Advanced Analysis of Algorithms - Homework II

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

1 Instructions

1. The homework is due on October 13, in class.
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. **Mathematics for Analysis**
   (a) Let X and Y denote two non-negative random variables.
      i. Show that $\mathbf{E}[\max(X,Y)] \leq \mathbf{E}[X] + \mathbf{E}[Y]$.
      ii. Assume that X can only assume the values 0 and 1. Prove that $\mathbf{Var}[X] = \mathbf{E}[X] \cdot \mathbf{E}[1-X]$.
   (b) Solve the following recurrences:
      i. $T(n) = T(\sqrt{n}) + 1$.
      ii. $T(n) = T(\frac{n}{3}) + T(\frac{2 \cdot n}{3}) + \Theta(n)$.

      In both cases, you may assume some constant value for $T(1)$.
2. **Divide and Conquer:**
   (a) Professor Krustowski claims to have discovered a new sorting algorithm. Given an array $\mathbf{A}$ of n numbers, his algorithm breaks the array into 3 equal parts of size $\frac{n}{3}$, viz., the first third, the middle third and the bottom third. It then recursively sorts the first two-thirds of the array, the bottom two-thirds of the array and finally the first two-thirds of the array again. Using mathematical induction, prove that the Professor has indeed discovered a correct sorting algorithm. You may assume the following: The input size n is a power of 3. Additionally, the algorithm sorts by brute-force, when n is exactly 3. Formulate a recurrence relation to describe the complexity of Professor Krustowski's algorithm and obtain tight asymptotic bounds.
   (b) Design a divide-and-conquer algorithm to find the second smallest element in an unsorted array $\mathbf{A}[1 \cdot \cdot n]$. Your algorithm should make at most $(n + \lceil \log n \rceil)$ element to element comparisons.
3. **Randomization:**
   (a) Let $A_1, A_2, \ldots, A_n$ denote a set of events. Argue that:

$$\mathbf{Pr} \cap_{i=1}^{n} A_i = \mathbf{Pr}(A_1) \cdot \mathbf{Pr}(A_2|A_1) \cdot \mathbf{Pr}(A_3|A_1 \cap A_2) \ldots \mathbf{Pr}(A_n| \cap_{i=1}^{n-1} A_i).$$

(b) Devise a Divide-and-Conquer procedure for computing the k^{th} smallest element in an array of n integers. Analyze the asymptotic time complexity of your algorithm. *(Hint: Use the Partition procedure discussed in class.)* Your algorithm can be deterministic or randomized. For maximum points, your algorithm should run in $O(n)$ time.

4. **Greedy:**

(a) In the fractional knapsack problem, you are given n objects $O = \{o_1, o_2, \ldots, o_n\}$ with respective profits $P = \{p_1, p_2, \ldots, p_n\}$ and weights $W = \{w_1, w_2, \ldots w_n\}$. The goal is to pack these objects into a knapsack of capacity M, while maximizing the profit of the items in the knapsack. You are permitted to choose any fraction x_i of object o_i, i.e., $0 \leq x_i \leq 1$. Design a greedy algorithm for this problem. Provide a proof of correctness of your algorithm and analyze its running time.

(b) In class, we discussed the coin-changing problem, where you required to make change for a target value using the fewest number of coins. Assume that the available coins are in the denominations c^i, $i = 1, 2, \ldots, k$ for some $c > 1$ and some fixed $k \geq 1$. Argue that the greedy algorithm provides the optimal solution in this case.

5. **Dynamic Programming:**

(a) In class we discussed the rod-cutting problem and I proposed a dynamic programming based solution. Maryam wanted to solve the problem in top-down fashion. I derived the following equation for the number of callas to CUT-ROD, if we followed Maryam's suggestion:

$$\begin{aligned} T(0) &= 1 \\ T(n) &= 1 + \sum_{j=0}^{n-i} T(j) \end{aligned}$$

Find a closed form solution for $T(n)$. Now consider a modification of the rod-cutting problem where there is a fixed charge of c, each time you cut the rod. That is the revenue associated with a solution is the sum of the prices of the individual pieces minus the costs of making the cuts. How would you modify the dynamic program discussed in class to satisfy this additional constraint?

(b) Assume that you are given a collection $A = \{a_1, a_2, \ldots, a_n\}$ of n positive integers that add up to N. Design an algorithm to check if there exists a set $B \subset A$, such that $\sum_{a_i \in B} a_i = \sum_{a_i \in A-B} a_i$? Argue the correctness of your algorithm and analyze its running time.