Fundamental Algorithms

Homework 3

Due: June 21st, 5:30 PM EST

Instructions

Please answer each **Problem** on a separate page. Submissions must be uploaded to your account on Gradescope by the due date and time above.

Problems to submit

Problem 1 (25 points)

Given the input array A of size n with distinct elements and the positive integer k satisfying $1 \leq k \leq n$, modify the SELECT algorithm to develop an $O(n)$ time randomized algorithm to find all the kth smallest elements of A. In other words, your algorithm must output the 1st smallest, and the 2nd smallest, ..., and the kth smallest elements of A. Note that your algorithm must return these k elements but not necessarily in the order.
You should write the pseudo-code of your algorithm.

Problem 2 (20 points)

We learnt how to solve recursions by using a recursion tree based approach. In this exercise, we learn another approach to solve recursions: using the *Master Theorem*.

Theorem 1. *(Master Theorem) Consider the following recursion*

$$T(n) = aT(n/b) + f(n),$$

with the constants $a \geq 1$ and $b > 1$ and the asymptotically positive function $f(n)$ (i.e., $f(n) > 0$ for large enough values of n).
There are three cases:

(a) If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

(b) If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.

(c) If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large values of n, then $T(n) = \Theta(f(n))$.

Some remarks regarding the Master Theorem:

- The Master Theorem can be proved by drawing the corresponding recursion tree but we do not cover it for the sake of simplicity.
- Note that not all the recursions can be solved by using the Master Theorem (see Example III below).

Let's see some examples:

Example I: Consider the following recursion

$$T(n) = 4T(n/2) + n^{1.5}.$$

We have $a = 4$ and $b = 2$. Choosing $\epsilon = 0.1$ (or any value of ϵ satisfying $\epsilon \leq 0.5$), we get

$$f(n) = n^{1.5} = O(n^{\log_b a - \epsilon}).$$

Thus, according to Case (a) of the Master Theorem, we have $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

Example II: Consider the following recursion

$$T(n) = 2T(n/2) + n^{1.5}.$$

We have $a = 2$ and $b = 2$. Choosing $\epsilon = 0.1$ (or any value of ϵ satisfying $\epsilon \leq 0.5$), we get

$$f(n) = n^{1.5} = \Omega(n^{\log_b a + \epsilon}).$$

Also, choosing $c = 1/\sqrt{2}$ (note that $c < 1$), we get

$$2f(n/2) \leq cf(n),$$

for all n (note that $\leq$ can be even replaced by $=$).
Thus, according to Case (c) of the Master Theorem, we have $T(n) = \Theta(f(n)) = \Theta(n^{1.5})$.

Example III: The Master Theorem cannot be applied to the following recursion (why?)

$$T(n) = T(n-1) + n.$$

For each of the following recursions, if it can be solved with the Master Theorem, use the Theorem to find the explicit answer for $T(n)$. Otherwise, fully justify why the Master Theorem does not apply.

(a) $T(n) = 4T(n/2) + n^2$

(b) $T(n) = 2T(n/2) + \log n$

(c) $T(n) = 0.2T(n/2) + n$

(d) $T(n) = 8T(n/2) + 2^n$

Problem 3 (25 points)

Recall that the bottom-up dynamic programming algorithm for finding the nth Fibonacci number required $\Theta(n)$ extra space. Modify the algorithm to develop a linear-time bottom-up DP approach with $O(1)$ extra space. You have to write the pseudo-code of your algorithm.

Hint: Note that in order to compute $F(i)$, we only need to know $F(i-1)$ and $F(i-2)$, and do not require the previous Fibonacci numbers. Use this remark to replace the auxiliary array of size n used in the algorithm covered in the lecture by two auxiliary variables.

Problem 4 (25+5 points)

Recall *the rod cutting problem* discussed in the lecture: Consider a rod of length n inches and an array of prices $P[1 \ldots n]$, where $P[i]$ denotes the price of any i inches long piece of rod. Now suppose we have to pay a cost of \$1 per cut.
Thus, for example, if we cut the rod into k pieces of lengths $n_1, n_2, \ldots, n_k$, this means that we made $k-1$ cuts, which gives their final selling price as $P[n_1] + \cdots + P[n_k] - (k-1)$.
Let MaxPrice(n) denote the maximum selling price we can get this way among all possible options we have to make the cuts (note that one possible option is to make no cut and sell the whole rod of length n).

(a) Find the recursion that MaxPrice(n) satisfies. In other words, you should write MaxPrice(n) in terms of MaxPrice($n-1$), MaxPrice($n-2$), ..., MaxPrice(0). Fully justify your answer.

(b) Identify the base case for your recursion in part (a) and find its corresponding value. Justify your answer.

(c) Write the pseudo-code for the bottom-up DP algorithm to compute MaxPrice(n). Find and justify the time complexity of your algorithm in the form of $\Theta(.)$.

Remark: You do not need to submit the solution to part (c) this week! Just try to think about it!

Bonus problem

The following problems are optional. They will NOT be graded and they will NOT appear on any of the exams. Work on them only if you are interested. They will help you to develop problem solving skills for your future endeavors.

Bonus Problem 1

Consider the following recursion

$$T(n) = 3T\left(\frac{n-\sqrt{n}}{2}\right) + n^2.$$

(a) Show that the Master Theorem cannot be directly applied to solve $T(n)$.

(b) Replace $\frac{n-\sqrt{n}}{2}$ by a simplified upper bound. Choose the upper bound such that the resulting recursion can be solved by the Master Theorem. The result of this modified recursion gives an upper bound for $T(n)$.

(c) Repeat part (b) to find a lower bound for $T(n)$:
Replace $\frac{n-\sqrt{n}}{2}$ by a simplified lower bound for sufficiently large values of n. Choose the lower bound such that the resulting recursion can be solved by the Master Theorem. The result of this modified recursion gives a lower bound for $T(n)$.

(d) Can you choose the upper and lower bounds for $\frac{n-\sqrt{n}}{2}$ in such a way that the resulting upper and lower bounds for $T(n)$ obtained in parts (b) and (c) match? If so, this gives an answer to $T(n)$ which is of the form $\Theta(.)$.

Bonus Problem 2

Given the input array A of size n with distinct elements and the positive integers k, l, satisfying $1 \leq l \leq k \leq n$, develop an $O(n)$ time randomized algorithm to find the lth smallest, and the $(l+1)$th smallest, ..., and the kth smallest elements of A (Thus, your algorithm must return these $k - l + 1$ elements but not necessarily in the order).