Computational Complexity - Homework II

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Instructions

1. The homework is due on March 10.
2. Each question is worth 4 points.
3. Attempt as many problems as you can. You will be given partial credit, as per the policy discussed in class.

2 Problems

1. Assume that you are given a row of n cards. Each card has a number x_i written on it. The cards have been placed face down. Your task is to identify a local minimum card, i.e. a card i such that $x_{i-1} \geq x_i \leq x_{i+1}$. The end cards can also be local minima; in these cases, there is only one neighbor comparison that is required. You are permitted to turn over as many cards as you need. Clearly, this problem can be solved by turning over all n cards. Can you accomplish this task by turning over $O(\log n)$ cards?

2. Solve the following recurrence:

$$T(n) = \begin{cases} 0, & \text{if } n = 0 \\ (n-1) + \frac{2}{n}\sum_{r=1}^{n} T(r-1), & \text{otherwise} \end{cases}$$

3. The Maximum Subarray problem is defined as follows: Given an array $\mathbf{A}[1 \cdots n]$ of n integers (with at least one positive element) find a contiguous sub-array within $\mathbf{A}$ which has the largest sum.

   For instance, in the array $\mathbf{A}$ defined by $A[1] = -2$, $A[2] = 1$, $A[3] = -3$, $A[4] = 4$, $A[5] = -1$, $A[6] = 2$, $A[7] = 1$, $A[8] = -5$, $A[9] = 4$, the contiguous subarray with the largest sum is $A[4 \cdots 7]$ with sum $4 + (-1) + 2 + 1 = 6$.

   Design a Divide-and-Conquer algorithm for the Maximum Subarray problem and analyze its running time.

4. (a) Let S denote a finite set. Let I_k denote the set of all subsets of S of cardinality at most k. Argue that (S, I_k) is a matroid.

   (b) Let (S, I) be a matroid. Let I' be defined as follows:

$$I' = \{A' : S - A' \text{ contains some maximal } A \in I\}.$$

   Argue that (S, I') is a matroid.

5. In the Ford-Fulkerson algorithm discussed in class, we can increase the flow along any augmenting path. Suppose that we always use one of the shortest paths from s to t in G_f, where shortest means simply the number of edges, regardless of their capacity. Furthermore, we increase the flow as much as possible, i.e., by the minimum of $\{c_f(e)\}$

over all the edges e appearing in the augmenting path. Show that this improved algorithm finds the maximum flow in at most $O(|V| \cdot |E|) = O(n^3)$ iterations even if the capacities are exponentially large in the number of vertices and edges of the network. Conclude that the Max-Flow problem is in **P**.