Sample Solution for Problem Set 3

Data Structures and Algorithms, Fall 2019

November 10, 2020

Contents

1 Problem 1

(a)

Lemma: Nodes indexed $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$ are nodes with $h = 0$.
Proof: $(\lfloor n/2 \rfloor + 1)$'s left-child should be indexed $2(\lfloor n/2 \rfloor + 1) > n$.

Mathematical Induction:

- I.B: For $h = 0$, from lemma we know there are $n - \lfloor n/2 \rfloor = \lceil n/2 \rceil$ nodes.
- I.H: There are at most $\lceil n/2^{i+1} \rceil$ nodes of height i, $i \geq 0$.
- I.S: For $h = i + 1$, there are $\lceil \frac{\lceil n/2^{i+1} \rceil}{2} \rceil = \lceil n/2^{i+2} \rceil$ nodes.

Note: The definition of **height** of a node in a heap is the number of edges on the longest simple downward path from the node to a leaf.

(b)

HeapDel(A, i)

```
1: if A[i] ≤ A[heap_size] then
2:     A[i] ← A[heap_size]
3:     MaxHeapify(A, i)
4: else
5:     HeapIncreaseKey(A, i, A[heap_size])
6: heap_size ← heap_size − 1
```

Note: If node heap_size is not a descendant of node i. A[heap_size] is probably greater than $A[i]$, in this case we need to adjust the heap upwards.

2 Problem 2

We consider the procedure of extracting all the elements from a max-heap, and prove the best running time is $\Omega(n \log n)$. Suppose the max-heap is H. For convenience we suppose all the number in H are $1, 2, ..., n$ and $n = 2^k - 1$ for some $k \geq 1$, i.e., it is a perfect tree.

Lemma 1. *Among all the number with height* 0 *in* H*, there are at least a half of the number is in* $L = \{1, 2, ..., \lceil \frac{n}{2} \rceil\}$.

Proof. Write $R = \{1, 2, ..., n\} \backslash L$. If a number with height 0 is in R, then its father is in R since the heap is a max-heap. Write the nodes at height 0 in R as R', since a number has at most 2 sons, then we have

$$|R'| + |R'|\frac{1}{2} + |R'|\frac{1}{4} + ... + 1 \leq |R|$$

For sufficiently large n, we have $|R'| \leq \frac{|R|}{2}$. Thus, the lemma is proved. □

Write L' as the nodes with height 0 in L. With the above lemma we know that L' is at least around $\frac{n}{4}$. Consider the first half popping operation, those nodes in L' will not be popping since they are the half smallest number. But they will be exchange to the top of the heap and go down during the procedure. Suppose after the first $\frac{n}{2}$ popping operation, and number i in L' is at position with distance $D(i)$ to the root. Obviously the complexity is at least

$$\sum_{i \in L'} D(i)$$

Since there are at most 2^i nodes with distance i to the root, and $|L'|$ is at least $\frac{n}{4}$, the amount is at least

$$\sum_{i \leq \log \frac{n}{8}} i \cdot 2^i = \Omega(n \log n)$$

Remark 1. *When* n *is not the power of* 2*, we can only consider the running time after the first layer is popping out (and the tree become a perfect tree again).* n *will be at least half of the origin* n*, but the constant is omit in* Ω*.*

Similarly, during the procedure we do not care much about constant, since they are all contained in Ω*.*

3 Problem 3

(a)

$$\begin{bmatrix} 2 & 3 & 4 & \infty \\ 5 & 8 & 9 & \infty \\ 12 & 14 & 16 & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix}$$

(b)

- ExtractMin()

  (1) Record $A[1,1]$ then replace it with ∞. Let $i = 1$ and $j = 1$.

  (2) If $i + 1 > n$ or $A[i, j + 1] < A[i + 1, j]$ goto (2.1), else goto (2.2).

  (2.1) Swap $A[i, j]$ with $A[i, j + 1]$, $i \leftarrow i + 1$. Goto (3).

  (2.2) Swap $A[i, j]$ with $A[i + 1, j]$, $j \leftarrow j + 1$. Goto (3).

  (3) If $i = n$ and $j = m$ then **return** $A[1, 1]_{old}$, else goto (2).

- Prove:

  – Loop invariant: Before step (2), $A[1 : i, 1 : j]$ is a magic matrix.

  – Initialization: There are only one element, $A[1, 1]$.

  – Maintain:

  ∗ If (2.1) run, j-th column is sorted;

  ∗ else (2.2) run, i-th row is sorted.

  – Termination: When $i = n$ and $j = m$, all rows and column are sorted. Therefore $A[1 : n, 1 : m]$ is a magic matrix.

