CPSC 335 Homework 1 (85 points)

Deadline: Sunday, March 5, 11:59 PM

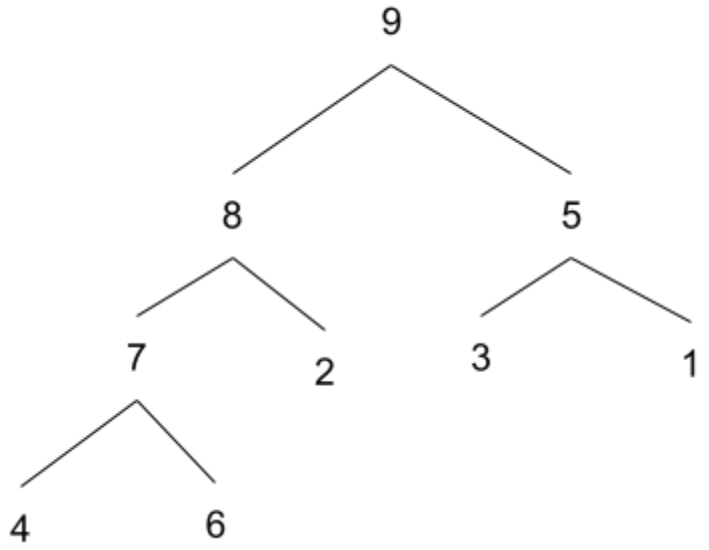
This homework can be done individually or in a group of maximum **3** people. For a group of 2 or 3 members, each of the group members will have to submit, even though the answers will be the same for all the members of the group. All members of the group will get the same grade. If one group member does not submit, that person gets 0. Indicate in an additional .txt file, the names and email ids of members in the group. If working individually, indicate in the .txt, your name and email id.

#1 [20 points]

In class we learnt how to apply the Heap sort algorithm to a set of numbers. Please refer to this algorithm ("Heap Sort") on Canvas. For a set of numbers, the steps in the first part involving Build Max heap are posted on Canvas: "Heap sort Part 1 Build Max heap".

Working with the same set of numbers, now that the heap is built, sketch the next steps of the algorithm, which will eventually sort the array. Draw the tree and corresponding array at each step, just like the steps in the partial example solution.

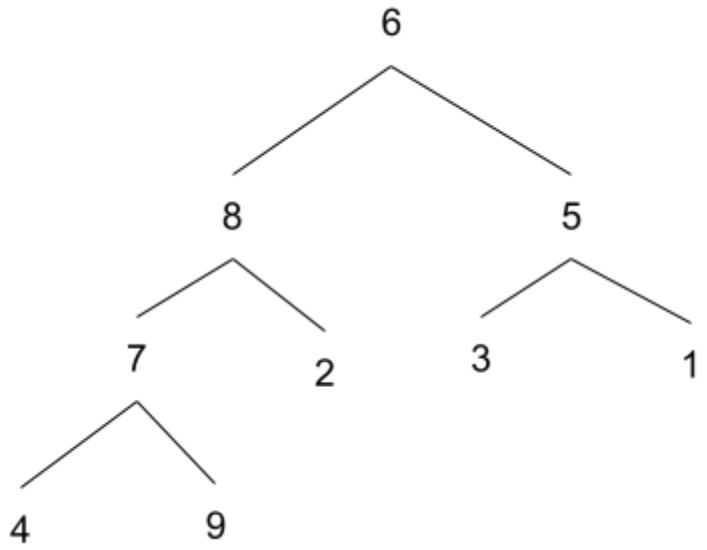
Given heap:



Given array: 9, 8, 5, 7, 2, 3, 1, 4, 6

Step 1: exchange A[1] and A[9]

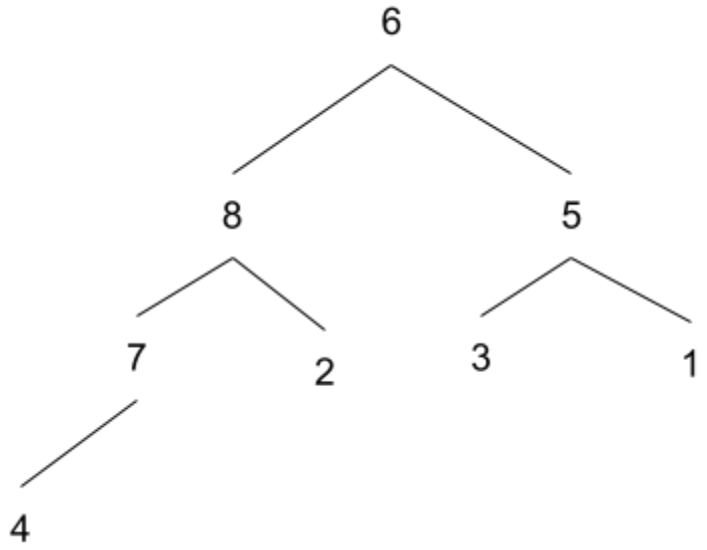
Heap:



