CSE 101: Homework #5

Due on May 16, 2024 at 23:59pm

Professor Jones

Ray Tsai, Kevin Yu

Problem 1

Consider the following divide and conquer algorithm that claims to find an MST when the input is a complete graph G with positive edge weights:

Algorithm Description: Given an undirected complete graph $G = (V, E)$ with positive edge weights where $V = [v_1, \dots, v_n]$,

- If $n = 1$ then return the empty set of edges.
- Otherwise, split the set of vertices into two sets: $V' = [v_1, \dots, v_{\lfloor n/2 \rfloor}]$ and $V'' = [v_{\lfloor n/2 \rfloor} + 1, \dots, v_n]$.
- Create two new graphs $G' = (V', E')$ and $G'' = (V'', E'')$ where $E' \subseteq E$ is the set of edges with both endpoints in V' and $E'' \subseteq E$ is the set of edges with both endpoints in V''.
- Recursively run the algorithm on G' and G'' to get T' and T'', respectively. Find the lightest edge that connects a vertex in T' to a vertex in T'' and call that edge e.
- Return $T' \cup T'' \cup \{e\}$.

Disprove the correctness of this algorithm by giving a counterexample.

Proof. Consider $G = C_4$, where the edge $\{v_3, v_4\}$ has weight 2 and the remaining edges each has weight 1. The algorithm recurses on subgraph G'' with vertex set $V'' = [v_3, v_4]$, so the resulting spanning tree T contains the edge $\{v_3, v_4\}$. Since T has 3 edges with an edges of weight 2, the total cost of T is 4. But then $\{\{2, v_i\} : i \neq 2\} \subset E$ spans G with a total weight of 3, as it only uses edges of weight 1. □

Problem 2

You are given an increasing sequence of integers: $(A[1], A[2], \ldots, A[n])$. Design an algorithm that determines (returns TRUE or FALSE) if there exists an index i such that $A[i] = i$.

Your algorithm should run in $O(\log n)$ time.

Proof. We first give a description of the algorithm.

Algorithm Description:

Let $l = 1$ and $r = n$. While $l < r$: put $m = \lfloor (l+r)/2 \rfloor$. If $A[m] = m$, return TRUE. If $A[m] < m$, put $l = m+1$. Otherwise, put $r = m$. After the loop, if $A[l] = l$, return TRUE. Otherwise, return FALSE.

Justification of Correctness:

Let l_k and r_k denote the value of l and r at the end of the kth iteration of the loop, respectively (0th iteration means before the loop starts). Notice that $r_k \geq r_{k+1} \geq l_{k+1} \geq l_k$, for all $k \geq 0$

We show that for all indices $i < l_k$ and $j > r_k$, $A[i] < i$ and $A[j] > j$ by induction on $k \geq 0$. At the start, $l_k = 1$ and $r_k = n$. Hence, no elements are outside the range of l_k and r_k, and so the base case $k = 0$ is done.

Suppose $k \geq 1$. Assume that for all indices $i < l_{k-1}$ and $j > r_{k-1}$, we have $A[i] < i$ and $A[j] > j$. There are three cases:

Case 1: $A[m] = m$.

The loop terminates without changing the values of l and r. By induction, $A[i] < i$ and $A[j] > j$, for all indices $i < l_{k-1} = l_k$ and $j > r_{k-1} = r_k$.

Case 2: $A[m] < m$.

l_k is set to $m+1$ and $r_k = r_{k-1}$. By induction, $A[j] > j$ for all $j > r_{k-1} = r_k$, so it remains to show that $A[i] < i$ for all $i \leq m$. Since the sequence of integers $(A[1], A[2], \ldots, A[n])$ is strictly increasing, we may observe that

$$A[i] \leq A[m] - (m - i),$$

for all $i \leq m$. But then $A[m] - m < 0$, so indeed

$$A[i] \leq A[m] - (m - i) = (A[m] - m) + i < i,$$

for all $i \leq m$.

Case 3: $A[m] > m$.

r_k is set to m and $l_k = l_{k-1}$. By induction, $A[i] < i$ for all $i < l_{k-1} = l_k$, so it remains to show that $A[j] < j$ for all $j > m$. Since the sequence of integers $(A[1], A[2], \ldots, A[n])$ is strictly increasing, we may observe that

$$A[j] \geq A[m] + (j - m),$$

for all $j > m$. But then $A[m] - m > 0$, so indeed

$$A[j] \geq A[m] + (j - m) = (A[m] - m) + j > j,$$

for all $j > m$.

And this completes the induction. Note that the loop breaks half way only if there exists some $A[m] = m$ and the algorithm returns TRUE. Now suppose the loop is terminated by the natural condition. Since $r_k \geq r_{k+1} \geq l_{k+1} \geq l_k$ for all $k \geq 0$, we must have $l = r$. But then by our induction result, $A[i] \neq i$ for all index $i \neq r$. Hence, there exists $A[i] = i$ for some i if and only if $A[r] = r$, and the result now follows.

