COMP 3711 – Design and Analysis of Algorithms
2024 Fall Semester – Written Assignment 2
Distributed: 9:00 on September 30, 2024
Due: 23:59 on October 11, 2024

Your solution should contain
(i) your name, (ii) your student ID #, and (iii) your email address
at the top of its first page.

Some Notes:

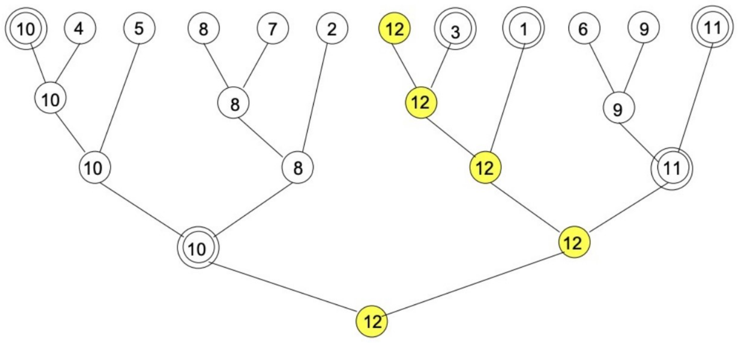
- Please write clearly and briefly. In particular, your solutions should be written or printed on *clean* white paper with no watermarks, i.e., student society paper is not allowed.
- Please also follow the guidelines on doing your own work and avoiding plagiarism as described on the class home page. ***You must acknowledge individuals who assisted you, or sources where you found solutions.*** Failure to do so will be considered plagiarism.
- The term *Documented Pseudocode* means that your pseudocode must contain documentation, i.e., comments, inside the pseudocode, briefly explaining what each part does.
- Many questions ask you to explain things, e.g., what an algorithm is doing, why it is correct, etc. To receive full points, the explanation must also be *understandable* as well as correct.
- Submit a SOFTCOPY of your assignment to Canvas by the deadline. If your submission is a scan of a handwritten solution, make sure that it is of high enough resolution to be easily read. At least 300dpi and possibly denser.

1. (20 points) You are given n numbers, where n is a positive power of 2. Describe an algorithm that finds the largest and second largest numbers in $n + \log_2 n - 2$ comparisons. Explain the correctness of your algorithm. Show that your number of comparisons is indeed $n + \log_2 n - 2$.

   Solution: *1. We use a tournament-style approach to find the largest number. Which means we pair numbers two by two and each pair "compete" again each other and the larger numbers goes to the next level.*

   2. Then we track the numbers that lost to the largest number to find the second largest.

   Look at the following example that was available here.



   You can see here 12 is the largest number. and the second largest number is 11 which was found by looking through the set of numbers that lost to the largest number $\{3, 1, 11, 10\}$.

   Correctness: *Note that the largest value will be correctly picked as it is impossible for the largest value to lose at any step. Now Lets argue why the second largest value has to be among the set of numbers that lost to largest number. Assume the second largest number is not in that list. This means that this number had to be beaten by another number except the largest number which is a contradiction as it is the second largest number.*

   Time analysis: *In order to find the largest number it takes* $\frac{n}{2} + \frac{n}{4} + \cdots = n - 1$ *comparisons. Note that in this approach the largest number is compared with* $\log n$ *numbers but one was already counted in the leaf level. So finding the second largest number takes* $\log n - 1$ *comparisons. So in total it takes* $n + \log n - 2$ *comparisons.*

2. (20 points) Let Random$(1, k)$ be a procedure that draws an integer uniformly at random from $[1, k]$ and returns it. We assume that a call of Random takes $O(1)$ worst-case time. The following recursive algorithm Random-Sample generates a random subset of $[1, n]$ with $m \le n$ distinct elements. Prove that Random-Sample returns a subset of $[1, n]$ of size m drawn uniformly at random.

```
RANDOM-SAMPLE($m, n$)
  if $m = 0$ then
    return $\emptyset$
  else
    $S \leftarrow$ RANDOM-SAMPLE($m-1, n-1$)

    $i \leftarrow$ RANDOM($1, n$)
    if $i \in S$ then
      return $S = S \cup \{n\}$
    else
      return $S = S \cup \{i\}$
    end if
    return $S$
  end if
```

