# Solution for Problem Set 3

Mianzhi Pan, 181240045

October 5, 2020

## 1 Problem 1

**(a) Induction basis:** Notice nodes of height 0 are leaf nodes which have neither left child or right child, we can easily find their indexes are $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \cdots, n$, so the total number of them is $\lceil n/2 \rceil = \lceil n/2^{0+1} \rceil$.

**Induction step:** Suppose the property holds for height $h-1$, i.e. there are at most $\lceil n/2^h \rceil$ nodes of height $h-1$. If $\lceil n/2^h \rceil$ is even, the number of nodes of height $h$ is $\frac{\lceil n/2^h \rceil}{2}$. If it is odd, there are $\frac{\lceil n/2^h \rceil + 1}{2}$ nodes of height $h$. Consider both of the cases, the number of nodes of height $h$ is

$$\begin{aligned} &\lceil \frac{\lceil n/2^h \rceil}{2} \rceil \\ &= \lceil n/2^{h+1} \rceil \end{aligned}$$

**(b)** Swap $A[i]$ with the last element $x$ and then delete the last element of $A$. Comparing $x$ with its parent, if it is bigger than its parent, swap them and continue this operation until $x$ is no bigger than its parent. If $x$ is equal to its parent, we need to do nothing. If $x$ is smaller than its parent, we just need to do $MAX - HEAPIFY(A, i)$ to maintain the heap property.

## 2 Problem 3

**(a)**

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

**(b)** Since matrix $A$ is nonempty, $A[1][1]$ will not be $\infty$, and it must be the minimum element in the matrix. Compare its right element with its lower element and swap it with the smaller one, continue this process until both its right element and lower element are $\infty$(Notice the element will reach the boundary of $A$, we define the 'elements' outside $A$ are $\infty$). Then we set the minimum element to be $\infty$ and retrun its value stored before.

**Algorithm 1** EXTRACTMIN(A, i, j)

```
1: min = A[i][j]
2: if A[i][j+1] == ∞ and A[i+1][j] == ∞ then
3:     A[i][j] = ∞
4:     return min
5: end if
6: if A[i][j+1] < A[i+1][j] then
7:     swap(A[i][j+1], A[i][j])
8:     return EXTRACTMIN(A, i, j+1)
9: else
10:    swap(A[i+1][j], A[i][j])
11:    return EXTRACTMIN(A, i+1, j)
12: end if
```

Then we just need to call $EXTRACTMIN(A, 1, 1)$.

We say that when the minimum element reach $A[i][j]$, elements in each of its above rows $A[1][], \cdots, A[i-1][]$ and each of its left columns $A[][1], \cdots, A[][j-1]$ are all in sorted order.

Proof: **Induction basis:** Before we move the minimum element, it is at $A[1][1]$, the above rows and left columns are empty, hence they are sorted.

**Inductive step:** Suppose when the minimum element reaches $A[i-1][j-1]$, rows $A[1][], \cdots, A[i-2][]$ and columns $A[][1], \cdots, A[][j-2]$ are sorted.

If $A[i-1][j]$ and $A[i][j-1]$ are both $\infty$, we just set $A[i-1][j-1]$ to $\infty$, the property holds obviously.

If $A[i-1][j] < A[i][j-1]$, which means the right element is bigger than the lower one, we will swap $A[i-1][j-1]$ with $A[i-1][j]$ and the minimum element reaches $A[i-1][j]$. Among its above rows and left columns, only column $A[][j-1]$ is modified. Notice the new element $A[i-1][j-1]$ is the origin element $A[i-1][j]$, and it must be bigger than $A[i-2][j-1]$ and smaller than $A[i][j-1]$, the column $A[][j-1]$ is sorted.

If $A[i-1][j] \geq A[i][j-1]$, we will swap $A[i-1][j-1]$ with its lower element. Notice the new element $A[i-1][j-1]$ is the origin $A[i][j-1]$, it is smaller than $A[i-1][j]$ and greater than $A[i-1][j-2]$, the modified row $A[i-1][]$ is still sorted as well.

