# CSC263 - Assignment 1

Cristyn Howard

January 18, 2018

explicit description of input where processing time proportional to input size

## Question 1:

- After the $k^{th}$ iteration of the for-loop in lines 3-5 (hence referred to as "loop 3-5"), *executed without returning on line 5*, we know that:
  - $A[1] = -2k$, because:
    - A[1] is initialized to 0 when line 1 executes
    - $A[1] = A[1] - 2$ occurs every time line 4 is executed, which happens once per loop 3-5 iteration, or $k$ times for $k$ iterations
  - for $j \in \mathbb{Z},\ \ 2 \leq j \leq k+1: \ \ A[j] = A[j-1] + 1$
    - on $k^{th}$ iteration loop 3-5 has executed with $i = \{2, 3, ..., k+1\}$
    - each iteration, line 5 executing without return shows that $A[i] = A[i-1] + 1$
    - line 4 increasing $A[1]$ to $A[i-1]$ on future iterations does not alter this equality because $A[i] = A[i-1] + 1 \rightarrow A[i] + 2x = A[i-1] + 1 + 2x \ \ \forall x \in \mathbb{Z}$
  - (i) $\boxed{\therefore \ \ A[1:k+1] = [-2k, -2k+1, -2k+2, ..., -2k+(k-1), -2k+k]}$
    - Notation: A[a:b] denotes a segment of A containing A[a], A[b], and all elements between them.
- Loop 3-5 starts with $i = 2$ and goes to $i = n$, $\ \therefore$ there are at most $n-1$ iterations of loop 3-5.
- From (i), we know that the input array B that does not return for all $n-1$ iterations looks like:

  $B = B[1:n] = [-2(n-1), -2(n-1)+1, -2(n-1)+2, ..., -2(n-1)+((n-1)-1), -2(n-1)+(n-1)]$

  $$\text{(ii)} \ \boxed{\therefore B = [-2n+2, -2n+3, -2n+4, ..., -n, -n+1]}$$
- Given the original input $A = [a_1, a_2, a_3, ..., a_n]$, after the $k^{th}$ iteration of loop 3-5, we have:

  $$\text{(iii)} \ \boxed{A = [-2k, a_2 - 2(k-1), a_3 - 2(k-2), ..., a_k - 2, a_{k+1}, ..., a_n]}$$
  - $j^{th}$ iteration of loop 3-5 decreases elements in A[1:j] by 2 on line 4
  - by $k^{th}$ iteration we have had each $j \in \mathbb{Z}, \ \ 1 \leq j \leq k$ exactly once
- Then input A of size n that ran through all n-1 iterations of loop 3-5 without return, we know from (ii) and (iii) with $k = n-1$ that:

  $[-2n+2, -2n+3, -2n+4, ..., -n, -n+1] \equiv [-2(n-1), a_2 - 2((n-1)-1), a_3 - 2((n-1)-2), ..., a_{n-1} - 2, a_n]$

This gives us $a_2 = -1$, $a_3 = -2$, etc. Or more generally, $a_j = 1 - j$ $\forall$ valid indices j in A.

- So for input A of size n, the array $A = [x, -1, -2, -3, ..., -n+1]$ will run through loop 3-5 without triggering a return on line 5 for a total of n-1 iterations.
  - Note: x is an arbitrary integer. $a_1$ is set to 0 in line 1, so the original input value of $a_1$ is irrelevant.
- Therefore, $\exists$ an input array A of size n $\forall n \in \mathbb{N}$ such that loop 3-5 runs $n-1$ times.
  On the $k^{th}$ call of loop 3-5, the for-loop in line 4 executes exactly k times, with each execution occuring in constant time.
  So $\exists$ input of size n $\forall n \in \mathbb{N}$ where the number of constant calls is:

$$\sum_{i=1}^{n-1} i = (\sum_{i=1}^{n} i) - n = \frac{n^2+n}{2} - n = \frac{n^2+n-2n}{2} = \frac{n^2-n}{2} \in \Theta(n^2)$$

  So $\exists$ input X of size n $\forall n \in \mathbb{N}$ such that $t(X) \in \Theta(n^2)$. $\therefore$ $T(n) \in \Omega(n^2)$

- Loop 3-5 can run at most n-1 times on an input of size n.
  On $k^{th}$ iteration of loop 3-5, for-loop in line 4 executes exactly k times, thus loop 4 has at most n-1 iterations on a given call.
  So for arbitrary input X of size n $\forall n \in \mathbb{N}$, $t(X) \leq c \cdot (n-1)(n-1) = c(n^2 - 2n + 1) \in \Theta(n^2)$.
  $\therefore$ $T(n) \in O(n^2)$

