Announced at STOC’83. URL: [http://doi.acm.org/10.1145/3828.](http://doi.acm.org/10.1145/3828.3835)  
691 3835, doi:10.1145/3828.3835.
- 692 22 Rajamani Sundar. On the deque conjecture for the splay algorithm. *Combinatorica*, 12(1):95–  
693 124, 1992. URL: <http://dx.doi.org/10.1007/BF01191208>, doi:10.1007/BF01191208.
- 694 23 Robert Endre Tarjan. Sequential access in splay trees takes linear time. *Combinatorica*, 5(4):367–  
695 378, 1985. URL: <http://dx.doi.org/10.1007/BF02579253>, doi:10.1007/BF02579253.
- 696 24 Chengwen C. Wang, Jonathan C. Derryberry, and Daniel D. Sleator.  $O(\log \log n)$ -competitive  
697 dynamic binary search trees. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium*  
698 *on Discrete Algorithm*, SODA ’06, pages 374–383, Philadelphia, PA, USA, 2006. Society for  
699 Industrial and Applied Mathematics. URL: [http://dl.acm.org/citation.cfm?id=1109557.](http://dl.acm.org/citation.cfm?id=1109557.1109600)  
700 1109600.
- 701 25 R. Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM Journal on*  
702 *Computing*, 18(1):56–67, 1989. Announced at FOCS’86. URL: [http://dx.doi.org/10.1137/](http://dx.doi.org/10.1137/0218004)  
703 0218004, arXiv:<http://dx.doi.org/10.1137/0218004>, doi:10.1137/0218004.  
704



■ **Figure 3** Canonical and  $T$ -special solutions of  $X$ . The input points are shown as circles; the points that belong to the solution  $Y$  are shown as squares.

## 705 A A Detailed Description of the Algorithms

706 In this section we provide additional details for the proof of Theorem 2. Due to lack of space,  
707 some of the proofs are deferred to the full version.

### 708 A.1 Special Solutions

709 Our algorithm will produce feasible solutions of a special form, that we call *special solutions*.  
710 Recall that, given a semi-permutation point set  $X$ , the auxiliary columns for  $X$  are a set  $\mathcal{L}$   
711 of vertical lines with half-integral coordinates. We say that a solution  $Y$  for  $X$  is *special* iff  
712 every point of  $Y$  lies on a row that is active for  $X$ , and on a column of  $\mathcal{L}$ . In particular,  
713 special solutions are by definition non-canonical (see Figure 3 for an illustration). The main  
714 advantage of the special solutions is that they allow us to easily use the divide-and-conquer  
715 approach. We use the following observation, whose proof appears in the full version of the  
716 paper.

717 ► **Observation 24.** *There is an algorithm, that, given a set  $X$  of points that is a semi-*  
718 *permutation, and a canonical solution  $Y$  for  $X$ , computes a special solution  $Y'$  for  $X$ , such*  
719 *that  $|Y'| \leq 2|X| + 2|Y|$ . The running time of the algorithm is  $O(|X| + |Y|)$ .*

720 If  $\sigma$  is any ordering of the auxiliary columns in  $\mathcal{L}$ , and  $T = T_\sigma$  is the corresponding partitioning  
721 tree, then any point set  $Y$  that is a special solution for  $X$  is also called a  *$T$ -special solution*  
722 (although the notion of the solution  $Y$  being special does not depend on the tree  $T$ , this  
723 notion will be useful for us later; in particular, a convenient way of thinking of a  $T$ -special  
724 solution is that every point of  $Y$  must lie on an active row of  $X$ , and on a column that serves  
725 as a boundary for some strip  $S(v)$ , where  $v \in V(T)$ .)

### 726 A.2 Redundant Points and Reduced Point Sets

727 Consider a semi-permutation  $X$ , that we think of as a potential input to the Min-Sat  
728 problem. We denote  $X = \{p_1, \dots, p_m\}$ , where the points are indexed in their natural  
729 bottom-to-top order, so  $(p_1).y < (p_2).y < \dots < (p_m).y$ . A point  $p_i$  is said to be *redundant*,  
730 iff  $(p_i).x = (p_{i+1}).x = (p_{i-1}).x$ . We say that a semi-permutation  $X$  is in the *reduced form* if  
731 there are no redundant points in  $X$ ; in other words, if  $p_{i-1}, p_i, p_{i+1}$  are three points lying  
732 on three consecutive active rows, then their  $x$ -coordinates are not all equal. We use the  
733 following observation and lemma, whose proofs appear in the full version of the paper.