In conclusion, every time the minimum element reaches a new postion, its above rows and left columns are always sorted. And when the recurrence terminates, the rest elements are all $\infty$, hence the matrix is still magical.

**Time complexity:** Suppose no element in the origin matrix is $\infty$, then the minimum element $A[1][1]$ will reach $A[m][n]$ at last. Notice at each recurrence the algorithm just swap two elements and the minimum element moves right or down for one step. Therefore the total time is $c(m+n)$, the time complexity is $O(m+n)$.

**(c)** This algorithm is similar to that in problem (b). Notice the bottom right element must be $\infty$, we put the new element there, compare it with its upper element and its left element. If both of them are no bigger than it, the matrix

is already magical. Otherwise we swap the new inserted element with the larger one of its upper element and left one. Notice the new element may reach the boundary, we define the 'elements' outside the matrix is all $-\infty$. Continuse this operation until the new element need not to move.

We say that when the new inserted element reaches $A[i][j]$, the submatrix $A[i, \cdots, m][j, \cdots, n]$ is magical.

Proof: **Induction basis:** When the element is first inserted at bottom right, the submatrix $A[i, \cdots, m][j, \cdots, n]$ contains only one element $A[m][n]$, hence it is magical.

**Inductive step:** Suppose when the inserted element reaches $A[i-1][j-1]$, the submatrix $A[i-1, \cdots, m][j-1, \cdots, n]$ is magical. If neither of its upper element and left element is bigger than it, the recurrence terminates. Otherwise, if $A[i-1][j-2] > A[i-2][j-1]$, the algorithm will swap $A[i-1][j-1]$ with $A[i-1][j-2]$. Notice $A[i-1][j-2]$ must be greater than $A[i-1][j-1]$, we can easily find that the submatrix $A[i-1, \cdots, m][j-2, \cdots, n]$, whose up left element is the inserted one, is still magical.

If the upper element is bigger than the left one, i.e. $A[i-2][j-1] > A[i-1][j-2]$, it is similar to find the submatrix $A[i-2, \cdots, m][j-1, \cdots, n]$ is magical.

When the recurrence terminates, the matrix, which consists of the magical submatrix and the rest unmodified part, is magical.

**Time complexity:** Considering the worst case that the new inserted element is smaller than the origin $A[1][1]$, then it will reach $A[1][1]$ at last, totally $m+n$ steps and the time complexity is $O(m+n)$.

**(d)** Use the algorithm in (c) to insert the $n^2$ elements into an empty $n \times n$ matrix one by one. Then use the algorithm in (b) to extract the minimum element one by one and place them in the origin array from head to tail.

**Proof:** Since the origin matrix is empty, it will always be non-full before all elements are inserted, so we can use the algorithm in (c) to build a $n \times n$ magical matrix. After that, the matrix becomes full and it will always be nonempty before all elements are extracted. As a result, we can use algorithm in (b) to extract the minimum element from current matrix one by one and place them in the right place of the array. Since both of the algorithms are correct, this algorithm is correct as well.

**Time complexity:** Every time the insertion or extraction an element needs $O(n+n) = O(n)$ time, notice there are total $n^2$ elements, the time complexity is $2n^2 O(n) = O(n^3)$.

**(e)** We start from the bottom left element $A[1][n]$ and set it as *current*, if target value is smaller than it, we move *current* upwards, i.e. $current = A[1][n-1]$. If target is greater than it, we move *current* right. Otherwise, we find the target element. Continue this process until *current* can't move. If the algorithm doesn't find target until that process terminates, the target is not in the matrix.

**Time complexity:** Every time *current* may move upwards or right for one step and the algorithm just compare once during that process, the total steps is limited to $m+n$. So the time complexity is $O(m+n)$.