## Question 2:

a) Nodes in a complete ternary tree are mapped one-to-one to the elements of an array in a top-to-bottom, left-to-right fashion.

| Parent to child navigation: | Child to parent navigation: |
|---|---|
| $leftChildIndex = 3 \cdot parentIndex - 1$<br>$middleChildIndex = 3 \cdot parentIndex$<br>$rightChildIndex = 3 \cdot parentIndex + 1$ | $parentIndex = \lfloor \frac{childIndex+1}{3} \rfloor$ |

b) (1) Note: Internal nodes refers to non-leaf nodes.
Let height of complete ternary tree (CTT) A, $h_A = \lfloor \log_3(Heapsize_A \cdot 2) \rfloor$ (proven in part 2).
Max number of leaf nodes in A$= 3^{(h_A)}$.
Max # of nodes in A $= \sum_{i=0}^{h_A} 3^i = \frac{3^{(h_A+1)}-1}{2}$.
Then the number of non-leaf nodes in A is $\frac{3^{(h_A+1)}-1}{2} - 3^{(h_A)}$.
So in array A storing CTT, internal nodes are $A[i]$ for $i = 1$ to $i = \frac{3^{(h_A+1)}-1}{2} - 3^{(h_A)}$.

b) (2) <u>Assertion</u>: Height $h$ of complete ternary tree (CTT) $= \lfloor \log_3(Heapsize \cdot 2) \rfloor$
<u>Proof</u>:

- maxNodes = Max # of nodes of CTT of height h $= \sum_{i=0}^{h} 3^i = \frac{3^{h+1}-1}{2}$
- minNodes = Min # nodes of CTT of height h $= [\sum_{i=0}^{h-1} 3^i] + 1 = \frac{3^h-1}{2} + 1 = \frac{3^h+1}{2}$
- minNodes $\leq$ Heapsize of array containing CTT of height h $\leq$ maxNodes
- Must show that $\forall h \in \mathbb{N}$, $h = \lfloor \log_3(minNodes \cdot 2) \rfloor = \lfloor \log_3(maxNodes \cdot 2) \rfloor$

– First we will show the equality holds with maxNodes:

$$h = \lfloor \log_3(maxNodes \cdot 2) \rfloor = \lfloor \log_3(\frac{3^{h+1}-1}{2} \cdot 2) \rfloor = \lfloor \log_3(3^{h+1}-1) \rfloor$$

$$h \leq log_3(3^{h+1}-1) < h+1$$
$$3^h \leq 3^{h+1}-1 < 3^{h+1} \quad \equiv \quad 3^h \leq 3 \cdot 3^h - 1 < 3 \cdot 3^h$$
$$0 \leq 2 \cdot 3^h - 1 < 2 \cdot 3^h$$

This equality holds $\forall h \in \mathbb{N}$, so we can conclude that $\forall h \in \mathbb{N}, \quad h = \lfloor \log_3(maxNodes \cdot 2) \rfloor$.

– Next, we will show the equality holds with minNodes:

$$h = \lfloor \log_3(minNodes \cdot 2) \rfloor = \lfloor \log_3(\frac{3^h+1}{2} \cdot 2) \rfloor = \lfloor \log_3(3^h+1) \rfloor$$

$$h \leq log_3(3^h+1) < h+1$$
$$3^h \leq 3^h+1 < 3^{h+1} \quad \equiv \quad 3^h \leq 3^h+1 < 3 \cdot 3^h$$
$$0 \leq 1 < 2 \cdot 3^h$$

This equality holds $\forall h \in \mathbb{N}$, so we can conclude that $\forall h \in \mathbb{N}, \quad h = \lfloor \log_3(minNodes \cdot 2) \rfloor$.

– Can conclude that given an array A storing a CTT, height of CTT $h = \lfloor \log_3(Heapsize_A \cdot 2) \rfloor$.

c) (i) INSERT(A, x):

– line 1 $A.Heapsize += 1;$ (increase heapsize to make space for new element on end of array)
line 2 $A[Heapsize] = x;$ (store x in new last element of A)
line 3 myIndex = Heapsize;
line 4 parentIndex = $\lfloor \frac{myIndex+1}{3} \rfloor$;
line 5 if $A[parentIndex] < A[myIndex]$:
line 6 swap elements at myIndex and parIndex;
line 7 myIndex = $\lfloor \frac{myIndex+1}{3} \rfloor$;
line 8 go to line 4;

– What is the worst-case running time of Insert?