Solution:

Proof

We prove that `RANDOM-SAMPLE(m,n)` *returns a subset of* $[1, n]$ *of size* m *drawn uniformly at random by proving that for each possible subset* $\{a_1, a_2, \ldots, a_m\}$*, the probability of returning* $\{a_1, a_2, \ldots, a_m\}$ *is given by:*

$$\Pr(\texttt{RANDOM-SAMPLE(m,n)} = \{a_1, a_2, \ldots, a_m\}) = \frac{1}{C_n^m} = \frac{m! \cdot (n-m)!}{n!}.$$

We will prove this by induction.

Base Case

When $m = 1$*,* `RANDOM-SAMPLE(1,n)` *calls* `RANDOM(1,n)` *to draw an integer uniformly from* $[1, n]$*. Then for any* $a \in [1, n]$*:*

$$\Pr(\texttt{RANDOM-SAMPLE(1,n)} = \{a\}) = \frac{1}{n} = \frac{(n-1)!}{n!}.$$

Inductive Step

When $1 < m \leq n$*,* `RANDOM-SAMPLE(m,n)` *first calls* `RANDOM-SAMPLE(m-1,n-1)` *to returns a subset of* $[1, n-1]$ *of size* $m-1$*. Suppose it is* $\{a_1, \ldots, a_{m-1}\}$*. For any such subset* $\{a_1, \ldots, a_{m-1}\}$*:*

$$\Pr(\texttt{RANDOM-SAMPLE(m-1,n-1)} = \{a_1, \ldots, a_{m-1}\}) = \frac{(m-1)! \cdot (n-m)!}{(n-1)!}.$$

Then `RANDOM-SAMPLE(m,n)` *calls* `RANDOM(1,n)` *to draw an integer uniformly from* $[1, n]$*. There are two cases:*

- *If* `RANDOM(1,n)` $\in \{a_1, \ldots, a_{m-1}\}$*, then the algorithm appends* n *to the returning subset.*
- *If* `RANDOM(1,n)` $\notin \{a_1, \ldots, a_{m-1}\}$*, then the algorithm appends* i *to the returning subset.*

Thus, for any possible subset $\{a_1, \ldots, a_m\}$ *of* $[1, n]$*:*

- *If* $n \in \{a_1, \ldots, a_m\}$*, without loss of generality, suppose* $a_m = n$*, then:*

$$\Pr(\texttt{RANDOM-SAMPLE(m,n)} = \{a_1, \ldots, a_m\}) =$$

$$\Pr(\texttt{RANDOM-SAMPLE(m-1,n-1)} = \{a_1, \ldots, a_{m-1}\}) \cdot \Pr(\texttt{RANDOM(1,n)} \in \{a_1, \ldots, a_{m-1}\}) =$$

$$\Pr(\texttt{RANDOM-SAMPLE(m-1,n-1)} = \{a_1, \ldots, a_{m-1}\}) \cdot \Pr(\texttt{RANDOM(1,n)} = n)$$

$$= \frac{(m-1)! \cdot (n-m)!}{(n-1)!} \cdot \left(\frac{m-1}{n} + \frac{1}{n}\right) = \frac{m! \cdot (n-m)!}{n!}.$$

- *If* $n \notin \{a_1, \ldots, a_m\}$*:*

$$\Pr(\texttt{RANDOM-SAMPLE(m,n)} = \{a_1, \ldots, a_m\}) =$$

$$m \cdot \Pr(\texttt{RANDOM-SAMPLE(m-1,n-1)} = \{a_1, \ldots, a_{j-1}, a_j, \ldots, a_m\}) \cdot \Pr(\texttt{RANDOM(1,n)} = a_j).$$

$$= m \cdot \frac{(m-1)! \cdot (n-m)!}{(n-1)!} \cdot \frac{1}{n} = \frac{m! \cdot (n-m)!}{n!}.$$

