Name: Sabyasachi Sahoo
COSC 40403 - Analysis of Algorithms: Fall 2018: Homework 3
Due: 23:59:59 on September 18, 2018

Question	Points	Score
1	5	
2	5	
3	5	
4	5	
5	5	
6	5	
7	5	
8	5	
9	5	
Total:	45	

1. (5 points) 6.1-1. What are the minimum and maximum number of elements in a heap of height h?

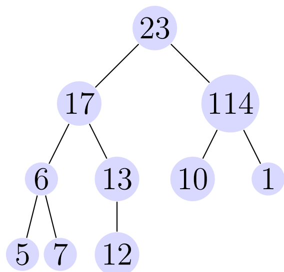
    Solution: Minimum: The minimum no. of nodes are when the last level has only one node i.e., 2^h.
    Maximum: The maximum no. of nodes are when the last level is completely full i.e., $2^{h+1} - 1$.

2. (5 points) 6.1-2. Show that an n-element heap has height $\lfloor \lg n \rfloor$.

Solution:
From Question-1 we get,
an n-element heap must satisfy $2^h \leq$ n $\leq 2^{h+1} - 1 \leq 2^{h+1}$.
Taking log on the sides we get, h $\leq$ lg n $\leq$ h+1.
So, h = lgn + α
where $0 \leq \alpha < 1$
Thus h = $\lfloor lgn \rfloor$.

3. (5 points) 6.1-6. Is the array with values $\langle 23, 17, 114, 6, 13, 10, 1, 5, 7, 12 \rangle$ a max-heap? Show your work by using computer software to draw the heap.

Solution:

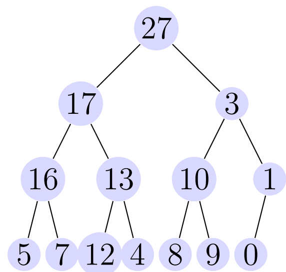


It is not a max-heap because (A,4) is not max-heapified as $6 < 7$.

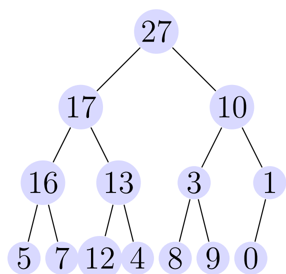
4. (5 points) 6.2-1. Using Figure 6.2 as a model, illustrate (with computer software) the operation of MAX-HEAPIFY$(A, 3)$ on the array $A = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$.

Solution:

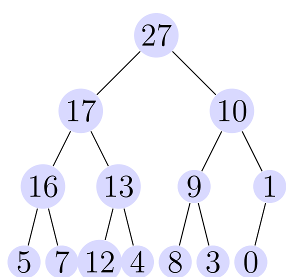
(i) Step 1:



(ii) Step 2:



(iii) Step 3:



5. (5 points) 6.2-2. Starting with the procedure MAX-HEAPIFY(A, i), write pseudocode for the procedure MIN-HEAPIFY(A, i), which performs the corresponding manipulation on a min-heap. How does the running time of MIN-HEAPIFY(A, i) compare to that of MAX-HEAPIFY$(A, 3)$?

Solution:

Min-Heapify(A,i):
l = left(i)
r = right(i)
if l $\leq$ A.heap - size and A[l] < A[i]

    smallest = l

else smallest = i

if r $\leq$ A.heap - size and A[r] < A[smallest]

    smallest = r

if smallest $\neq$ i

    exchange A[i] with A[smallest]

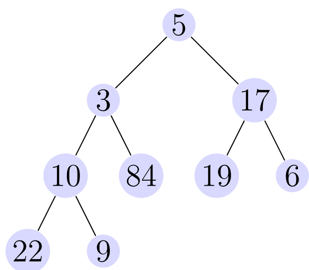
    Min-Heapify(A,smallest)

There is no change in the run time form Max-Heapify to Min-Heapify as the recurrence relation stays the same for the worst case i.e., T(n)$\leq$T(2n/3)+θ(1). Hence, the worst case running time is θ(lg n).

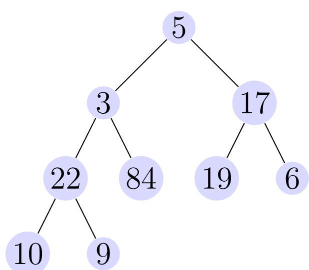
6. (5 points) 6.3-1. Using Figure 6.3 as a model, illustrate (using computer software) the operation of BUILD-MAX-HEAP on the array $A = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$

Solution:

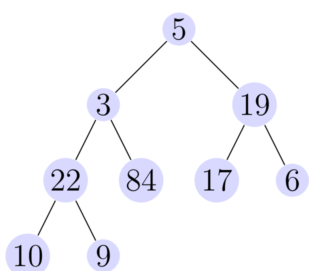
(i) BUILD-MAX_HEAP(A):



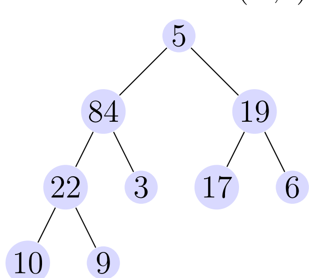
(ii) MAX_HEAPIFY(A,4):



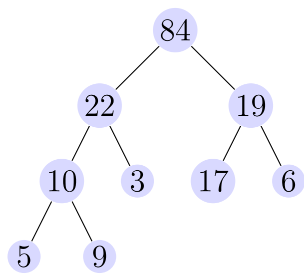
(iii) MAX_HEAPIFY(A,3):



(iv) MAX_HEAPIFY(A,2):



(v) MAX_HEAPIFY(A,1):



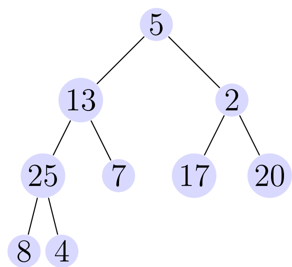
7. (5 points) 6.3-2. Why do we want the loop index i in line 3 of BUILD-MAX-HEAP to decrease from $\lfloor A.length/2 \rfloor$ to 1 rather than increase from 1 to $\lfloor A.length/2 \rfloor$?