# 3 Problem 4

**(a)** Suppose $T(n) \geq cnlgn$ for some constant $c$, we have

$$\begin{aligned} T(n) &= min_{0 \leq q \leq n-1}(T(q) + T(n-q-1)) + \Theta(n) \\ &\geq min_{0 \leq q \leq n-1}(cqlgq + c(n-q-1)lg(n-q-1)) + \Theta(n) \end{aligned}$$

Let the first derivative of the expression with respect to $q$ be 0, then

$$\begin{aligned} &\geq c(2 \cdot \frac{n-1}{2} lg \frac{n-1}{2}) + \Theta(n) \\ &= c(n-1)lg(n-1) - c(n-1) + \Theta(n) \\ &\geq c(n-1)lg\frac{n}{2} - c(n-1) + \Theta(n) \quad (n \geq 2) \\ &= c(n-1)lgn - 2c(n-1) + \Theta(n) \\ &= cnlgn - (clgn + 2c(n-1)) + \Theta(n) \\ &\geq cnlgn \end{aligned}$$

for c small enough. So $T(n) = \Omega(n)$.

**(b)** worst case: $\Theta(n)$. best case: $\Theta(n)$.

# 4 Problem 5

**(a)** $\Theta(n^2)$, $\Theta(n^2)$.

**(b)**

**Algorithm 2** PARTITION'(A, p, r)

```
1: x = A[r], q = p − 1, t = p − 1
2: for j = p to r do
3:     if A[j] < x then
4:         y = A[j]
5:         A[j] = A[t + 1]
6:         A[t + 1] = A[q + 1]
7:         A[q + 1] = y
8:         q = q + 1
9:         t = t + 1
10:    end if
11:    if A[j] == x then
12:        exchange A[t + 1] with A[j]
13:        t = t + 1
14:    end if
15: end for
16: q = q + 1
17: return (q, t)
```

During the loop, $A[q+1, \cdots, t]$ are all equal to $A[r]$(Notice this is inconsistent with what the question says, we will do $q = q + 1$ after the loop). When it comes to a $A[j]$ smaller than $A[r]$, the algorithm let $A[t + 1] = A[q + 1]$ and $A[q + 1] = A[j]$, from which we can find that the algorithm is not stable.

**(c)** $\Theta(n)$, $\Theta(n)$.

# 5 Problem 6

**(a)**

$$p_i = \frac{C_{i-1}^1 \cdot C_{n-i}^1}{C_n^3} = \frac{(i-1)(n-i)}{C_n^3},\ i = 2, 3, \cdots, n-1.$$

**(b)** The likelihood of choosing $A'[\lfloor (n+1)/2 \rfloor]$ is

$$P_{median-of-3} = \frac{(\lfloor (n+1)/2 \rfloor - 1)(n - \lfloor (n+1)/2 \rfloor)}{C_n^3} \to \frac{3}{2n}\ (n \to \infty)$$

while the probability by the ordinary implement is $P_{ordinary} = \frac{1}{n}$, and their ratio is

$$lim_{n\to\infty} \frac{P_{median-of-3}}{P_{ordinary}} = \frac{3/2n}{1/n} = \frac{3}{2}.$$

**(c)**

$$\begin{aligned} P_{median-of-3} &= \int_{n/3}^{2n/3} \frac{(i-1)(n-i)}{C_n^3} di \\ &= \frac{1}{C_n^3} \int_{n/3}^{2n/3} (-i^2 + (n+1)i - n) di \\ &= \frac{1}{C_n^3} (-\frac{7n^3}{81} + \frac{n^2(n+1)}{6} - \frac{n^2}{3}) \\ &= \frac{n(13n-27)}{27(n-1)(n-2)} \end{aligned}$$

So $lim_{n\to\infty} P_{median-of-3} = \frac{13}{27}$, while $P_{ordinary} = \frac{1}{3}$.

**(d)** No. The best-case time complexity is still $\Theta(nlgn)$, this method leads us to the best case: as the recurrence goes on, the probability we choose the meidan element(the best case) becomes greater.