Thus, for each possible subset $\{a_1, a_2, \ldots, a_m\}$*, the probability of returning* $\{a_1, a_2, \ldots, a_m\}$ *is:*

$$\Pr(\texttt{RANDOM-SAMPLE(m,n)} = \{a_1, a_2, \ldots, a_m\}) = \frac{m! \cdot (n-m)!}{n!}.$$

3. (20 points) You are given a set of rectangles $S = \{R_1, R_2, \ldots, R_n\}$ such that all have their bottom sides on the x-axis. This means that each rectangle R_i is specified by a triple (l_i, r_i, h_i), where l_i and r_i are the x coordinates of its left and right sides, and h_i is the height. For simplicity, you may assume that all l_i, r_i, h_i for $i = 1, \ldots, n$ are distinct.

   You will design an algorithm to compute the union $C = R_1 \cup R_2 \cup \cdots \cup R_n$. Clearly, the bottom side of C is a straight line segment from the leftmost lower-left corner u of $R_1, \ldots, R_n$ to the rightmost lower-right corner v of $R_1, \ldots, R_n$. So you only need to find its upper boundary γ. Your algorithm should output γ as a list of "key points" sorted by their x-coordinate in the form $((x_1, y_1), (x_2, y_2), \ldots)$. Each key point is the left endpoint of some horizontal segment in γ except the last point in the list, which always has a y-coordinate 0 and is used to mark the termination.

   The following figure shows an example, where the input is

$$((2, 9, 10), (3, 7, 15), (5, 12, 12), (15, 20, 10), (19, 24, 8)),$$

and the output should be

$$((2,10),(3,15),(7,12),(12,0),(15,10),(20,8),(24,0)).$$

Design an algorithm that constructs γ in $O(n \log n)$ time and analyze its running time. Your algorithm MUST use the following divide-and-conquer strategy:

(i) Recursively solve the subproblems for $R_1, \dots, R_{n/2}$ and $R_{n/2+1}, \dots, R_n$, respectively. Note that there is no particular ordering among $R_1, R_2, \dots, R_n$ in the input.

(ii) Let γ_1 and γ_2 be the outputs of the two recursive calls in (i).

(iii) Combine γ_1 and γ_2 to produce γ.

<u>Solution:</u> *This is known as the "skyline" problem.*

We use a divide-and-conquer strategy as given by the hint. The base case has only one building (l, r, h). The skyline in this case is $((l, h), (r, 0))$.

For $n \geq 2$, we split our buildings into two subsets of $n/2$ buildings each. The first $n/2$ buildings and the second $n/2$ buildings are the two sub-problems that needs to be solved. Assume γ_1 and γ_2 are the skylines to the two sub-problems. We need to explain a method to combine γ_1 and γ_2 together.

This is how we approach:

Assume γ_1 looks like this:

$$((a_1, h_1), (a_2, h_2), \cdots, (a_k, h_k))$$

And assume γ_2 looks like this:

$$((a'_1, h'_1), (a'_2, h'_2), \cdots, (a'_{k'}, h'_{k'}))$$

note that k, k' are bounded by n (Think why?). Also we know that in γ_1 and γ_2, the values a_i and a'_i are sorted in an increasing order. Now we perform an operation like the merge procedure, while keeping track of current height of each sub-skyline. We use two pointers, p_1 and p_2, to iterate over γ_1 and γ_2, respectively. Meanwhile, we maintain the current heights of the two skylines as h_1 and h_2, which are initialized to 0. Suppose p_1 has a smaller x coordinate than p_2. First update h_1 to p_1's height. Then output (p_1's x-coordinate, $\max(h_1, h_2)$) to the merged skyline. Then we advance p_1. The case when p_2 has a smaller x coordinate is symmetric. Note that the merged skyline may contain some duplicated key points, i.e., consecutive key points with the same height. We can make another pass to remove them (or avoid adding them during the merging). This process takes $O(k + k') = O(n)$.