734 ▶ **Observation 25.** *Let  $X$  be a semi-permutation, and let  $X' \subseteq X$  be any point set, that is*  
 735 *obtained from  $X$  by repeatedly removing redundant points. Then  $\text{OPT}(X') \leq \text{OPT}(X)$ .*

736 ▶ **Lemma 26.** *Let  $X$  be a semi-permutation, and let  $X' \subseteq X$  be any point set, that is*  
 737 *obtained from  $X$  by repeatedly removing redundant points. Let  $Y$  be any feasible solution for*  
 738  *$X'$  such that every point of  $Y$  lies on a row that is active for  $X'$ . Then  $Y$  is also a feasible*  
 739 *solution for  $X$ .*

740 From Lemma 26, whenever we need to solve the Min-Sat problem on an instance  $X$ , it is  
 741 sufficient to solve it on a sub-instance, obtained by iteratively removing redundant points  
 742 from  $X$ . We obtain the following immediate corollary of Lemma 26.

743 ▶ **Corollary 27.** *Let  $X$  be a semi-permutation, and let  $X' \subseteq X$  be any point set, that is*  
 744 *obtained from  $X$  by repeatedly removing redundant points. Let  $Y$  be any special feasible*  
 745 *solution for  $X'$ . Then  $Y'$  is also a special feasible solution for  $X$ .*

746 Lastly, we need the following lemma, which is a simple application of the Wilber bound.

747 ▶ **Lemma 28.** *Let  $X$  be a point set that is a semi-permutation in reduced form. Then*  
 748  *$\text{OPT}(X) \geq |X|/4 - 1$ .*

### 749 A.3 The Algorithm Description

750 Suppose we are given an input set  $X$  of points that is a semi-permutation. Let  $T$  be any  
 751 partitioning tree for  $X$ . We say that  $T$  is a *balanced* partitioning tree for  $X$  iff for every  
 752 non-leaf vertex  $v \in V(T)$  the following holds. Let  $v'$  and  $v''$  be the children of  $v$  in the tree  $T$ .  
 753 Let  $X'$  be the set of all input points lying in strip  $S(v)$ , and let  $X''$ ,  $X'''$  be defined similarly  
 754 for  $S(v')$  and  $S(v'')$ . Let  $c$  be the number of active columns in instance  $X'$ , and let  $c'$  and  $c''$   
 755 be defined similarly for  $X''$  and  $X'''$ . Then we require that  $c', c'' \leq \lceil c/2 \rceil$ .

756 Given a partitioning tree  $T$ , we denote by  $\Lambda_i$  the set of all vertices of  $T$  that lie in the  $i$ th  
 757 layer of  $T$  – that is, the vertices whose distance from the root of  $T$  is  $i$  (so the root belongs to  
 758  $\Lambda_0$ ). The *height* of the tree  $T$ , denoted by  $\text{height}(T)$ , is the largest index  $i$  such that  $\Lambda_i \neq \emptyset$ .  
 759 If the height of the tree  $T$  is  $h$ , then we call the set  $\Lambda_{\lceil h/2 \rceil}$  of vertices the *middle layer* of  
 760  $T$ . Notice that, if  $T$  is a balanced partitioning tree for input  $X$ , then its height is at most  
 761  $2 \log c(X)$ .

762 Our algorithm takes as input a set  $X$  of points that is a semi-permutation, a balanced  
 763 partition tree  $T$  for  $X$ , and an integral parameter  $\rho > 0$ .

