# CSC263 Assignment 1

Zikun Chen, Yue Li, Yueen Ma

September 7, 2018

## Question 1

(a) (Written by Yue Li & Yueen Ma, Read by Zikun Chen)
$T(n)$ is $\mathbb{O}(n^2)$

Prove:

By assumption:

- line 1, 2, 5, 6 take constant time, denoted by $c_1$, $c_2$, $c_5$, $c_6$.
- for loop in line 3 and line 4 take constant time denoted by $c_3$ and $c_4$ .

For worst-case, we assume that the return statement in line 5 will never be reached, which means that A[i] is always equal to A[i-1].

Thus we get the worst-case time complexity T(n) as follows:

$$
\begin{aligned}
T(n) &= c_1 + c_2 + \sum_{i=2}^{n}(c_3 + \sum_{j=1}^{i-1} c_4 + c_5) + c_6 \\
&= \sum_{i=2}^{n}(c_3 + (i-1)c_4 + c_5) + b \\
&= \sum_{i=2}^{n}(c_3 + c_5) + \sum_{i=2}^{n} c_4(i-1) + b \\
&= (c_3 + c_5)(n-1) + c_4 \sum_{i=2}^{n}(i-1) + b \\
&= (c_3 + c_5)(n-1) + c_4(1 + 2 + ... + (n-1)) + b \\
&= (c_3 + c_5)(n-1) + c_4 \frac{n(n-1)}{2} + b \\
&\leq (c_3 + c_5)(\frac{n(n-1)}{2}) + c_4 \frac{n(n-1)}{2} + b \\
&= (c_3 + c_4 + c_5)\frac{n(n-1)}{2} + b \\
&= a\frac{n(n-1)}{2} + b \\
&= a(\frac{n^2}{2} - \frac{n}{2}) + b \\
&= \frac{a}{2}n^2 - \frac{a}{2}n + b \\
&\leq \frac{a}{2}n^2 + bn^2 \\
&= (\frac{a}{2} + b)n^2 \in O(n^2)
\end{aligned}
$$

Note that we combined constant part: Let $b = c_1 + c_2 + c_6$, $a = c_3 + c_4 + c_5$

Then there exists $c_0 = \frac{a}{2} + b \in \mathbb{R}$ and there is a constant $n_0 = 1 \in \mathbb{N}$, such that for all $n > n_0$, $T(n) \leq c_0 n^2$

Therefore $T(n) \in O(n^2)$

(b) (Written by Yue Li & Yueen Ma, Read by Zikun Chen)
$T(n)$ is $\Omega(n^2)$

Prove:
Assume the index of an array starts from 1.
Consider the input A of size n:
$\forall i \in \{1, ..., n\}$: A[i] = i, where 1≤i≤n, i∈ $\mathbb{N}^*$
ie: A = [1, 2, 3,...,n]

For worst-case, for each i from 2 to n, A[i] has to equal to A[i-1] in order to continuing inner for loop according to line 5, then

- When i = 2:

for $j = 1$ to $i - 1 = 1$: $A[j] = A[j] + 1$, then $A[1] = 1 + 1 = 2$
then $A[i] = A[2] = 2 = A[1] = A[i - 1]$
$A[i]$ equals $A[i - 1]$
The outer for loop continues to be executed

- when i = 3:

for $j = 1$ to $i - 1 = 2$: $A[j] = A[j] + 1$, then $A[1] = A[2] = 2 + 1 = 3$
then $A[i] = A[3] = 3 = A[2] = A[i - 1]$
$A[i]$ equals $A[i - 1]$
The outer for loop continues to be executed...

Following this pattern we know:
Since $A[i-1] = i-1$ and for each i, A[$i-1$] will end up adding 1 in inner for loop according to line 4, which means that $A[i - 1] = i - 1 + 1 = i$, therefore the condition $A[i - 1] = A[i]$ is always satisfied, therefore every outer and inner loop will be executed for each $i = 2$ to $i = n$, and then return statement in line 6 will be reached.
The outer loop takes n-2 times, for each i from 2 to n, the inner loop takes $i - 1$ times.

