CS2223: Algorithms
D-Term
Midterm I
50 Minutes

Solution

Question 1

Assume the following functions:

$N + 3\text{Log}N$, $\quad \text{LogLog}\ N$, $\quad N\ \text{Log}N$, $\quad \sqrt{N}$, $\quad 1/N^2$, $\quad N^2 + 100N$

$N^{0.5} + 10\ \text{Log}N$, $\quad 1/\text{Log}N$, $\quad 2^{\text{Log}_2 N}$

Hint: $a^{\text{Log}_b N} = N^{\text{Log}_b a}$

1.1) [5 Points] Sort the functions in an ascending order according to their growth

$1/N^2$, $\quad 1/\text{Log}N$, $\quad \text{LogLog}\ N$, $\quad \sqrt{N}$, $\quad N^{0.5} + 10\ \text{Log}N$,

$2^{\text{Log}_2 N}$, $\quad N + 3\text{Log}N$, $\quad N\ \text{Log}N$, $\quad N^2 + 100N$

1.2) [5 Points] If multiple functions have the same theta order, then list them in groups

We have $\theta(n)$ ← $\{2^{\text{Log}_2 N}$, $\quad N + 3\text{Log}N\}$

We have $\theta(n^{0.5})$ ← $\{\sqrt{N}$, $\quad N^{0.5} + 10\ \text{Log}N\}$

Question 2 **[10 Points]**
Assume the following two recurrences

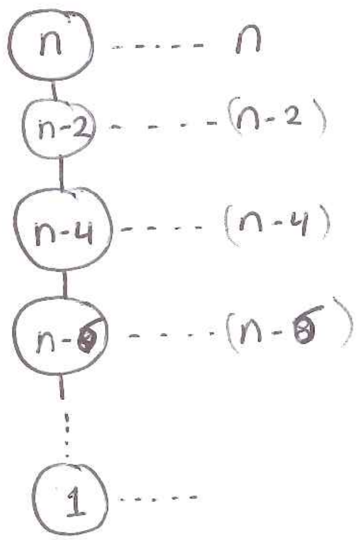
$T(n) = T(n-2) + n$ and $F(n) = 2\,F(n/4) + n$

2.1) **[5 Points] Use either Master Theorem or Tree-Based Method to derive the Big-O complexity for T(n) & F(n)**

F(n) ⇒ Use Master Theorem ⇒ $a=2,\ b=4,\ \alpha=1,\ \beta=\log_4 2 < 1$

$\Rightarrow \boxed{F(n) = O(n)}$

T(n) ⇒ Use Tree-Based



$n + (n-2) + (n-4) + (n-6) + \dots + 6 + 4 + 2$

=n

=n

=n

=n

n/2 Times

$\Downarrow$

$\boxed{T(n) = O(n^2)}$

2.2) **[5 Points] Mark each of the following statements with True or False**

(a) $T(n) = \theta(F(n))$	False
(b) $F(n) = O(T(n))$	True
(c) $T(n) = O(n\text{Log}n)$	False
(d) $F(n) + T(n) = \theta(F(n)^2)$	True
(e) $T(n) + F(n) = \Omega(n\text{Log}n)$	True

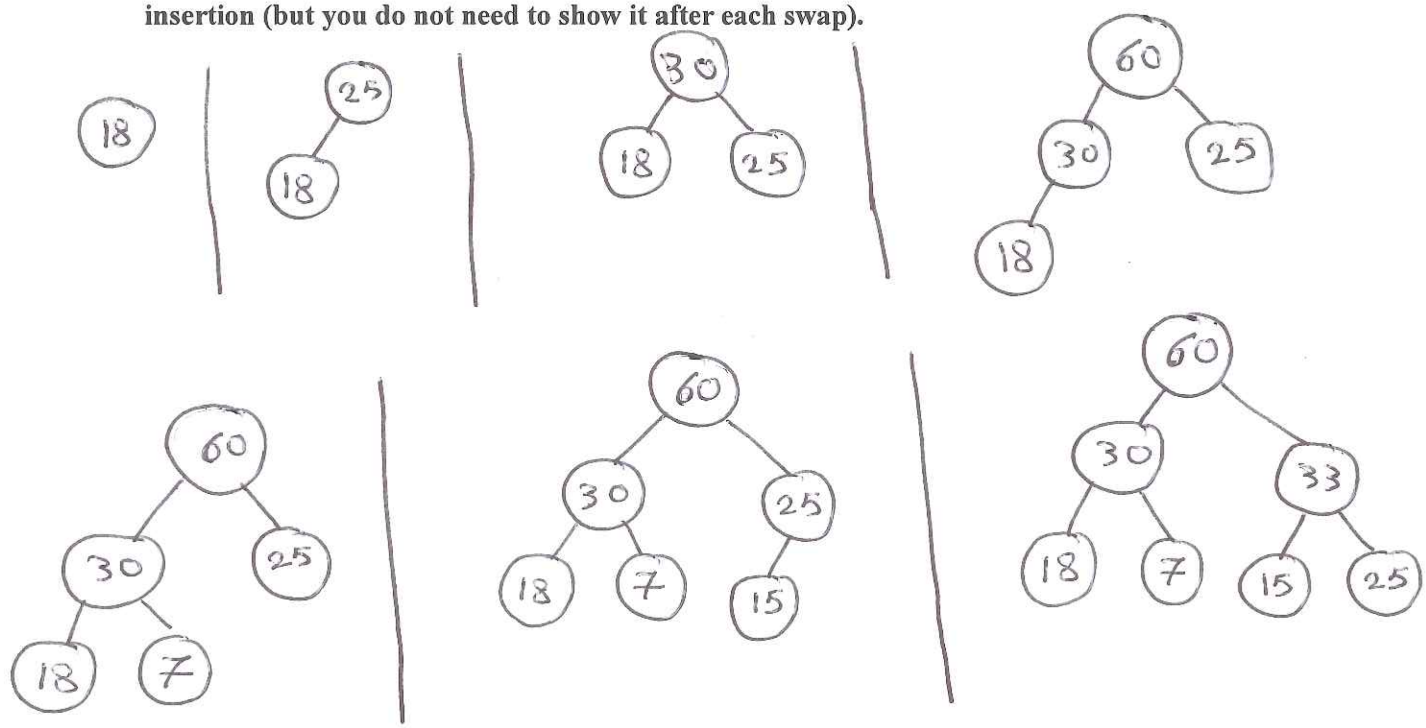
Question 3 [10 Points]