764 Intuitively, the algorithm uses the splitting operation to partition the instance  $X$  into  
 765 subinstances that are then solved recursively, until it obtains a collection of instances whose  
 766 corresponding partitioning trees have height at most  $\rho$ . We then either employ dynamic  
 767 programming, or use a trivial  $O(\log c(X))$ -approximation algorithm. The algorithm returns  
 768 a special feasible solution for the instance. Recall that the height of the tree  $T$  is bounded  
 769 by  $2 \log c(X) \leq 2 \log n$ . The following two theorems will be used as the recursion basis.

770 ▶ **Theorem 29.** *There is an algorithm called LEAFBST-1 that, given a semi-permutation*  
 771 *instance  $X$  of Min-Sat in reduced form, and a partitioning tree  $T$  for it, produces a feasible*  
 772  *$T$ -special solution for  $X$  of cost at most  $2|X| + 2\text{OPT}(X)$ , in time  $|X|^{O(1)} \cdot c(X)^{O(c(X))}$ .*

## 33:20 Pinning Down the Strong Wilber 1 Bound

773 ► **Theorem 30.** *There is an algorithm called LEAFBST-2 that, given a semi-permutation*  
 774 *instance  $X$  of Min-Sat in reduced form, and a partitioning tree  $T$  for it, produces a feasible*  
 775  *$T$ -special solution of cost at most  $2|X| \text{height}(T)$ , in time  $\text{poly}(|X|)$ .*

776 The proofs of both theorems are deferred to the full version of the paper. We now provide  
 777 a schematic description of our algorithm. Depending on the guarantees that we would like  
 778 to achieve, whenever the algorithm calls procedure LEAFBST, it will call either procedure  
 779 LEAFBST-1 or procedure LEAFBST-2; we specify this later.

RECURSIVEBST( $X, T, \rho$ )

1. Keep removing redundant points from  $X$  until  $X$  is in reduced form.
2. IF  $T$  has height at most  $\rho$ ,
3.     **return** LEAFBST( $X, T$ )
4. Let  $U$  be the set of vertices lying in the middle layer of  $T$ .
5. Compute the split  $(X^c, \{X_v^s\}_{v \in U})$  of  $(X, T)$  at  $U$ .
6. Compute the corresponding sub-trees  $(T^c, \{T_v^s\}_{v \in U})$  of  $T$ .
- 780 7. For each vertex  $v \in U$ , call to RECURSIVEBST with input  $(X_v^s, T_v^s, \rho)$ , and let  $Y_v$  be  
 the solution returned by it.
8. Call RECURSIVEBST with input  $(X^c, T^c, \rho)$ , and let  $\hat{Y}$  be the solution returned by it.
9. Let  $Z$  be a point set containing, for each vertex  $v \in U$ , for each point  $p \in X_v^s$ , two  
 copies  $p'$  and  $p''$  of  $p$  with  $p'.y = p''.y = p.y$ , where  $p'$  lies on the left boundary of  
 $S(v)$ , and  $p''$  lies on the right boundary of  $S(v)$ .
10. **return**  $Y^* = Z \cup \hat{Y} \cup (\bigcup_{v \in U} Y_v)$

### 781 A.4 Analysis

782 We start by showing that the solution that the algorithm returns is  $T$ -special in the following  
 783 observation, whose proof appears in the full version of the paper.

784 ► **Observation 31.** *Assuming that LEAFBST( $X, T$ ) returns a  $T$ -special solution, the solution*  
 785  *$Y^*$  returned by Algorithm RECURSIVEBST( $X, T, \rho$ ) is a  $T$ -special solution.*

786 We next turn to prove that the solution  $Y^*$  computed by Algorithm RECURSIVEBST( $X, T, \rho$ )  
 787 is feasible. In order to do so, we will use the following immediate observation.

788 ► **Observation 32.** *Let  $Y^*$  be the solution returned by Algorithm RECURSIVEBST( $X, T, \rho$ ),*  
 789 *and let  $u \in U$  be any vertex. Then:*

- 790 ■ *Any point  $y \in Y^*$  that lies in the interior of  $S(u)$  must lie on an active row of instance*  
 791  *$X_u^s$ .*
- 792 ■ *Any point  $y \in Y^*$  that lies on the boundary of  $S(u)$  must belong to in  $\hat{Y} \cup Z$ . Moreover,*  
 793 *the points of  $\hat{Y} \cup Z$  may not lie in the interior of  $S(u)$ .*
- 794 ■ *If  $R$  is an active row for instance  $X_u^s$ , then set  $Z$  contains two points, lying on the*  
 795 *intersection of  $R$  with the left and the right boundaries of  $S(u)$ , respectively.*

796 The following theorem, whose proof is deferred to the full version of the paper, shows that  
 797 the algorithm returns a feasible solution.