(c)

- InsertMatrix(k)

  (1) $A[n, m] \leftarrow k$. Let $i = n$ and $j = m$.

  (2) If $i = 1$ or $A[i, j - 1] > A[i - 1, j]$ goto (2.1), else goto (2.2).

  (2.1) Swap $A[i, j]$ with $A[i, j - 1]$, $i \leftarrow i - 1$. Goto (3).

  (2.2) Swap $A[i, j]$ with $A[i - 1, j]$, $j \leftarrow j - 1$. Goto (3).

  (3) If $i = 1$ and $j = 1$ then **return**, else goto (2).

- Prove:

  – Loop invariant: Before step (2), $A[i : n, j : m]$ is a magic matrix.

  – Initialization: There are only one element, $A[n, m]$.

  – Maintain:

  ∗ If (2.1) run, j-th column is sorted;

  ∗ else (2.2) run, i-th row is sorted.

  – Termination: When $i = 1$ and $j = 1$, all rows and column are sorted. Therefore $A[1 : n, 1 : m]$ is a magic matrix.

(d)

- Run InsertMatrix n^2 times to build a magical matrix, then run ExtractMin n^2 times. The results are arranged in a new array, which is sorted.
- Prove: Each time we run ExtractMin, we got the minimum of magical matrix. The i-th value we got is the i-th smallest number
- Time Complexity: $\Theta(n^2) \times (O(n+n) + O(n+n)) = O(n^3)$.

(e)

- FindMatrix(k)
  (1) Let $i = 1$ and $j = m$.
  (2) If $A[i, j] = k$, **return** $true$.
  (3) If $i = n$ and $j = 1$, **return** $false$.
  (4) If $i = n$ or $A[i, j] < k$, $j \leftarrow j - 1$
  (5) If $j = m$ or $A[i, j] > k$, $i \leftarrow i + 1$.
  (6) Goto (2).
- Prove:
  - Loop invariant: After step (4) and (5), k cannot be in $A[1 : i, j : m]$.
  - Initialization: From step (2), $A[1, m] \neq k$.
  - Maintain:
    * If $A[i, j] < k$, $A[1, j-1] \leq A[i, j-1] \leq A[i, j] < k$, k cannot be in $A[1 : i, j - 1]$;
    * If $A[i, j] > k$, $A[i+1, m] \geq A[i+1, j] \leq A[i, j] > k$, k cannot be in $A[i + 1, j]$.
  - Termination: When $i = n$ and $j = 1$, k cannot be in $A[1 : n, 1 : m]$.
- i only goes up, j only goes down. That is $O(n + m)$.

4 Problem 4

(a) We'll use the substitution method to show that the best-case running time is $\Omega(n \lg n)$. Let $T(n)$ be the best-case time for the procedure QUICKSORT on an input of size n. We have

$$T(n) = \min_{1 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n).$$

Suppose that $T(n) \ge c(n \lg n + 2n)$ for some constant c and d is a constant. Substituting this guess into the recurrence gives

$$\begin{aligned}
T(n) &\ge \min_{1 \le q \le n-1} (cq \lg q + 2cq + c(n-q-1) \lg(n-q-1) + 2c(n-q-1)) + dn \\
&= (cn/2) \lg(n/2) + cn + c(n/2-1) \lg(n/2-1) + cn - 2c + dn \\
&\ge (cn/2) \lg n - cn/2 + c(n/2-1)(\lg n - 2) + 2cn - 2cdn \\
&= (cn/2) \lg n - cn/2 + (cn/2) \lg n - cn - \lg n + 2 + 2cn - 2cdn \\
&= cn \lg n + cn/2 - \lg n + 2 - 2cdn.
\end{aligned}$$

Taking a derivative with respect to q shows that the minimum is obtained when $q = n/2$. Taking c large enough to dominate the $-\lg(n) + 2 - 2cdn$ term makes this greater than $cn \lg n$, proving the bound.

(b) In the worst case, the number of calls to RANDOM is

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

As for the best case,

$$T(n) = 2T(n/2) + 1 = \Theta(n).$$

5 Problem 5

a. Since all elements are equal, RANDOMIZED-QUICKSORT always returns $q = r$. We have recurrence $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$.

b. **PARTITION'**(A, p, r)

```
1  x = A[p]
2  l = h = p
3  FOR j = p + 1 → r
4      IF A[j] < x
5          y = A[j]
6          A[j] = A[h + 1]
7          A[h + 1] = A[l]
8          A[l] = y
9          l = l + 1
10         h = h + 1
11     ELSE IF A[j] = x
12         exchangeA[h + 1] with A[j]
13         h = h + 1
14 RETURN (l, h)
```