Array: 6, 8, 5, 7, 2, 3, 1, 4, 9

Step 2: heap-size[A] = 8 so ignore 9

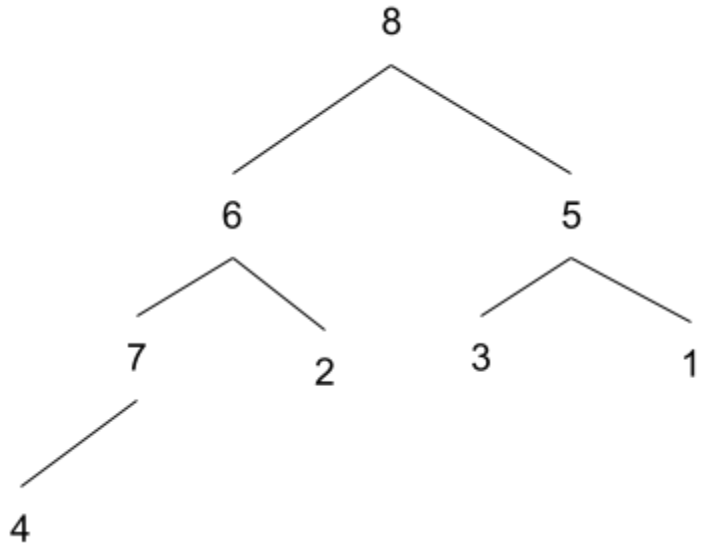
Heap:



Array: 6, 8, 5, 7, 2, 3, 1, 4, 9

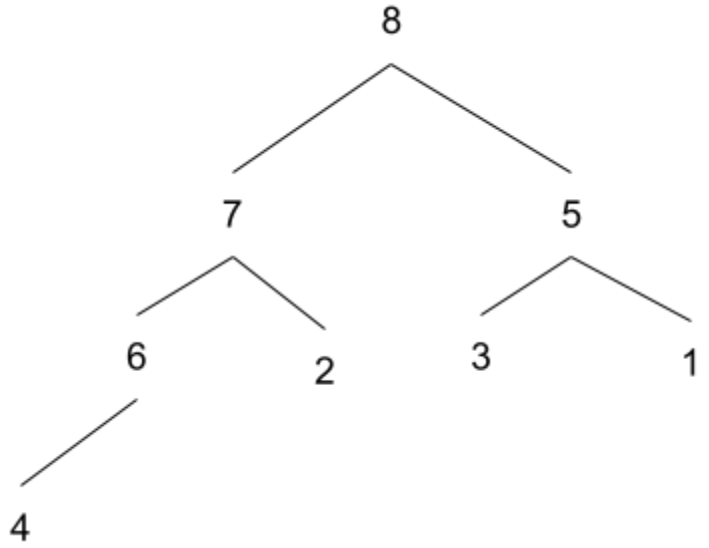
Step 3: max-heapify, exchange 6 and 8

Heap:



Array: 8, 6, 5, 7, 2, 3, 1, 4, 9

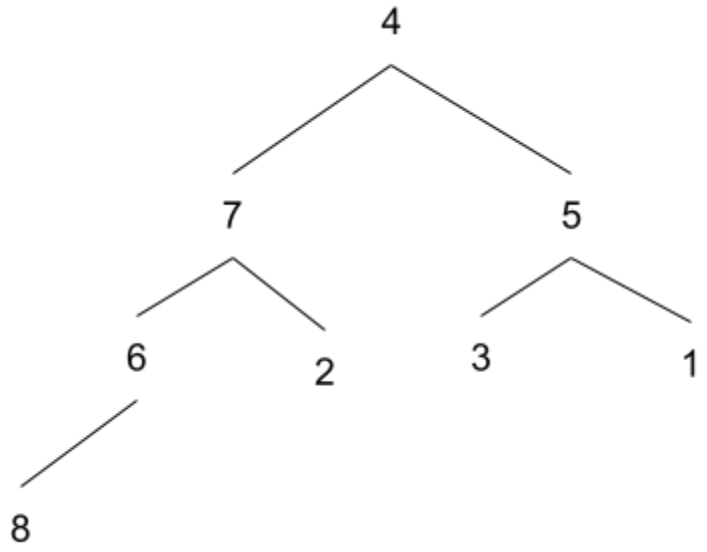
Step 4: max-heapify, exchange 6 and 7
Heap:



Array: 8, 7, 6, 5, 2, 3, 1, 4, 9

Step 5: exchange A[1] and A[8]

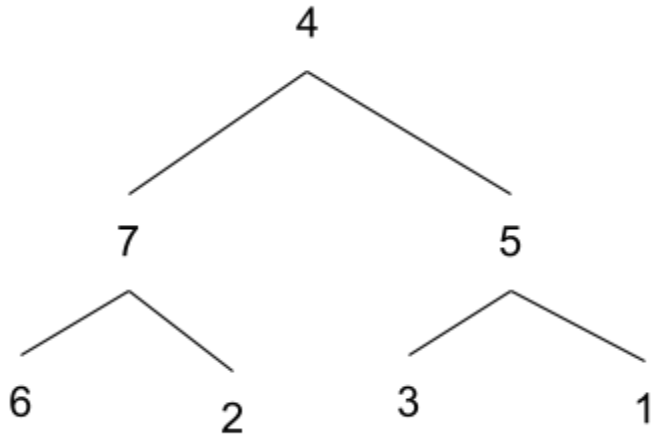
Heap:



Array: 4, 7, 5, 6, 2, 3, 1, 8, 9

Step 6:heap-size[A] = 7 so ignore 8

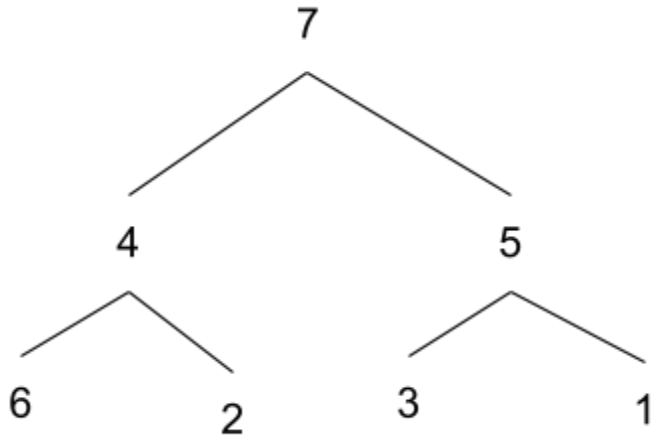
Heap:



Array: 4, 7, 5, 6, 2, 3, 1, 8, 9

Step 7: Max-Heapify , exchange 4 and 7

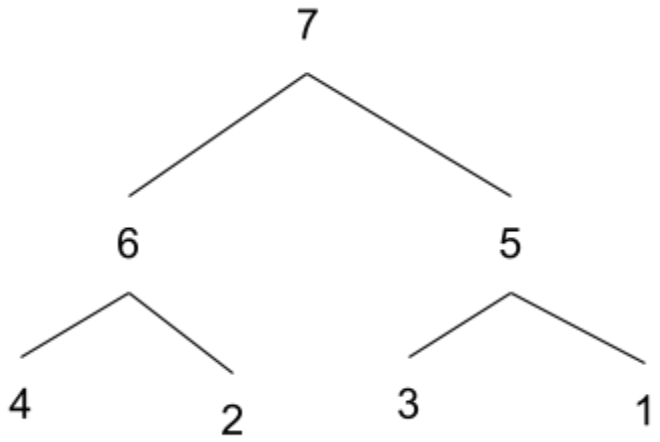
Heap:



Array: 7, 4, 5, 6, 2, 3, 1, 8, 9

Step 8: max-heapify, exchange 4 and 6

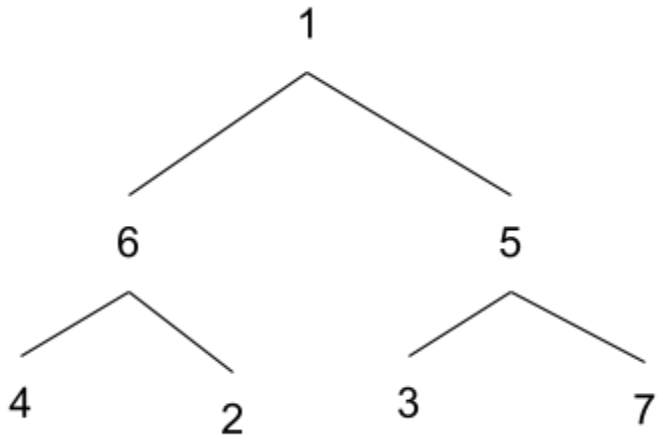
Heap:



Array: 7, 6, 5, 4, 2, 3, 1, 8, 9

Step 9: exchange A[1] and A[7]

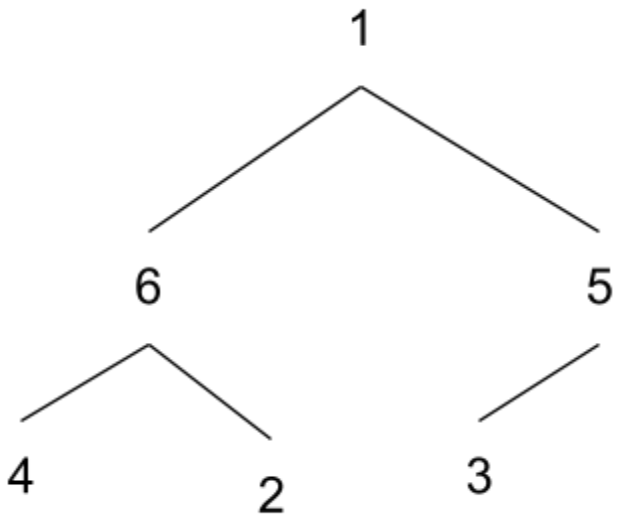
Heap:



Array: 1, 6, 5, 4, 2, 3, 7, 8, 9

Step 10: heap-size[A] = 6 so ignore 7

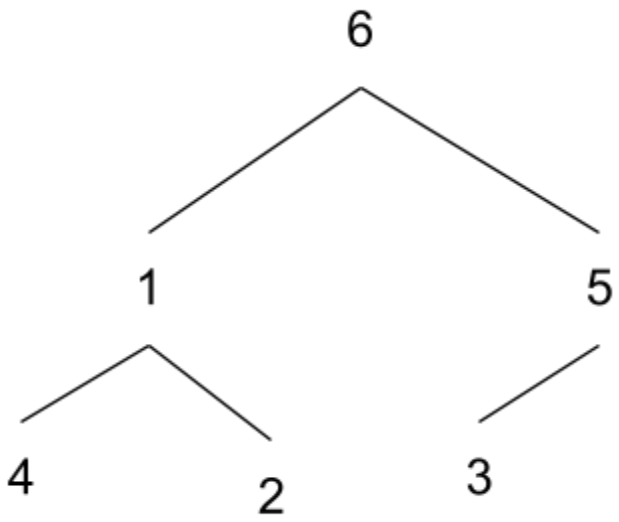
Heap:



Array: 1, 6, 5, 4, 2, 3, 7, 8, 9

Step 11: Max-Heapify, exchange 1 and 6

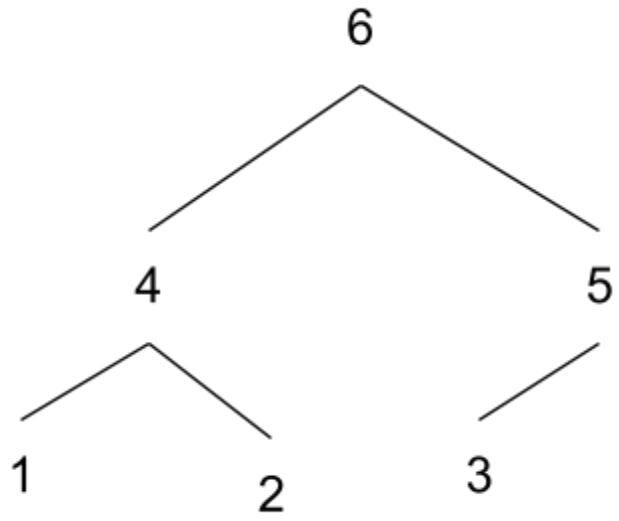
Heap:



Array: 6, 1, 5, 4, 2, 3, 7, 8, 9

Step 12: max-heapify, exchange 1 and 4

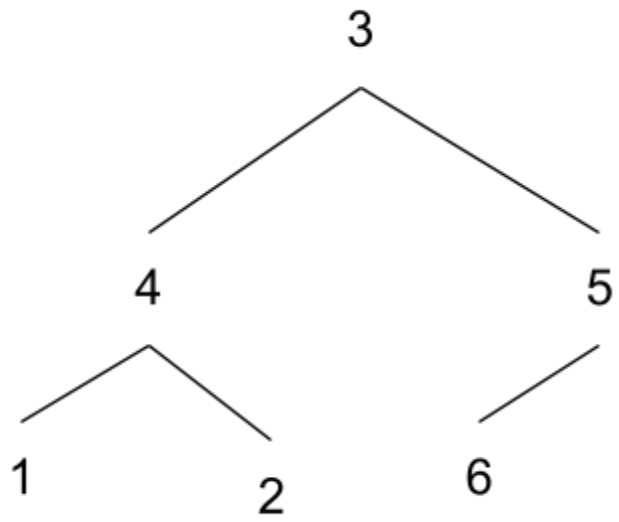
Heap:



Array: 6, 4, 5, 1, 2, 3, 7, 8, 9

Step 13: exchange A[1] and A[6]

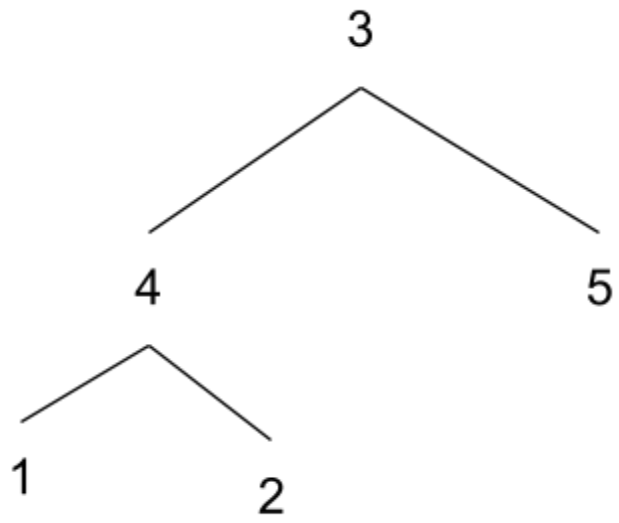
Heap:



Array: 3, 4, 5, 1, 2, 6, 7, 8, 9

Step 14: heap-size[A] = 5 so ignore 6

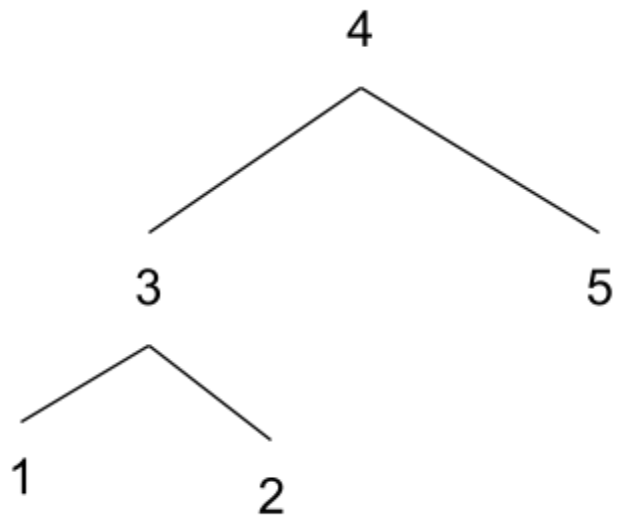
Heap:



Array: 3, 4, 5, 1, 2, 6, 7, 8, 9

Step 15: Max-Heapify, exchange 3 and 4

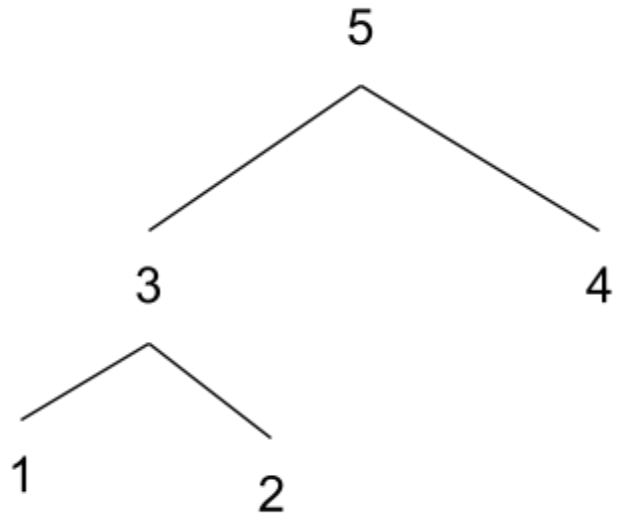
Heap:



Array: 4, 3, 5, 1, 2, 6, 7, 8, 9

Step 16: max-heapify exchange 4 and 5

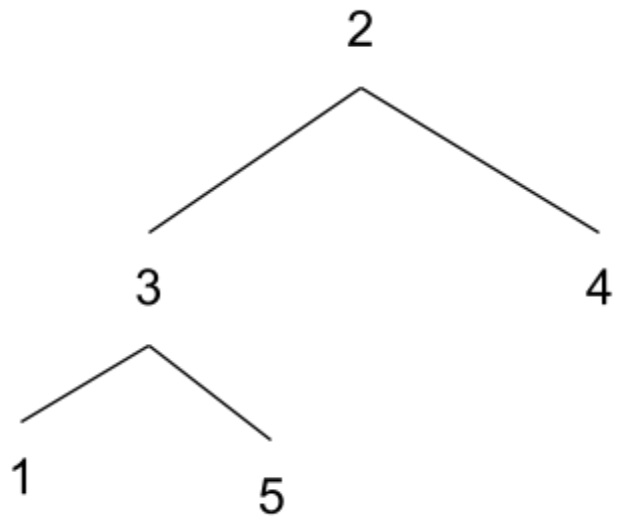
Heap:



Array: 5, 3, 4, 1, 2, 6, 7, 8, 9

Step 17: exchange A[1] and A[5]

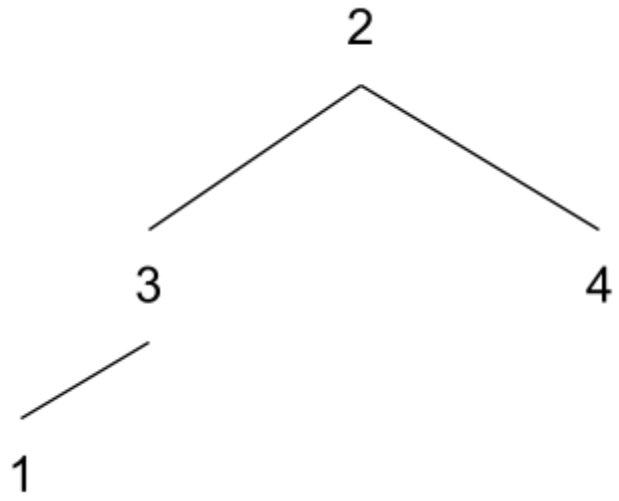
Heap:



Array: 2, 3, 4, 1, 5, 6, 7, 8, 9

Step 18: heap-size[A] = 4 so ignore 5

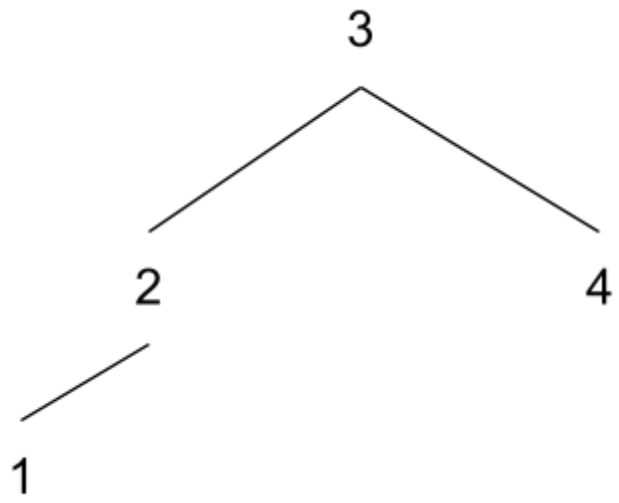
Heap:



Array: 2, 3, 4, 1, 5, 6, 7, 8, 9

Step 19: Max-Heapify, exchange 2 and 3

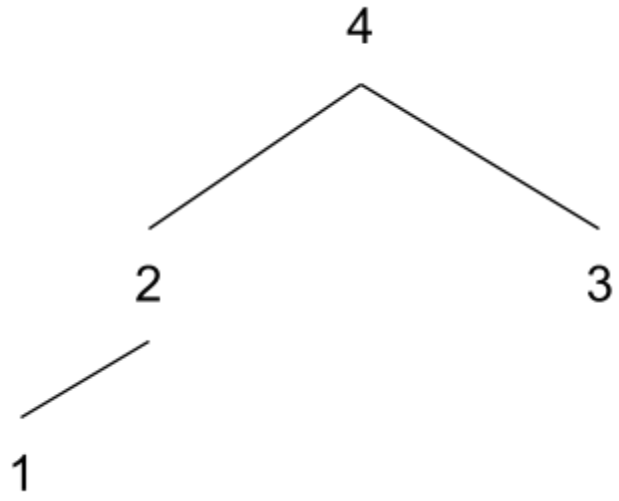
Heap:



Array: 3, 2, 4, 1, 5, 6, 7, 8, 9

Step 20: max-heapify, exchange 3 and 4

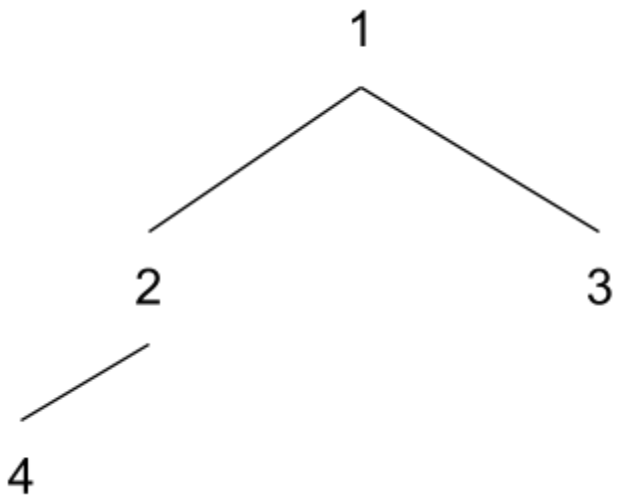
Heap:



Array: 4, 2, 3, 1, 5, 6, 7, 8, 9

Step 21: exchange A[1] and A[4]

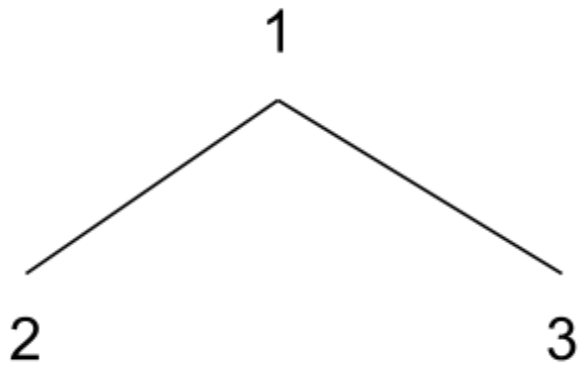
Heap:



Array: 1, 2, 3, 4, 5, 6, 7, 8, 9

Step 22: heap-size[A] = 3 so ignore 4

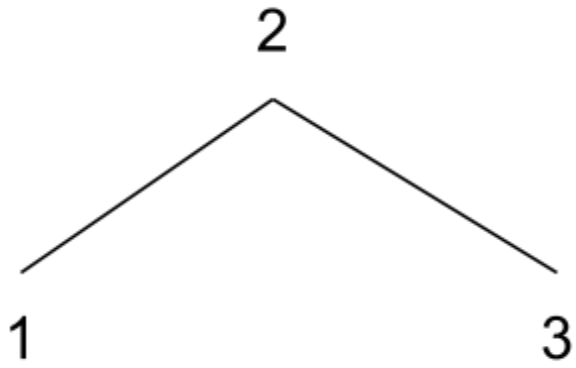
Heap:



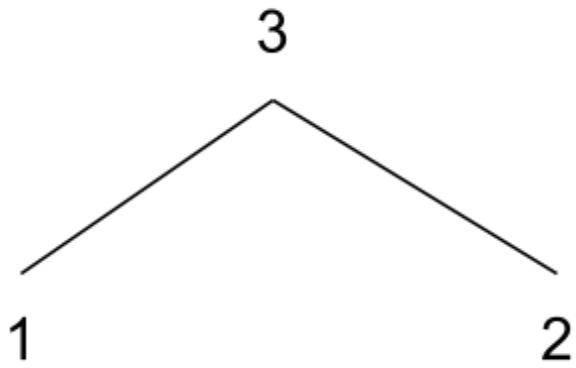
Array: 1, 2, 3, 4, 5, 6, 7, 8, 9

Step 23: Max-Heapify, exchange 1 and 2

Heap:



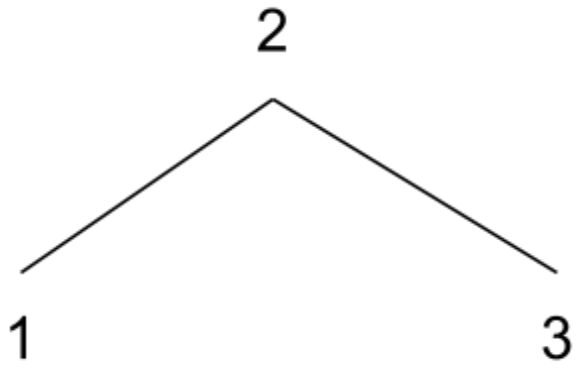
Array: 2, 1, 3, 4, 5, 6, 7, 8, 9
Step 24: max-heapify, exchange 2 and 3
Heap:



Array: 3, 1, 2, 4, 5, 6, 7, 8, 9

Step 25: exchange A[1] and A[3]

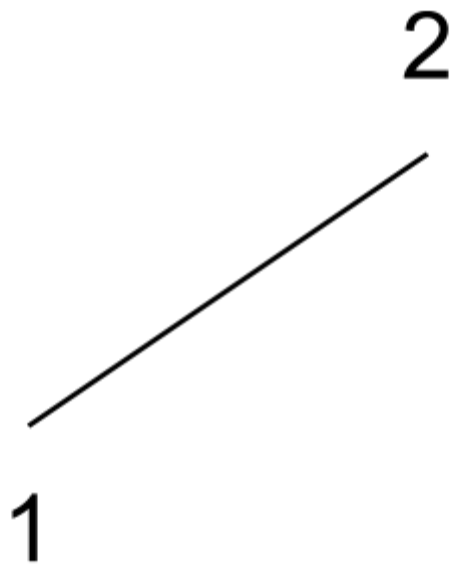
Heap:



Array: 2, 1, 3, 4, 5, 6, 7, 8, 9

Step 26: heap-size[A] = 2 so ignore 3

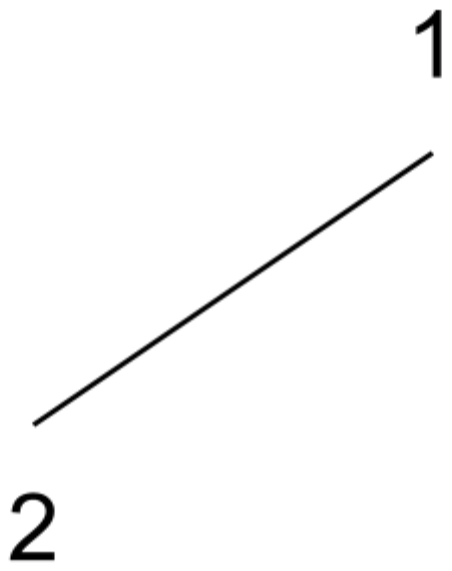
Heap:



Array: 2, 1, 3, 4, 5, 6, 7, 8, 9

Step 27: exchange A[1] and A[2]

Heap:



Array: 1, 2, 3, 4, 5, 6, 7, 8, 9

Step 28: heap-size[A] = 1 so ignore 2
Heap:

Array: 1, 2, 3, 4, 5, 6, 7, 8, 9

Note: the array was sorted at Step 22, but the algorithm required further processing, so I showed the extra steps it would take to run to completion, i.e., when the heap became sized 1.

#2 [10 points]

The analysis of recursive algorithms gives rise to recurrence relations. For Merge Sort, when n is a power of 2, the recurrence relation is:

$T(n) = 1$ if $n = 1$

$T(n) = 2T(n/2) + n$ if $n > 1$

Using the recursion tree for merge sort, we were able to solve this recurrence relation into its closed form solution: n log n + n

Prove by induction that $T(n) = n \log n + n$ and hence is $O(n \log n)$

To prove by induction that $T(n) = n \log n + n$, we will use strong induction, assuming that the equation holds for all values of k up to n.

Base case: When $n = 1$, $T(1) = 1 \log 1 + 1 = 1$, which satisfies the equation.

Inductive step: Assume that $T(k) = k \log k + k$ holds for all values of k up to n.

We want to show that $T(n) = n \log n + n$ also holds.

