CS6033 Design and Analysis of Algorithms I
Homework Assignment 5 Part 1
10.30.2022

Group Members:
rk4305 (Sai Rajeev Koppuravuri)
sg7372 (Sriharsha Gaddipati)
vt2184 (Veeravenkata Raghavendra Naveenkumar Tata)
vt2182 (Venu Vardhan Reddy Tekula)

---

1. (5 points) Show that the number of leaves in a heap is $\lceil n/2 \rceil$.

   Ans:

   Let $P(n)$ be the number of leaves in a binary heap with n nodes.

   $P(1) = 1 = \lceil 1/2 \rceil$ is true as we know that binary heap with one node has 1 leaft (itself).

   Hence, $P(1)$ is true.

   Let $P(n) = \lceil n/2 \rceil$ be true for $n = k$.

   $P(k) = \lceil k/2 \rceil$

   Now, let $n = k + 1$.

   Now me know that, if k is even then addition of $(k+1)^{th}$ element will increase the number of leaves by 1 but if k is old them addition of k^{th} element will have no impact on the number of leaves.

   $$P(k+1) = \begin{cases} P(k) & \text{if k is odd} \\ 1 + P(k) & \text{if k is even} \end{cases}$$

   $$P(k+1) = \begin{cases} \lceil k/2 \rceil & \text{if k is odd} \\ 1 + \lceil k/2 \rceil & \text{if k is even} \end{cases}$$

   also $1 + \lceil k/2 \rceil = \lceil (1+k)/2 \rceil$ if k is even

   and $\lceil k/2 \rceil = \lceil (k+1)/2 \rceil$ if k is odd

   Hence,

   $P(k+1) = \lceil (k+1)/2 \rceil$

   $P(k)$ and $P(k+1)$ are true then $P(n) = \lceil n/2 \rceil$ is true for all n.

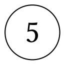
2. (10 points) Illustrate the operation of BUILD-MAXHEAP on the array A = [5, 3, 17, 10, 84, 19, 6, 22, 9]. Show your work step by step showing array and return the final MAX HEAP array A.

   Ans:

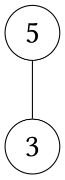
   Given, A [5, 3, 17, 10, 84, 19, 6, 22, 9]