798 ► **Theorem 33.** *Assume that the recursive calls to Algorithm RECURSIVEBST return a*  
 799 *feasible special solution  $\hat{Y}$  for instance  $X^c$ , and for each  $v \in U$ , a feasible special solution  $Y_v$*

800 for the strip instance  $X_v^s$ . Then the point set  $Y^* = Z \cup \hat{Y} \cup (\bigcup_{v \in U} Y_v)$  is a feasible solution  
801 for instance  $X$ .

802 In order to analyze the solution cost, consider the final solution  $Y^*$  to the input instance  $X$ .  
803 We distinguish between two types of points in  $Y^*$ : a point  $p \in Y^*$  is said to be of type 2 if it  
804 was added to the solution by Algorithm LEAFBST, and otherwise we say that it is of type 1.  
805 We start by bounding the number of points of type 1 in  $Y^*$ .

806  $\triangleright$  **Claim 34.** The number of points of type 1 in the solution  $Y^*$  to the original instance  $X$  is  
807 at most  $O(\log(\text{height}(T)/\rho)) \cdot \text{OPT}(X)$ .

808 **Proof.** Observe that the number of recursive levels is bounded by  $\lambda = O(\log(\text{height}(T)/\rho))$ .  
809 This is since, in every recursive level, the heights of all trees decrease by a constant factor,  
810 and we terminate the algorithm once the tree heights are bounded by  $\rho$ . For each  $1 \leq i \leq \lambda$ ,  
811 let  $\mathcal{X}_i$  be the collection of all instances in the  $i$ th recursive level, where the instances are in  
812 the reduced form. Notice that the only points that are added to the solution by Algorithm  
813 RECURSIVEBST directly are the points in the sets  $Z$ . The number of such points added  
814 at recursive level  $i$  is bounded by  $\sum_{X' \in \mathcal{X}_i} 2|X'|$ . It is now sufficient to show that for all  
815  $1 \leq i \leq \lambda$ ,  $\sum_{X' \in \mathcal{X}_i} |X'| \leq O(\text{OPT}(X))$ . We do so using the following observation.

816  $\blacktriangleright$  **Observation 35.** For all  $1 \leq i \leq \lambda$ ,  $\sum_{X' \in \mathcal{X}_i} \text{OPT}(X') \leq \text{OPT}(X)$ .

817 Assume first that the observation is correct. For each instance  $X' \in \mathcal{X}_i$ , let  $T'$  be the  
818 partitioning tree associated with  $X'$ . From Lemma 28,  $|X'| \leq O(\text{OPT}(X'))$ . Therefore, the  
819 number of type-1 points added to the solution at recursive level  $i$  is bounded by  $O(\text{OPT}(X))$ .  
820 We now turn to prove Observation 35.

821 **Proof of Observation 35.** The proof is by induction on the recursive level  $i$ . It is easy to  
822 see that the claim holds for  $i = 1$ , since, from Observation 25, removing redundant points  
823 from  $X$  to turn it into reduced form cannot increase  $\text{OPT}(X)$ .

824 Assume now that the claim holds for level  $i$ , and consider some level- $i$  instance  $X' \in \mathcal{X}_i$ .  
825 Let  $(X^c, \{X_u^s\}_{u \in U})$  be the split of  $(X', T')$  that we have computed. Then, from Theorem 12,  
826  $\sum_{v \in U} \text{OPT}(X_v^s) + \text{OPT}(X^c) \leq \text{OPT}(X')$ . Since, from Observation 25, removing redundant  
827 points from an instance does not increase its optimal solution cost, the observation follows.  $\blacktriangleleft$