Thus we get T(n) as follows:

$$
\begin{aligned}
T(n) &= c_1 + c_2 + \sum_{i=2}^{n} (c_3 + \sum_{j=1}^{i-1} c_4 + c_5) + c_6 \\
&= \sum_{i=2}^{n} (c_3 + (i-1)c_4 + c_5) + b \\
&= \sum_{i=2}^{n} (c_3 + c_5) + \sum_{i=2}^{n} c_4(i-1) + b \\
&= (c_3 + c_5)(n-1) + c_4 \sum_{i=2}^{n} (i-1) + b \\
&= (c_3 + c_5)(n-1) + c_4(1 + 2 + ... + (n-1)) + b \\
&= (c_3 + c_5)(n-1) + c_4 \frac{n(n-1)}{2} + b \\
&\geq c_4 \frac{n(n-1)}{2} + b \\
&= a \frac{n(n-1)}{2} + b \\
&= a(\frac{n^2}{2} - \frac{n}{2}) + b \\
&= \frac{a}{2}n^2 - \frac{a}{2}n + b \\
&\geq \frac{a}{2}n^2 - \frac{a}{2}n
\end{aligned}
$$

Note that we combined constant part: Let $b = c_1 + c_2 + c_6$, $a = c_4$
Want to find $c_0 > 0$, $n_0 \in \mathbb{R}$ such that $\forall n \geq n_0, T(n) \geq \frac{a}{2}n^2 - \frac{a}{2}n \geq c_0 n^2$

let $\frac{a}{2}n^2 - \frac{a}{2}n \geq cn^2$
$\Rightarrow c < \frac{a}{2}$
Let $c_0 = \frac{a}{4}$, now find $n_0$:

$\frac{a}{2}n^2 - \frac{a}{2}n \geq \frac{a}{4}n^2$
$\Rightarrow n \geq 2$
Then there exists $c_0 = \frac{a}{4} \in \mathbb{R}^+$and there is a constant $n_0 = 2 \in \mathbb{N}$, such

that $\forall n \geq n_0, T(n) \geq c_0 n^2$
Therefore, $T(n)$ is $\Omega(n^2)$

## Question 2

(a) (Written by Yue Li, Read by Zikun Chen, Yueen Ma)
Assume the index of arrays starts from 1.
Use data structure minimum heap and its related operations.
We define a *Node* as a subclass of *list* with a new attribute *priority_index*
Below algorithm outputs the m smallest elements from A1, A2,..., Ak
in increasing sorted order.

```
def FindMSmallest(m, A1, A2, ..., Ak):
    result = [] of size m
    H = [] of size k

    for i = 1 to k:
    # We add a new attribute in each list indicating at which index,
    # the value represents the priority
    # i.e. priority of Bi = Bi.list[priority_index]
        Bi = Node(Ai); Bi.priority_index = 1
        H.append(Bi)

    # Build min heap based on the integer indicated by priority_index
    BuildMinHeap(H)

    for j = 1 to m:
        # Extract the root list whose element indicated by
        # priority_index is min_value in heap_array
        root = ExtractMin(H)
        int PI = root.priority_index
        result.append(root.list[PI])
        if PI < n / k:
            root.priority_index += 1
            H.Insert(root_list)
    return result
```

**Explanation**:

The ideas of above algorithm are as follows:

- First we build a min heap $H$. The content of each node in the min heap $H$ is a sorted list and the priority of each node is equal to the element indicated by the *priority_index* of each sorted list. Since given lists are sorted, initially, priorities are equal to the first elements of each list and the priority of parent nodes are less than or equal to those of its children due to heap ordering.

- Since $H$ is a min heap, the priority of the *root* node must be the min of the integers that are not yet in *result*. Then we use the heap extract operation $ExtractMin$ to extract the root node, and append its priority to the *result*. This ensures that the elements in *result* are in increasing order, because all the lists are in increasing order.

- After adding the element to *result*, if the *priority_index* of the list extracted from the root node is greater than the size of the list, which means that all elements of the list are now in *result*, we do not insert the node containing this list back in $H$. Otherwise, we need to the update *priority_index* to make sure it refers to the next integer in the list. We then insert that same list back to min heap $H$ with new priority. And the heap insertion operation is by definition based on the priority of each list in min heap.

- By inserting the *root* back in with a new priority, it ensures that after each $ExtractMin$ and $Insert$ procedure, the smallest integer among all the integers that are not in *result* is the the priority for the new root list. Therefore, repeating $ExtractMin$ and $Insert$ $m$ times would give us the $m$ smallest integers from all the given lists.
  Since in each cycle, the current smallest integer we got is added at the end of the result and each list is in increasing order, the $m$ integers in *result* are in increasing sorted order.

(b) (Written by Zikun Chen, Read by Yue Li, Yueen Ma)
On line 13, build heaps takes linear time in the worst case O(k).