   BUILD-MAXHEAP Steps

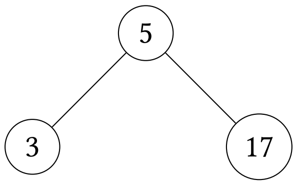
   1. insert 5



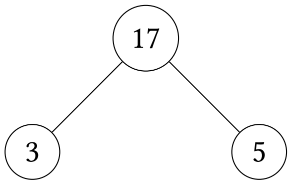
2. insert 3



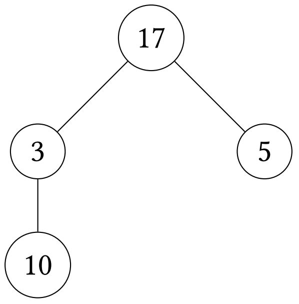
3. insert 17



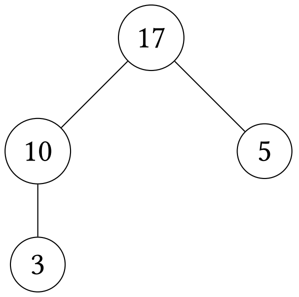
as $17 > 5$, swap the nodes



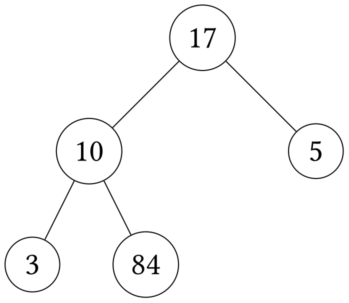
4. insert 10



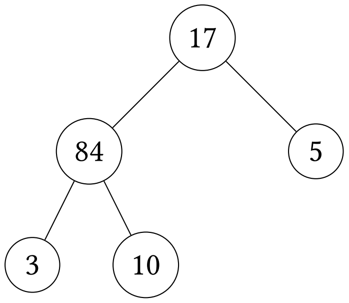
as $10 > 3$, swap the nodes



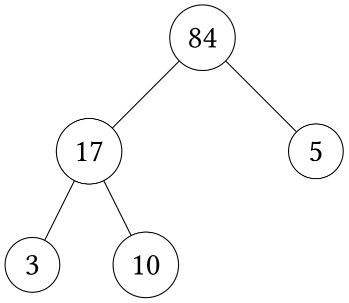
5. insert 84



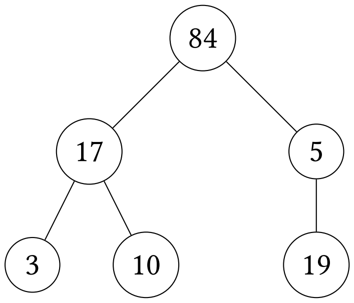
as $84 > 10$



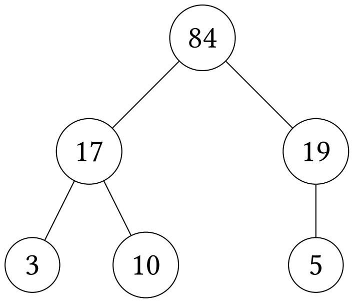
as $84 > 17$, swap the nodes



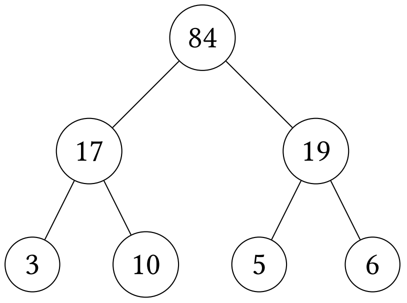
6. insert 19



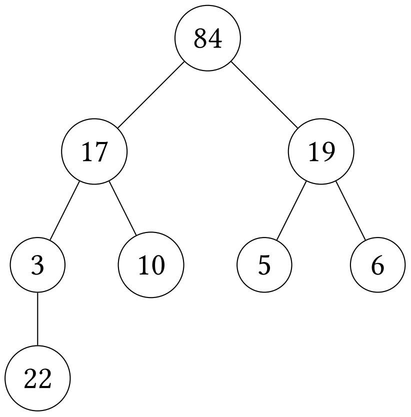
as $19 > 5$, swap the nodes



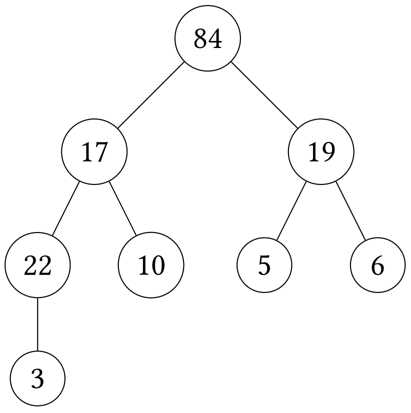
7. insert 6



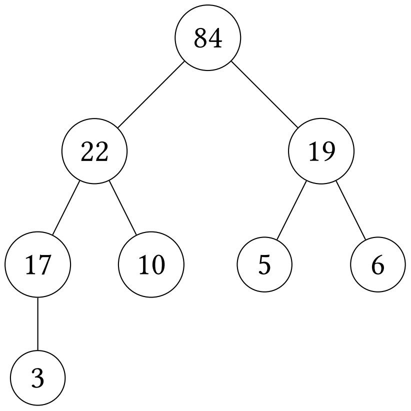
8. insert 22



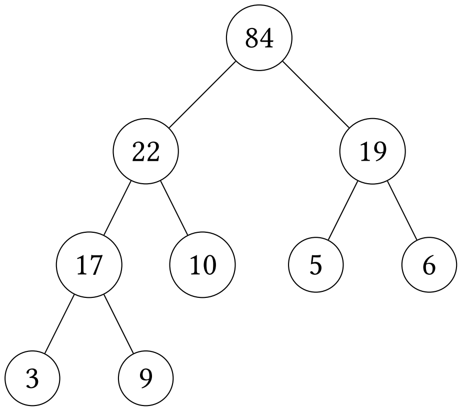
as $22 > 3$, swap the nodes



as $22 > 17$, swap the nodes



9. insert 9



The final MAX HEAP array A is [84, 22, 19, 17, 10, 5, 6, 3, 9].

3. (5 points) Let suppose we have a toolkit of algorithms. For heap operations it has only single subroutine which is MIN-HEAP. How will we then implement MAX-HEAP functionality using only given MIN- HEAP subroutine.

   Ans:

   In order to implement the MAX-HEAP functionality, using only the MINHEAP subroutine we will add one additional preprocessing step. Before running the MINHEAP subroute, we will iterate through the input array and multiply each value by -1. By doing this, the bigger the number will become negative (smaller) after the preprocessing. And hence the MIN-HEAP subroutine will now operate with a MAX-HEAP functionality.

   Therefore for operations insert, EXTRACT-MAX we need to follow this procedure-

   To Insert, we need multiply with -1 to the value and then insert in MIN-HEAP.
   To EXTRACT-MAX we need to get the minimum using EXTRACT-MIN from MIN-HEAP and then multiply with -1 and return the element.