Time Complexity: *Note that at the beginning we did one sorting which takes $O(n \log n)$. Then we have $T(n) = 2T(n/2) + O(n)$. So the total complexity would be $O(n \log n)$.*

4. (20 points) We explore a different analysis of the application of randomized quicksort to an array of size n.

   (a) (2 points) For $i \in [1, n]$, let X_i be the indicator random variable for the event that the ith smallest number in the array is chosen as the pivot. That is, $X_i = 1$ if this event happens, and $X_i = 0$ otherwise. Derive $\mathrm{E}[X_i]$.

   (b) (2 points) Let $T(n)$ be a random variable that denotes the running time of randomized quicksort on an array of size n. Prove that

$$\mathrm{E}\big[T(n)\big] = \mathrm{E}\left[\sum_{i=1}^{n} X_i \cdot (T(i-1) + T(n-i) + \Theta(n))\right].$$

   (c) (2 points) Prove that $E\big[T(n)\big] = \frac{2}{n} \cdot \sum_{i=2}^{n-1} \mathrm{E}\big[T(i)\big] + \Theta(n)$.

   (d) (7 points) Prove that $\sum_{k=2}^{n-1} k \log k \le \frac{1}{2} n^2 \log n - \frac{1}{8} n^2$. (Hint: Consider $k = 2, 3, \ldots, (n/2) - 1$ and $k = n/2, \ldots, n-1$ separately.)

   (e) (7 points) Use (d) to show that the recurrence in (c) yields $\mathrm{E}\big[T(n)\big] = \Theta(n \log n)$. (Hint: Use substitution to show that $\mathrm{E}\big[T(n)\big] \le cn \log n$ for some positive constant c when n is sufficiently large.)

Solution:

(a)

$$E[X_i] = \Pr(X_i = 1) = \frac{1}{n}.$$

(b) If the i-th smallest number in the array is chosen as the pivot, i.e., $X_i = 1$, then the algorithm would partition the array into two sub-arrays: one with $i-1$ numbers and another with $n-i$ numbers. The partition costs $\Theta(n)$, and the algorithm continues to do quicksort for the two sub-arrays. Therefore,

$$E[T(n)] = E\left[\sum_{i=1}^{n} X_i \cdot (T(i-1) + T(n-i) + \Theta(n))\right].$$

(c)

$$\begin{aligned}
E[T(n)] &= E\left[\sum_{i=1}^{n} X_i \cdot (T(i-1) + T(n-i) + \Theta(n))\right] \\
&= \sum_{i=1}^{n} \Pr(X_i = 1) \cdot E[T(i-1) + T(n-i) + \Theta(n)]. \\
&= \frac{1}{n} \cdot \left(\sum_{i=1}^{n} E[T(i-1)] + \sum_{i=1}^{n} E[T(n-i)]\right) + \Theta(n). \\
&= \frac{2}{n} \cdot \sum_{i=2}^{n-1} E[T(i)] + \Theta(n).
\end{aligned}$$

(d) *Assume n is even:*

$$\sum_{k=2}^{n-1} k\log k = \sum_{k=2}^{n/2-1} k\log k + \sum_{k=n/2}^{n-1} k\log k \le \sum_{k=2}^{n/2-1} k\log\frac{n}{2} + \sum_{k=n/2}^{n-1} k\log n.$$

$$= \frac{(\frac{n}{2}-2)(\frac{n}{2}+1)}{2}\log\frac{n}{2} + \frac{\frac{n}{2}(\frac{3n}{2}+1)}{2}\log n$$

$$= \frac{n^2}{8}\log\frac{n}{2} + \frac{3n^2}{8}\log n - \frac{5n}{4}\log\frac{n}{2} - \frac{n}{4}\log n$$

$$\le \frac{n^2}{8}(\log n - \log 2) + \frac{3n^2}{8}\log n = \frac{n^2}{2}\log n - \frac{n^2}{8}$$

Assume n is odd; it can be proved similarly.