The first for loop on line 5 executes k times.
In the second for loop on line 15, ExtractMin runs at most log(k) times in worst case. In addition, we assume in the worst case the if condition on line 21 is satisfied at every iteration of the loop and insert takes at most log(k) times. The rest of the command lines in this for loop takes constant time, say c.
Therefore for the second for loop executing m times, in worst case, we have at most m(2log(k) + c) executions.
The rest of the lines in the algorithm takes constant time, say a. Overall, when k is large,

$$\begin{aligned} T(n) &= 2k + log(k) + m(2log(k) + c) + a \\ &\leq 2k + log(k) + m(2log(k) + c) + ak \\ &= (2 + a)k + m(2log(k) + c) \\ &\leq (2 + a)k + m(3log(k)) \\ &\leq 2t(mlog(k) + k) && \text{where } t = max(2 + a, 3) \\ &\in O(mlog(k) + k) \end{aligned}$$

## Question 3

(a) (Written by Zikun Chen, Read by Yue Li, Yueen Ma)

Proof by induction:
Let $e_n$ = number of edges of a binomial heap with n elements
Let $b_n$ = binary representation of n

P(n): Prove that $e_n = n - \alpha(n)$ edges, where $\alpha(n)$ is the number of 1's in $b_n$.

**Base Case:**
Let n = 0
There is only 1 node $\Rightarrow e_n = 0$
$b_n = 1 \Rightarrow \alpha(n) = 1$
$e_n = n - \alpha(n)$ is true

**Inductive Step:**

Let $n \in \mathbb{N}, n > 1$, suppose $P(n)$ holds.

Want to show: $P(n+1)$

Let $T_k$ be one binomial tree with height k within the n-element binomial heap.
Claim: if $b_n$ ends with $s$ 1's then the $e_{n+1} - e_n = s$ $(\star)$
Since adding 1 to a binomial number $b_n$ is analogous to inserting a new node to a binomial heap with n elements.
If $b_n$ ends with $s$ 1's, then a binomial heap with n elements contains $T_0, ..., T_{s-1}$.
Consider n+1 (adding a new node to this binomial heap), for each $0 \leq i \leq s-1$, by creating a new edge, we combine $T_i$ with another $T_i$ which already existed in the binomial heap and create $T_{i+1}$.
Notice now $T_i$ no longer exists in the binomial heap.
(position $i$ of $b_n$ is now changed from 1 to 0, counting from right to left, start from position 0)
Therefore, by the end of the process, we would create in total s edges.
i.e. $e_{n+1} - e_n = s$

Then $\alpha(n+1) - \alpha(n) = 1 - s$
Since in $b_n$, in total $s$ 1's has been changed to 0 by the process above, and we changed 0 to 1 at position $s$.

Then $n + \alpha(n+1) - \alpha(n) = n + 1 - s$

$$\begin{aligned}
\Rightarrow n + 1 - \alpha(n+1) &= n + \alpha(n) - s \\
&= e_n - s \text{ by inductive hypothesis} \\
&= e_n + e_{n+1} - e_n \text{ by } (\star) \\
&= e_{n+1}
\end{aligned}$$

i.e. $P(n+1)$ is true
This finishes the inductive step.
Therefore, number of edges of a binomial heap with n elements is equal to $e_n = n - \alpha(n)$ edges, where $\alpha(n)$ is the number of 1's in the binary representation of $n$.

(b) (Written by Yueen Ma, Read by Yue Li, Zikun Chen)

The resulting binomial heap has $n+k$ nodes, and the original binomial heap has $n$ nodes.

From part (a), we know that the number of edges of heaps with size $n$ is $n-\alpha(n)$.

When inserting a node, a new edge will appear after each key comparison, because key comparisons only happen when two trees are about to be merged, and one new edge will form when two trees are merged. Then the number of key comparisons is equal to the number of extra edges.

Let the cost of inserting k nodes into binomial heap with size $n$ be $T(n,k)$.

$T(n,k) = (n+k-\alpha(n+k)) - (n-\alpha(n))$
$= n+k-n-\alpha(n+k)+\alpha(n)$
$= k-\alpha(n+k)+\alpha(n)$

Then the average time cost is:

$\frac{T(n,k)}{k}$
$= 1+\frac{\alpha(n)-\alpha(n+k)}{k}$
$\leq 1+\frac{(\lfloor log(n)\rfloor+1)-0}{k}$
$\leq 1+\frac{(log(n)+1)-0}{k}$
$\leq 1+\frac{k+1}{k}$
$\leq 1+\frac{k+k}{k}$
$= 1+2$
$= 3 \in O(1)$

Therefore, the worst-case average cost of an insertion, is bounded above by constant.