Problem 2 continued on next page...

Runtime Analysis:

Since every iteration of the loop cuts out half the current list, the loop will iterate at most $\log n$ times until l meets r, given an input list of size n. Checking and updating l or r only take constant time. Hence, in total, the algorithm has a runtime of $O(\log n)$. $\square$

Problem 3

You are given a list of n ordered pairs $[(x_1, f_1), \ldots, (x_n, f_n)]$. This list describes a list of length $\sum f_i$ that contains f_1 copies of the value x_1, f_2 copies of the value x_2 and so on.

You wish to find the median value of this list in expected runtime of $O(n)$. (You can assume that $\sum f_i$ is odd.)

Proof. We give a description of the algorithm:

Algorithm Description:

Let $\ell([(x_1, f_1), \ldots, (x_u, f_u)])$ denote the length of the list described by $[(x_1, f_1), \ldots, (x_u, f_u)]$, namely $\sum_{i=1}^{u} f_i$.

We first define $Selection(L = [(x_1, f_1), \ldots, (x_m, f_m)], k)$, which takes in a list L of ordered pairs and an integer k, and outputs the kth smallest number in the list described in L:

If $|L| = 1$, return x_1. Otherwise, pick x_v randomly from L. Split L into L_l, $[(x_v, f_v)]$, and L_r, where L_l contains all the ordered pairs with x_i less than x_v and L_r contains the ordered pairs with x_i greater than x_v. If $k \le \ell(L_l)$, return $Selection(L_l, k)$. Else, if $k \le \ell(L_l) + f_v$, return x_v. Otherwise, return $Selection(L_r, k - \ell(L_l) - f_v)$.

Now for finding the median value of the list described in L, we simply run $Selection(L, (\ell(L) + 1)/2)$.

Runtime Analysis:

Since we select the pivot x_v uniformly at random, the input list L will be split into a list L_l of length $v-1$ and a list L_r of length $n-v$. Hence, when we recurse on L_l, L_r, it will take time proportional to $max(v-1, n-v)$. Note that if $\frac{n}{4} \le v - 1 \le \frac{3}{4}n$, then $max(v-1, n-v) \le \frac{3}{4}n$. Otherwise, $\frac{3}{4}n \le max(v-1, n-v) < n$. Let $ET(n)$ denote the expected runtime for *Selection* on a list of length n. It now follows that

$$ET(n) \le \frac{1}{2} ET\left(\frac{3}{4}n\right) + \frac{1}{2} ET(n) + cn,$$

where the cn term derived from the splitting process of L. But then

$$ET(n) \le ET\left(\frac{3}{4}n\right) + cn,$$

and thus

$$ET(n) \in O(n).$$

by the Master Theorem. □

Problem 4

(a) Let $T(n)$ be the runtime of a divide and conquer algorithm on an input of size n. The algorithm has 6 recursive calls each of size $n/4$ and the non-recursive part takes $O(n^{1.5})$ time. Use the Master theorem to find the best Big-Oh runtime.

Proof. We first note that
$$T(n) = 6T(n/4) + cn^{1.5}.$$
By the Master Theorem,
$$T(n) \in O(n^{1.5}),$$
as $6 < 4^{1.5} = 8$. □

(b) Let $R(n)$ be the runtime of a divide and conquer algorithm on an input of size n. The algorithm has 1 recursive call of size $n/2$ and the non-recursive part takes $O(\log n)$ time. Find the best Big-Oh runtime.

Proof. We first note that
$$R(n) = R(n/2) + c\log n.$$
Consider the levels of recurrence of this algorithm. Since the algorithm has 1 recursive call of size $n/2$, there are $\log n$ levels of recurrence, with 1 recursive call per level. It now follows that
$$\begin{aligned}
R(n) &= R(n/2) + c\log n \\
&= \left(R(n/4) + c\log\frac{n}{2}\right) + c\log n \\
&= c\sum_{k=0}^{\log n} \log\frac{n}{2^k} \\
&= c\sum_{k=0}^{\log n} (\log n - k) \\
&= c\log^2 n - c\sum_{k=0}^{\log n} k \\
&= c\log^2 n - \frac{c(\log n + 1)\log n}{2} \\
&\in O\left(\log^2 n\right).
\end{aligned}$$
□

(c) Let $S(n)$ be the runtime of a divide and conquer algorithm on an input of size n. The algorithm has 2 recursive calls each of size $2n/3$ and the non-recursive part takes $O(n)$ time. Find the best Big-Oh runtime.

Proof. We first note that
$$S(n) = 2T(2n/3) + cn.$$
By the Master Theorem,
$$S(n) \in O(n^{\log_{3/2} 2}) = O(n^{\frac{\log 2}{\log 3 - \log 2}}) \approx O(n^{1.71}),$$
as $2 > 3/2$. □