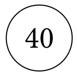
4. (15 points) For the following array A = [40, 6, 3, 11, 2, 4]

   (a) (5 points) Create a max heap using the algorithm discussed in class (BUILD-MAX-HEAP).

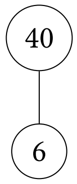
   Ans:

Approach 1:

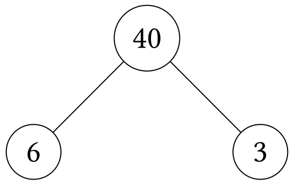
Step 1



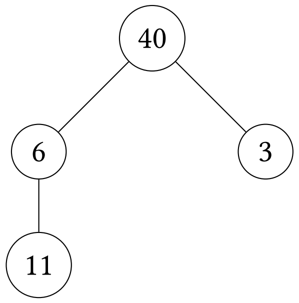
Step 2



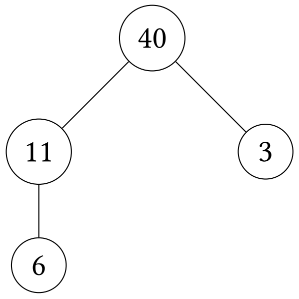
Step 3



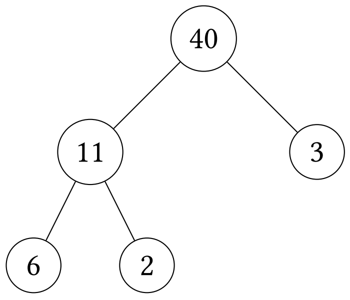
Step 4



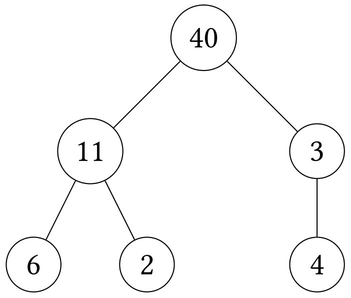
as $11 > 6$, swap the nodes



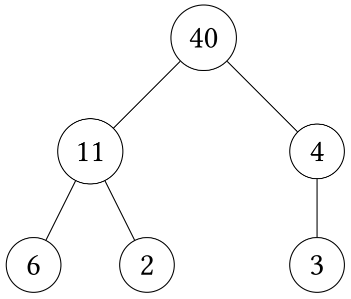
Step 5



Step 6

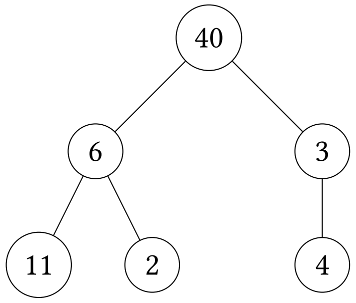


as $4 > 3$, swap the nodes

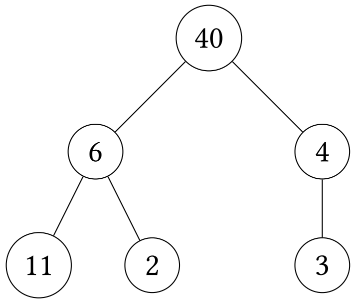


The max heap array is [40, 11, 4, 6, 2, 3].

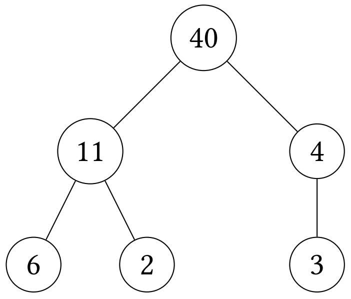
Approach 2:



The nodes 3 and 4 are exchanged



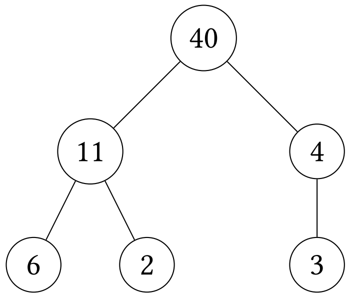
The nodes 6 and 11 are exchanged



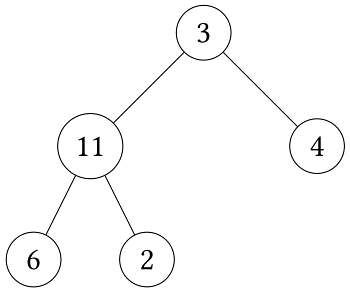
MAX-HEAPIFY will be called on the nodes 40 but the child values are less than the parent so, the heap remains same.