  * lines 1-3 and lines 4-8 run in constant time
  * the block from 4-8 is called at most $(height - 1)$ times
  * $(height - 1) = \lfloor \log_3(Heapsize_A \cdot 2) \rfloor - 1 = \lfloor \log_3(Heapsize_A) - \log_3 2 \rfloor - 1 \in \Theta(\log_3(x))$
  * given input A of size $n$ $\forall n \in \mathbb{N}$, $t(A) \leq \lfloor \log_3(Heapsize_A) - \log_3 2 \rfloor - 1 \in \Theta(\log_3(x))$
  * $\therefore \; T(n) \in O(\log_3 n)$
  * any input $x \geq A[1]$ requires full $(height - 1)$ swaps
  * $\therefore \; \forall n \in \mathbb{N}, \exists$ input x that when inserted in array A of size n, requires $(height - 1)$ swaps
  * $\therefore \; T(n) \in \Omega(\log_3 n)$
  * $T(n) \in O(\log_3 n) \wedge T(n) \in \Omega(\log_3 n) \rightarrow T(n) \in \Theta(\log_3 n)$

c) (ii) ExtractMax(A, x):

- line 1 Swap first and last elements of array, return last as max.
  line 2 myIndex = 1;
  line 3 Get values stored in all three children elements of myIndex.
  line 4 if any child element value is larger than myIndex value:
  line 5     swap myIndex value with largest child value
  line 6     set myIndex to largest child's index
  line 7     go to line 3;

- What is the worst-case running time of ExtractMax?
  * lines 1-2 and lines 3-7 run in constant time
  * the block from 3-7 is called at most $(height - 1)$ times
  * $(height - 1) = \lfloor \log_3(Heapsize_A \cdot 2) \rfloor - 1 = \lfloor \log_3(Heapsize_A) - \log_3 2 \rfloor - 1 \in \Theta(\log_3(x))$
  * given input A of size $n$ $\forall n \in \mathbb{N}$, $t(A) \leq \lfloor \log_3(Heapsize_A) - \log_3 2 \rfloor - 1 \in \Theta(\log_3(x))$
  * $\therefore\ T(n) \in O(\log_3 n)$
  * any input A where the last element of A has the smallest key requires full $(height - 1)$ swaps
  * $\therefore\ \forall n \in \mathbb{N}, \exists$ array A of size n for which ExtractMax requires $(height - 1)$ swaps
  * $\therefore\ T(n) \in \Omega(\log_3 n)$
  * $T(n) \in O(\log_3 n) \wedge T(n) \in \Omega(\log_3 n) \rightarrow T(n) \in \Theta(\log_3 n)$

c) (iii) Update(A, i, key):

- line 1 $A[i] = key$;
  line 2 myIndex = i;
  line 3 get parentIndex of myIndex, if parent key larger:
  line 4     swap myIndex with parent, set myIndex=parentIndex, go to line 3;
  line 5 else get existing children, if any child element value is larger than myIndex value:
  line 6     swap myIndex value with largest child value;
  line 7     set myIndex to largest child's index;
  line 8     go to line 5;

- Can swap at most height-1 times, $\therefore\ T(n) \in O(\log_3 n)$.
  Can update leaf node with value higher than root, will always require full height-1 swaps, $\therefore\ T(n) \in \Omega(\log_3 n)$.
  $T(n) \in O(\log_3 n) \wedge T(n) \in \Omega(\log_3 n) \rightarrow T(n) \in \Theta(\log_3 n)$

c) (iv) Remove(A, i):

- line 1 swap i and last element values, shrink array size by 1;<br>line 2 myIndex = i;<br>line 3 get parentIndex of myIndex, if parent key larger:<br>line 4     swap myIndex with parent, set myIndex=parentIndex, go to line 3;<br>line 5 else get children of myIndex, if any child element value is larger than myIndex value:<br>line 6     swap myIndex value with largest child value;<br>line 7     set myIndex to largest child's index;<br>line 8     go to line 5;

- Can swap at most height-1 times, $\therefore\ T(n) \in O(\log_3 n)$.<br>Removing root node from array where last element had the smallest key will always require full height-1 swaps (smallest key placed at root, pushed all the way back down to leaf node), $\therefore\ T(n) \in \Omega(\log_3 n)$.<br>$T(n) \in O(\log_3 n) \land T(n) \in \Omega(\log_3 n) \to T(n) \in \Theta(\log_3 n)$