(e) *We show $E[T(n)] \le cn\log n$ for some positive constant c by induction.*

First, when $n=2$, it is true. Suppose $E[T(k)] \le ck\log k$ for $k = 2,\dots,n-1$, then by (c):

$$E[T(n)] = \frac{2}{n}\cdot\sum_{i=2}^{n-1} E[T(i)] + \Theta(n) \le \frac{2}{n}\cdot\sum_{i=2}^{n-1} ck\log k + \Theta(n).$$

By (d), we further have:

$$E[T(n)] \le \frac{2c}{n}\cdot\sum_{i=2}^{n-1} k\log k + \Theta(n) \le \frac{2c}{n}\cdot\left(\frac{n^2}{2}\log n - \frac{n^2}{8}\right) + \Theta(n) = cn\log n - \frac{cn}{4} + \Theta(n).$$

When n is sufficiently large, $E[T(n)] \le cn\log n$.
Then, we show $E[T(n)] \ge mn\log n$ for a positive constant m.
When $n=2$, it is true. Suppose $E[T(k)] \ge mk\log k$ for $k = 2,\dots,n-1$,

$$\sum_{k=2}^{n-1} k\log k > \sum_{k=\lceil\frac{n}{2}\rceil}^{n-1} k\log k \ge \frac{\left(\lceil\frac{n}{2}\rceil + n - 1\right)\cdot\lceil\frac{n}{2}\rceil}{2}\log\left\lceil\frac{n}{2}\right\rceil \ge \left(\frac{3n^2}{8} - \frac{n}{4}\right)\log\frac{n}{2}.$$

By (c), we have

$$E[T(n)] = \frac{2}{n}\sum_{i=2}^{n-1} E[T(i)] + \Theta(n) \ge \frac{2}{n}\cdot\left(\frac{3n^2}{8} - \frac{n}{4}\right)\cdot\log\frac{n}{2} + \Theta(n).$$

When n is sufficiently large, $E[T(n)] \ge mn\log n$. Thus, $E[T(n)] = \Theta(n\log n)$.

5. (20 points) Let $A[1..n]$ be an array of n possibly non-distinct integers. The array A may not be sorted. The q-th **quantiles** of $A[1..n]$ are the k-th smallest elements of A for $k = \lfloor n/q \rfloor, \lfloor 2n/q \rfloor, \ldots, \lfloor (q-1)n/q \rfloor$. Note that the q-th quantiles consist of $q-1$ elements of A.

   For example, if $A = [5, 8, 16, 2, 7, 11, 0, 9, 3, 4, 6, 7, 3, 15, 5, 12, 4, 7]$, the 3rd quantiles of A are $\{4, 7\}$, because the 3rd quantiles consist of the 6-th and 12-th smallest elements of A, which are 4 and 7, respectively.

   Suppose you have a black box worst-case linear-time algorithm that can find the median of an array of integers. That is, this algorithm runs in $O(s)$ time on an array of size s. Describe an algorithm that determines the q-th quantiles of $A[1..n]$ in $O(n \log q)$ time. Argure that your algorithm is correct. Derive the running time of your algorithm.

   Solution: *First we can utilize the Finding median black box to start partitioning the array recursively. So we first find the median and split the array in two halves. Then we can continue partitioning the left half and the right half recursively. The recurrence relation here is* $T(n) = 2T(n/2) + O(n)$.

   Once the recursion reduces the array sizes sufficiently, each subarray can contain at most one quantile, as quantiles are spaced $\frac{n}{q}$ *elements apart. Since the array size is halved with each recursion level, it takes about* $\log(q)$ *levels for the subarrays to become small enough to isolate individual quantiles. And note that now each subarray contains at most 1 quantile. This process takes* $O(n \log q)$ *time as it has* $O(n)$ *cost in each level and we have* $\log q$ *levels.*

   Now Note that we have q subarrays of each length $\frac{n}{q}$. *We can use finding k-th order statistic algorithm to find the kth smallest element in an array of length n in* $O(n)$ *Note that this is possible because we know which element to search for. So this part can be done in* $O(\frac{n}{q})$ *for each sub-array ad we have* $O(q)$ *of them. This means this part takes* $O(q \cdot n/q) = O(n)$. *So the total Cost will be* $O(n \log q)$.-