Solution: We loop index to decrease from n/2 to 1, because we need to make sure that the precondition of Max-Heapify is be met before by calling it. Every index from n/2 + 1 to n is Max-Heapified as they are all leaves. So this way after each iteration i, both left and right subtrees of node i are Max-Heapified. Instead, if we go the other way we can destruct the heap property after each iteration.

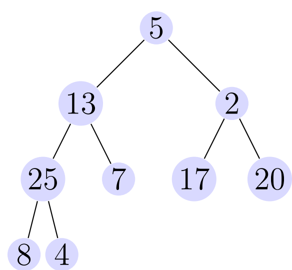
8. (5 points) 6.4-1. Using Figure 6.4 as a model, illustrate (with computer software) the operation of HEAPSORT on the array $A = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$.

Solution:

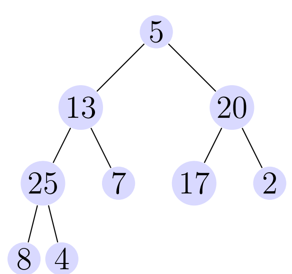
(i) Build-Max-Heap(A):



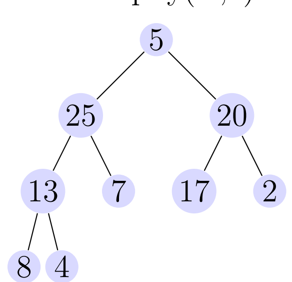
(ii) Max-Heapify(A,4):



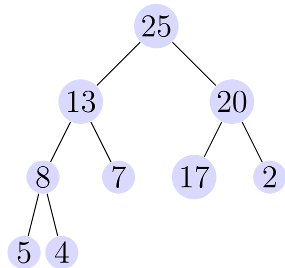
(iii) Max-Heapify(A,3):



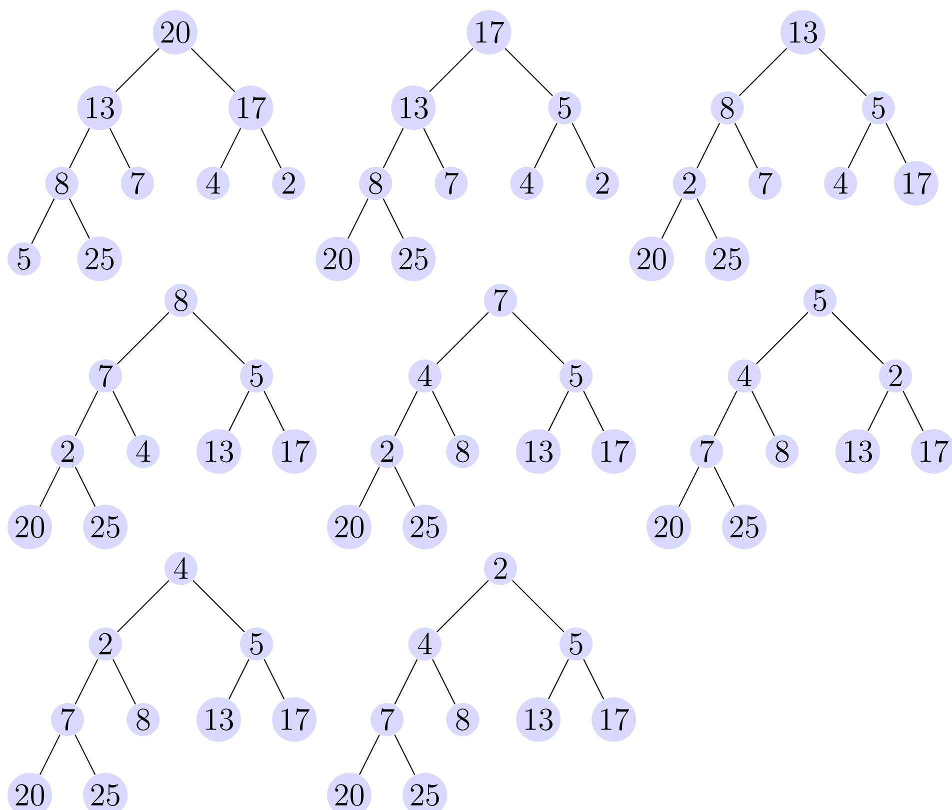
(iv) Max-Heapify(A,2):



(v) Max-Heapify(A,1):



(vi) Heap-Sort(A):



9. (5 points) 6.4-3. What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?

Solution:

If A is sorted in increasing order, Build-Max-Heap will attain the maximum running time of Θ(n). The n-1 calls Max-Heapify(A, 1) will take at most O(log(n)) time, hence the running time of Heapsort will be Θ(n lg(n)).

If A is sorted in decreasing order, Build-Max-Heap will be faster by a constant factor , still Θ(n) . Here as well the n1 calls Max-Heapify(A, 1) will take at most O(log(n)) time. Hence, the computation still remains the same and the running time of Heapsort will be Θ(n lg(n)).