Unstable. Since there is only one for-loop in this procedure, it's easy to prove that it takes $\Theta(r - p)$ time, and is in-place.

c. $\Theta(n)$

6 Problem 6

(a)

$p_i = \frac{(i-1)(n-i)}{\binom{n}{3}} = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$

(b)

$\frac{p_m}{p'_m} = \frac{6(m-1)(n-m)/n(n-1)(n-2)}{1/n} = \frac{6(m-1)(n-m)}{(n-1)(n-2)} \to \frac{3}{2}, \quad n \to \infty \ (m = \lfloor \frac{n+1}{2} \rfloor).$

(c)

$$\begin{aligned} P(\text{good pivot is NOT chosen}) &= 2\sum_{i=1}^{m} p_i \\ &= \frac{12}{n(n-1)(n-2)}\left(-\frac{m(m+1)(2m+1)}{6} + \frac{(n+1)m(m+1)}{2} - nm\right) \\ &= \frac{12m}{n(n-1)(n-2)}\left(-\frac{2m^2+3m+1}{6} + \frac{(3m+1)(m+1)}{2} - 3m\right) \\ &= \frac{2(7m^2-9m+3)}{3(3m-1)(3m-2)} \end{aligned}$$

where $m = \frac{n}{3}$. Let n tends to infinity, $Pr(\text{good pivot is NOT chosen}) \to \frac{14}{27}$, which is greater than unmodified quicksort with failure probability $\frac{2}{3}$.

(d)

No. Reasonable answers will be accepted [1].

7 Problem 7

(a)

The recursion function for $T(n)$ can be seen directly.

$$T(n) = 3T\left(\frac{2}{3}n\right) + \Theta(1)$$

Reserve $\alpha = \frac{1}{1-\log_3 2}$. Let's prove $T(n) = \Theta(n^\alpha)$. Without loss of generality, we will only prove the asymptotic upper bound via substitution method.

Suppose $T(n) \leq 3T\left(\frac{2}{3}n\right) + C$ for some constant C. We will now claim that $T(n) \leq C'n^\alpha - Cn$ holds for some sufficient large constant C'.

Base case is obvious, and note that

$$T(n) \leq 3T\left(\frac{2}{3}n\right) + C \leq 3C'\left(\frac{2}{3}\right)^\alpha n^\alpha - 2Cn + C \leq C'n^\alpha - Cn$$

which ends the proof.

[1] for example, running time of modified quicksort is more stable than original one.

(b)

We will prove the statement by induction on n, the size of array A.

Base case ($n = 2$) is obvious. Let's assume that statement is correct for all $n \leq k$, and our goal is to prove statement holds for $n = k + 1$. By induction hypothesis, array $A[0, \ldots, m-1]$ is sorted after execution of line 5, where $m = \lceil \frac{2}{3} n \rceil$, indicating that $A[n-m, n-1]$ contain i-th largest (if there exist elements with equal value, we will say the element with larger index is larger) element after executing line 5 for all $1 \leq i \leq n - m$ [2]. Hence, the i-th largest element will locate in $A[n-i]$ for $1 \leq i \leq n-m$ after executing line 6. The execution of line 7 simply sort the first m elements, which completes our proof.

(c)

No. Consider $A = [1, 4, 2, 3]$. It can be seen directly that modified algorithm fails to sort this given array.

(d)

Note that each swap will lower the number of inversions by exactly 1. Therefore, the number of swaps is equal to number of inversions, which is bounded by $\binom{n}{2}$.

[2] To be more specific, if i-th largest element is located in $A[0, n-m-1]$ after execution of line 5, then $i > m - (n-m) = 2m - n \geq n - m$ due to the fact $A[0, \ldots, m-1]$ is sorted

8 Problem 8

(a) n.

Proof. There are at most n loops so the worst case is at most n.
Consider the case $n, n-1, ..., 1$, it is exactly n. □

(b) Line 4 is executed in loop i is the same event as $A[i]$ is the smallest in $A[1, ..., i]$. The probability is $\frac{1}{i}$. For the n-th loop, it is $\frac{1}{n}$.

(c) Write I_i as the **Indicator random variable** that takes 1 if line 4 is executed in loop i, otherwise take 0. Now we have $E[I_i] = \Pr[I_i = 1] = \frac{1}{i}$, and the total time of execution is $\sum_{i\in[n]} I_i$, according to linear of expectation, $E[\sum_{i\in[n]} I_i] = \sum_{i\in[n]} E[I_i] = \sum_{i\in[n]} \frac{1}{i}$.

(a) n-1.