Using the recurrence relation, we have:

$T(n) = 2T(n/2) + n = 2((n/2) \log (n/2) + n/2) + n$ (by the inductive hypothesis) $= n \log(n/2) + 2n = n (\log n - 1) + 2n = n \log n - n + 2n = n \log n + n$

Therefore, $T(n) = n \log n + n$ holds for all values of n.

To show that $T(n)$ is $O(n \log n)$, we need to find constants c and k such that $T(n) \leq c\, n \log n$ for all $n > k$.

Using the equation $T(n) = n \log n + n$, we can see that $T(n)$ is $O(n \log n)$ with $c = 2$ and $k = 1$. Specifically, for $n > 1$, we have:

$T(n) = n \log n + n \leq 2n \log n = c\, n \log n$

Therefore, we have shown that $T(n) = n \log n + n$ and is $O(n \log n)$.

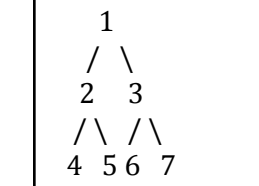
#3 [15 points]

Is there a heap T storing seven distinct elements such that a preorder traversal of T yields the elements of T in sorted order? How about an inorder traversal? How about a postorder traversal?

Let these distinct elements be: 1, 2, 3, 4, 5, 6, 7

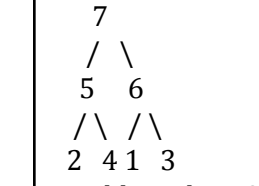
Note: min-heaps require children nodes to be >= parent nodes.
Max-heaps require children nodes to be <= parent nodes.

Preorder: Yes, the following min-heap is an example.



In preorder traversal, we visit the node before moving on. As such, the above example would visit the nodes in the order 1, 2, 3, 4, 5, 6, 7 which are sorted in ascending order.

Inorder: No, inorder traversal visits nodes starting going from the left-most leaf to the right-most leaf. Inorder traversal of the above min-heap example would produce 4, 2, 5, 1, 6, 3, 7 which are not sorted in any order.
A max-heap would produce a similar result. For example,



Would produce 2, 5, 4, 7, 1, 6, 3 which are also not sorted in any order.

Postorder: No, postorder traversal visits the left-child node, then the right-child node, then the parent node starting from the left-most node. Postorder traversal of the previous min-heap would produce 4, 5, 2, 6, 7, 3 ,1 which are not ordered in any way. Similarly, postorder traversal of the previous max-heap would produce 2, 4, 5, 1, 3, 6, 7 which are also not ordered in any way.

#4

(a) [15 points]

Briefly describe an algorithm for merging *k* sorted lists, each of length *n/k,* whose worst-case running time is O(nk). Clearly describe the algorithm in words, don't write the pseudo code. Explain mathematically that runtime is in fact O(nk).

The algorithm for merging k sorted lists of length n/k with a worst-case running time of O(nk)

can be done by using the approach called k-way merge. The idea is to pick the smallest element from each list and add it to the final sorted list. This process continues until all elements of all k lists are added to the final sorted list.

To achieve this, we can use a min-heap of size k, where we initially add the first element of each list. We then remove the smallest element from the heap and add it to the final sorted list. We then take the next element from the list that contained the smallest element we just removed and add it to the heap. We repeat this process until all elements from all lists are added to the final sorted list.

Mathematically, the worst-case time complexity of this algorithm can be shown to be O(nk). Each element is added to the final sorted list once, and there are n elements in total. Additionally, we perform k extract-min operations on the heap, each of which takes O(log k) time. Therefore, the overall worst-case running time is O(n log k). Since n/k is a constant factor, this simplifies to O(nk).

(b) [25 points]

Briefly describe an algorithm for merging *k* sorted lists, each of length *n/k,* whose worst-case running time is O(n log k). Clearly describe the algorithm in words, how you will implement it; don't write the pseudo code. Justify the running time. Explanation should be clear: doesn't have to be a mathematical derivation.

1. Create a min-heap of size k and insert the first element of each list into the heap.
2. Initialize an empty output list to store the sorted elements.
3. While the heap is not empty, repeat the following steps:
    a. Extract the minimum element from the heap and add it to the output list.
    b. Find the list that the extracted element belongs to and remove its first element.
    c. If the list is not empty, insert its new first element into the heap.
4. Return the output list.

In step 1, we are creating a min-heap of size k and inserting the first element of each list into the heap. By doing this, we are selecting the smallest element out of the k lists, which will be the first element in the final sorted list. The min-heap allows us to keep track of the smallest element across all the lists in constant time.

In step 3, we are repeating the process until the heap is empty. At each iteration, we extract the minimum element from the heap and add it to the output list. We then find the list that the extracted element belongs to and remove its first element. If the list is not empty, we insert its new first element into the heap.

The worst-case running time of this algorithm is O(n log k), where n is the total number of elements across all the lists. This is because we perform at most n operations, which correspond to adding each element to the output list once. Each operation takes O(log k) time to extract the minimum element from the heap and insert a new element into the heap. Therefore, the total time complexity is O(n log k).