Homework set 2

Note: the homework sets are not for submission. They are designed to help you prepare for the quizzes. It is highly recommended that you solve all problems and write your solutions down formally. Whenever a question asks to design an algorithm for a problem, you should prove its correctness.

1. Given a string X, we denote by $X[i]$ the ith character of X. Given an n-character string A, and two additional strings B and C, we say that string A is an *interleaving* of strings B and C iff we can partition the set $\{1,\ldots,n\}$ of indices into two disjoint subsets $I = \{i_1, i_2, \ldots, i_k\}$ and $J = \{j_1, j_2, \ldots, j_{n-k}\}$, where $i_1 < i_2 < \cdots < i_k$ and $j_1 < j_2 < \cdots < j_{n-k}$ such that:

   - $I \cup J = \{1,\ldots,n\}$
   - the string $(A[i_1], A[i_2], \ldots, A[i_k]) = B$, and the string $(A[j_1], A[j_2], \ldots, A[j_{n-k}]) = C$

   In other words, A is obtained by interleaving the characters of B and C. Design an efficient algorithm, that, given as input strings A, B and C, decides whether A is an interleaving of B and C. Prove the algorithm's correctness and analyze its running time.

2. The input to this problem is a set of n gems. Each gem has an integral value in dollars and is either a ruby or an emerald. Let the sum of the values of the gems be L. The problem is to determine if it is possible to partition of the gems into two sets P and Q, such that each set has the same value, the number of rubies in P is equal to the number of rubies in Q, and the number of emeralds in P is equal to the number of emeralds in Q. Note that a partition means that every gem must be in exactly one of P or Q. Design an algorithm for this problem, whose running time is polynomial in $n + L$. Prove the algorithm's correctness and analyze its running time.

3. We say that a set $A \subset \{1, 2, \ldots, n\}$ is *good* iff for all $1 \leq i \leq n-2$, among the numbers in the triple $T_i = (i, i+1, i+2)$, either one or two members of T_i belong to A. For example, the set $A = \{1, 2, 4, 5\} \subset \{1, 2, \ldots, 6\}$ is good, set $B = \{4, 5\} \subset \{1, 2, \ldots, 6\}$ is not good (none of the members of triple $\{1, 2, 3\}$ belong to B), and $C = \{2, 3, 4, 6\} \subset \{1, 2, \ldots, 6\}$ is not good (all members of triple $\{2, 3, 4\}$ belong C). Using dynamic programming, design an algorithm, that, given n, and a sequence $w_1, \ldots, w_n$ of non-negative numbers, finds a good subset $A \subset \{1, 2, \ldots, n\}$ that maximizes the sum $\sum_{i \in A} w_i$. The running time of the algorithm should be polynomial in n. Describe your algorithm in detail, prove its correctness, and analyze the running time.

4. Consider the following modification of the standard algorithm for incrementing a binary counter.

   Increment($B[0, \ldots, \infty]$)

   - $i \leftarrow 0$
   - While $B[i] = 1$:
     - $B[i] \leftarrow 0$
     - $i \leftarrow i + 1$
   - $B[i] \leftarrow 1$
   - SomethingElse(i)

The only difference from the standard algorithm is the function call at the end, to a black-box subroutine called SomethingElse. Suppose we call Increment n times, starting with counter value 0. The amortized time of each call clearly depends on the running time of SomethingElse. Let $T(i)$ denote the worst-case running time of SomethingElse(i).

(a) What is the amortized time per increment if $T(i) = 0$?

(b) What is the amortized time per increment if $T(i) = 42$?

(c) What is the amortized time per increment if $T(i) = i$?

(d) What is the amortized time per increment if $T(i) = 2^i$?

5. An extendable array is a data structure that stores a sequence of items and supports the following operations:

   - AddToFront(x) adds x to the beginning of the sequence.
   - AddToBack(x) adds x to the end of the sequence.
   - Lookup(k) returns the kth item in the sequence, or Null if the current length of the sequence is less than k.

   Describe a simple data structure that implements extendable array. Your AddToFront and AddToBack algorithms should take $O(1)$ **amortized** time, and your Lookup algorithm should take $O(1)$ **worst-case** time. The data structure should use $O(n)$ space, where n is the **current** length of the sequence.