Proof. There are at most n loops and the first loop will not call line 7 since $A[1]$ can not be larger than infinity. So the worst cast is at most $n-1$.
Consider the case $n, n-1, ..., 1$, it is exactly $n-1$. □

(b) Line 7 is executed in loop i is the same event as $A[i]$ is the second smallest in $A[1, ..., i]$. The probability is $\frac{1}{i}$ for $i \geq 2$. For the n-th loop, it is $\frac{1}{n}$ if $n \geq 2$, it is 0 if $n = 1$.

(c) Write I_i as the **Indicator random variable** that takes 1 if line 7 is executed in loop i, otherwise take 0. Now we have $E[I_i] = \Pr[I_i = 1] = \frac{1}{i}$ for $i \geq 2$, and the total time of execution is $\sum_{i\in[n]} I_i$, according to linear of expectation, $E[\sum_{i\in[n]} I_i] = \sum_{i\in[n]} E[I_i] = \sum_{2\leq i\leq n} \frac{1}{i}$.

9 Problem 9

Algorithm: Suppose the input is an array P with n points. All array in this problem is indexed begin with 1. For a point $p \in P$, write $p.x, p.y$ as the coordinate of p. We first use $O(n \log n)$ to sort P by the value of x.

We describe a function $FindClosest(l, r, Q)$ where Q is a array of points, and is exactly the points in $P[l...r]$ sorted by y.

1 If $l == r$, return (nul, nul, ∞).

2 Let $m = \lfloor \frac{l+r}{2} \rfloor$. Create two array Q_L and Q_R. For $i = 1$ to $r - l$ do

- If $Q[i].x \leq P[m].x$, then add $Q[i]$ to Q_L. Otherwise add it to Q_R.

3 Let $(p_L, q_L, d_L) \leftarrow FindClosest(l, m, Q_L)$ and $(p_R, q_R, d_R) \leftarrow FindClosest(m + 1, r, Q_R)$. Compare d_L and d_R and take the tuple that contain the smaller one to be (p, q, d)

4 Create a new array Q', For $i = 1$ to $r - l$ do

- If $P[m].x - d \leq Q[i].x \leq P[m].x + d$, then add $Q[i]$ to Q'.

5 For $i = 1$ to $Q'.size$ do

- Let the smallest distance between $Q'[i]$ and $Q'[i + 1], ..., Q'[\min(Q'.size, i + 7)]$ be d', if $d' < d$ then update $(p, q, d) \leftarrow (Q'[i], q', d')$ where q' is the point that we picked with the smallest distance to $Q'[i]$ among $Q'[i + 1], ..., Q'[i + 7]$.

6 Return (p, q, d).

The algorithm simply sort P by y to create Q initially, and call $FindClosest(1, n, Q)$ to get the answer.

Correctness:

Lemma 2. *If P is sorted by x and Q is exactly the points in $P[l..r]$ sorted by y, then $FindClosest(l, r, Q)$ will return the closest tuple bewteen points in $P[l..r]$.*

Proof. We prove it by induction on $r - l + 1$. When $r == l$, the lemma is true since there are no points pair.

Now suppose $r > l$. According to step 2 and Q is sorted by y, we know that Q_L and Q_R are both sorted by y. Thus, the recursion in step 3 will return correct answers. i.e., after step 3, (p, q, d) is the closest tuple between nodes both in Q_L or both in Q_R.

The following claim will help us prove the correctness.

Claim 2.1. *If there are two node in Q_L and Q_R seperately that has distance smaller than d, the pair will be find in step 5*

Proof. Suppose the point pair is (p, q) where $p \in Q_L$ and $q \in Q_R$. Then we have $|p.x - q.x| < d$ and $|p.y - q.y| < d$. Thus, $p, q \in Q'$. We now prove that there are at most 6 points between p and q in Q' as Q' is sorted by y: considet the rectangle between $P[m].x - d$,$P[m].x + d$ and between $p.y, q.y$. We can split the rectangle into 8 rectangle with length of side at most $\frac{d}{2}$, each is contained in space $x \leq P[m].x$ or in space $x > P[m.x]$(since the rectangle has length $2n$ and width n). Each rectanle will contain at most 1 points, otherwise the two points will form a pair that create a distance smaller than d that voilate the recursion result. Thus, there are at most 6 points between p and q (there are at most 8 points in the rectangle). So the claim is proved. □

With this claim, $FindClosest(l, r, Q)$ will return the closest tuple between points in $P[l..r]$. □

Complexity: The complexty for n points has recursion

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

According to master method, $T(n) = O(n \log n)$.

Remark 2. *When $T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$, be careful that master method do not cover this case. You can use recursion tree method and substitution method to prove it is $O(n \log^2 n)$.*