828  $\blacktriangleleft$

829 In order to obtain an efficient  $O(\log \log n)$ -approximation algorithm, we set  $\rho$  to be a constant  
830 (it can even be set to 1), and we use algorithm LEAFBST-2 whenever the algorithm calls to  
831 subroutine LEAFBST. Observe that the depth of the recursion is now bounded by  $O(\log \log n)$ ,  
832 and so the total number of type-1 points in the solution is bounded by  $O(\log \log n) \cdot \text{OPT}(X)$ .  
833 Let  $\mathcal{I}$  denote the set of all instances to which Algorithm LEAFBST is applied. Using the  
834 same arguments as in Claim 34,  $\sum_{X' \in \mathcal{I}} |X'| = O(\text{OPT}(X))$ . The number of type-2 points  
835 that Algorithm LEAFBST adds to the solution for each instance  $X' \in \mathcal{I}$  is bounded by  
836  $O(|X'| \cdot \rho) = O(|X'|)$ . Therefore, the total number of type-2 points in the solution is bounded  
837 by  $O(\text{OPT}(X))$ . Overall, we obtain a solution of cost at most  $O(\log \log n) \cdot \text{OPT}(X)$ , and  
838 the running time of the algorithm is polynomial in  $|X|$ .

839 Finally, in order to obtain the sub-exponential time algorithm, we set the parameter  $\rho$  to be  
840 such that the recursion depth is bounded by  $D$ . Since the number of active columns in instance

### 33:22 Pinning Down the Strong Wilber 1 Bound

841  $X$  is  $c(X)$ , and the height of the partitioning tree  $T$  is bounded by  $2 \log c(X)$ , while the depth of  
842 the recursion is at most  $2 \log(\text{height}(T)/\rho)$ , it is easy to verify that  $\rho = O\left(\frac{\log c(X)}{2^{D/2}}\right) = \frac{\log c(X)}{2^{\Omega(D)}}$ .  
843 We use algorithm LEAFBST-1 whenever the algorithm calls to subroutine LEAFBST. As  
844 before, let  $\mathcal{I}$  be the set of all instances to which Algorithm LEAFBST is applied. Using  
845 the same arguments as in Claim 34,  $\sum_{X' \in \mathcal{I}} (|X'| + \text{OPT}(X')) = O(\text{OPT}(X))$ . For each  
846 such instance  $X'$ , Algorithm LEAFBST-1 produces a solution of cost  $O(|X'| + \text{OPT}(X'))$ .  
847 Therefore, the total number of type-2 points in the final solution is bounded by  $O(\text{OPT}(X))$ .  
848 The total number of type-1 points in the solution is therefore bounded by  $O(D) \cdot \text{OPT}(X)$   
849 as before. Therefore, the algorithm produces a factor- $O(D)$ -approximate solution. Finally,  
850 in order to analyze the running time of the algorithm, we first bound the running time of  
851 all calls to procedure LEAFBST-1. The number of such calls is bounded by  $|X|$ . Consider  
852 now some instance  $X' \in \mathcal{I}$ , and its corresponding partitioning tree  $T'$ . Since the height of  
853  $T'$  is bounded by  $\rho$ , we get that  $c(X') \leq 2^\rho \leq 2^{\log c(X)/2^{\Omega(D)}} \leq (c(X))^{1/2^{\Omega(D)}}$ . Therefore,  
854 the running time of LEAFBST-1 on instance  $X'$  is bounded by  $|X'|^{O(1)} \cdot (c(X'))^{O(c(X'))} \leq$   
855  $|X'|^{O(1)} \cdot \exp(O(c(X') \log c(X'))) \leq |X'|^{O(1)} \cdot \exp\left(c(X)^{1/2^{\Omega(D)}} \cdot \log c(X)\right)$ .

856 The running time of the remainder of the algorithm, excluding the calls to LEAFBST-1, is  
857 bounded by  $\text{poly}(|X|)$ . We conclude that the total running time of the algorithm is bounded  
858 by  $|X|^{O(1)} \cdot \exp\left(c(X)^{1/2^{\Omega(D)}} \cdot \log c(X)\right) \leq \text{poly}(m) \cdot \exp\left(n^{1/2^{\Omega(D)}} \cdot \log n\right)$ .