# 6 Problem 7

**(a)** $T(n) = 3T(2n/3)+\Theta(n)$, use the master theorem, we have $T(n) = \Theta(n^{lg3/lg1.5})$.

**(b)** We prove this by induction.

**Induction basis:** The algorithm is obviously correct when $n == 2$.

**Inductive step:** Suppose this algorithm works for any array with length less than $n$. Then after line 5, the array $A[0, \cdots, m-1]$ is sorted and therefore

all elements in $A[n-m, \cdots, m-1]$(the middle 1/3 array) are no smaller than those in the first 1/3 array $A[0, \cdots, n-m-1]$.

After line 6, we know $A[n-m, \cdots, n-1]$ is sorted, therefore
(1) $A[m, \cdots, n-1]$ is also sorted.
(2) All elements in $A[m, \cdots, n-1]$ is no smaller than those in $A[n-m, \cdots, m-1]$.

From (2) and what we say in the beginning, we can conclude that the biggest 1/3 elements of $A$ are all at the end of $A$**(Remember the length of** $A[m, \cdots, n-1]$ **must be no bigger than the length of** $A[n-m, \cdots, m-1]$**)**, i.e.
(3) All elements in $A[m, \cdots, n-1]$ is no smaller than those in $A[0, \cdots, m-1]$.

After line 7, we have
(4) $A[0, \cdots, m-1]$ is sorted.

In conclusion, from (1),(4),(3) it is clear that the input array is sorted after line 7.

**(c)** If we do this, the algorithm is not correct. And we can argue this in two aspects.

From the proof above, the algorithm is correct if and only if $A[m, \cdots, n-1]$ is no longer than $A[n-m, \cdots, m-1]$, we can deduce that $m \geq 2n/3$, i.e. $m = \lceil 2n/3 \rceil$.

In another aspect, there should be overlap between the beginning 1/3 part and the last 1/3 part, i.e. $n - m \leq m - 1$. If $m = \lceil 2n/3 \rceil$, $n$ should be no smaller than 3, which is consistent with the algorithm. However, if $m = \lfloor 2n/3 \rfloor$, $n$ should be no smaller than 7.

**(d)** The swap will be executed if and only if $n == 2\ and A[0] > A[1]$. Notice the larger elements will never be swapped to the front of the smaller ones, i.e. No pair of elements will be swapped twice. Hence the number of swaps is equal to the number of 'inversed pair', which is

$$\binom{n}{2}$$

in the worst case.

# 7 Problem 8

**(a)** $n$.

**(b)** Notice $min1$ is always the smallest element we have ever met. "line 4 is executed during the $n^{th}$ iteration" is equivalent to "find the smallest element of $A$ at the $n^{th}$ iteration." Suppose all elements are distinct, the probability is $\frac{(n-1)!}{n!} = \frac{1}{n}$.

**(c)** Set

$$X_i = I(A[i] < min1)$$

Then the number of executions of line 4 is

$$N = \sum_{i=1}^{n} X_i = \sum_{i=1}^{n} I(A[i] < min1)$$

The expected number is

$$E(N) = \sum_{i=1}^{n} Pr(A[i] < min1) = \sum_{i=1}^{n} \frac{1}{i}$$

**(d)** $n - 1$.

**(e)** Similar to problem (b), "line 7 is executed during the $n^{th}$ iteration" is equivalent to "find the second smallest element in the $n^{th}$ iteration", whose probability is $\frac{1}{n}$ $(n > 1)$.

**(f)** Set

$$Y_i = I(min1 \leq A[i] < min2)$$

The expected number is

$$\begin{aligned} E(N) &= E(\sum_{i=1}^{n} Y_i) \\ &= \sum_{i=1}^{n} E(Y_i) \\ &= \sum_{i=1}^{n} Pr(min1 \leq A[i] < min2) \\ &= \sum_{i=2}^{n} \frac{1}{i} \end{aligned}$$