3.1) [3 Points]Complete the following sentence **// Assume root is at level 1:**

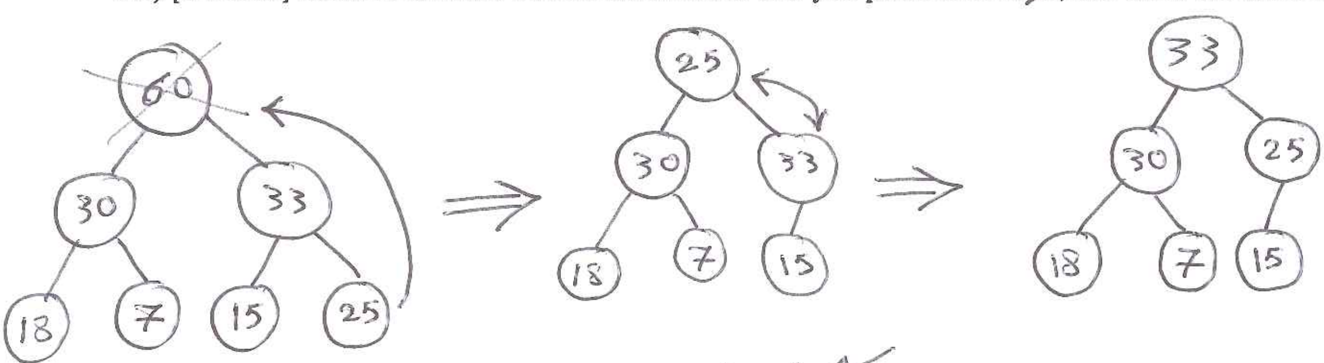
A MaxHeap tree containing 100 nodes will be of height ……7….., the number of nodes in the last level is ……….37..........., and the number of nodes in the 4th level is ....8...........

3.2) [5 Points] Assume the following sequence of values 18, 25, 30, 60, 7, 15, 33

Build the MaxHeap tree corresponding the above sequence of insertions. Show the tree after each insertion (but you do not need to show it after each swap).



3.3) [2 Points] Remove the root node from the final tree you produced in ~~4.1~~ 3.2, and show the fixed tree.



"optional To show"

Question 4 [10 Points]

Assume we have K sorted arrays of total size N (That is, each array is sorted and the size of all arrays combined is N).

4.1) [5 Points] A Naïve merging of these sorted arrays to get a final globally sorted array involves maintaining a pointer for the current minimum in each array and comparing them (Like in MergeSort). A tight complexity of this merging process will be:

(a) $\Theta(NK)$

(b) $\Theta(N^2K)$

(c) $\Omega(N^2K^2)$

(d) $O(N)$

The O(NK) is obtained By:
- -- Maintain a pointer at the beginning of each of the K lists
- -- Compare the positions from the K lists to find the smallest value. This value will be the next value in the globally sorted list. → **O(K)**
- -- Advance the pointer of the list you select from in Step 2 and repeat until all N values are consumed → Repeat O(K) work N times → **O(NK)**

4.2) [5 Points] There is a more efficient algorithm for merging the arrays in O(N Log K). Sketch a pseudocode for such algorithm.

--Instead of each time, we compare K values (one from each list) to find the smallest, we will use a MinHeap to organize the K values (the smallest from each list)

** The initial building will need O(K Log K)

--Then select the smallest from the MinHeap and add it to the globally sorted array

** This needs O(Log K)

-- Assume the selected value in Step 2 belongs to List J, then add the next value from List J to the MinHeap

** This needs O(Log K)

--Repeat Steps 2 and 3 N times

Total Complexity = O(N Log K)

Question 5 [5 Points]

Write the recurrent equation for the following function (no need to solve it)

```
Function F (Array A)              //initial size of A = n
  - Jump to the middle element to divide A into two halves
  - scan the 1st half to get the min value (Say x)
  - If (x is even)
       - Call F() on the 1st half
  - Else
       - Call F() on the 2nd half
  - End If
```

O(1)

n/2 -1

Only one of them is called ➔ T(n/2)

All of the following answers are equivalent and correct

$T(n) = T(n/2) + n/2 - 1$

$T(n) = T(n/2) + n/2$

$T(n) = T(n/2) + n/2 + O(1)$

$T(n) = T(n/2) + n/2 + c$