(b) (5 points) Remove the largest item from the max heap you created in (a), using the HEAP- EXTRACT-MAX function from the book. Show the array after you have removed the largest item.

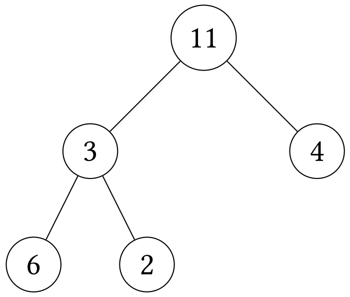
Ans:

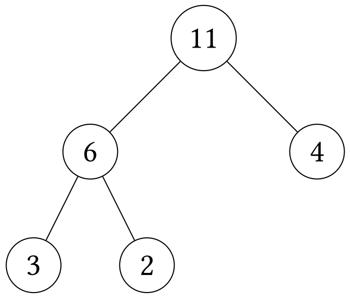


The largest element in the heap is 40, removing 40 from the heap using HEAP- EXTRACT-MAX function and the leaf node now becomes the root node.



Heapify the above heap

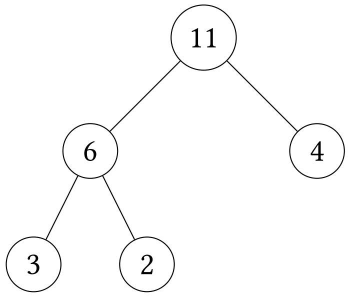




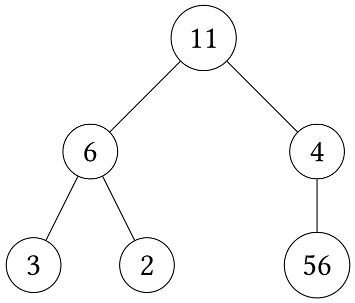
The array now is [11, 6, 4, 3, 2].

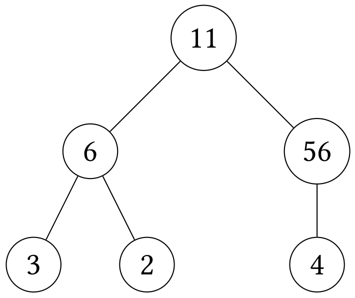
(c) (5 points) Using the algorithm from the book, MAX-HEAP-INSERT, insert 56 into the heap that resulted from question (b). Show the array after you have inserted the item.

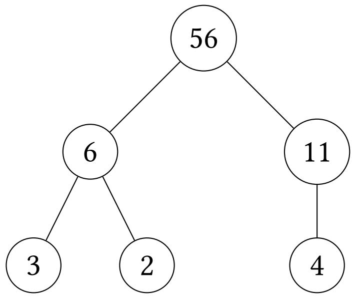
Ans:



Inserting the element 56 into the heap using MAX-HEAP-INSERT function.







5. (15 points) Now that your a graduate student, you are bombarded with things to do! You have to make sure you play FFXIV, re-clear Skyrim for the 300th time, and do your homework assignment for CS6033. Of course, as you are trying to finish all your tasks a new task might arise (your roommate might hit you up for a game of League, or you might remember you promised to call your parents last weekend) and you have to make sure it gets done.

   You decide to keep working until all the tasks are done because you want to enjoy your weekend. In a attempt to build momentum, you decide to complete the tasks in order of the time you estimated they would take. Devise an algorithm that will tell you the next task to do in $O(\log n)$ time, as well as integrate new tasks in $O(\log n)$.

   Write a puesdocode and Justify the running time of your algorithm.

   Ans:

We can solve this problem using a MIN HEAP. A list of pairs (tuples) can be used to contain all of the tasks, with the task name appearing as the first element and the estimated completion time appearing as the second. Let's say this list is $A(task\_name, completion\_time)$.

Then, we build a min-heap on this list of tasks using the BUILD-MIN-HEAP algorithm. The estimated time of a task, which is the second factor in the pair, is used to build the heap. The MIN-HEAPIFY procedure, which maintains the min-heap property at index is called internally by the BUILD-MIN-HEAP process. The process of comparing and swapping along the heap is continued by MIN-HEAPIFY until the subtree is a MIN-HEAP. MIN-HEAPIFY has $O(\log n)$ running time.

We have a MIN HEAP, where the root node now has the task which takes the least amount of time that is required to complete a task. We can use the HEAP-EXTRACT-MIN algorithm to extract the root element. We pop the root element and replace it with the last element of the heap. To restore the MIN-HEAP property to the remaining elements, we run MIN-HEAPIFY(A, 1) after popping the element at the root and replacing it with the last element of the heap. This process has an $O(\log n)$ running time since it calls MIN-HEAPIFY while the other steps are constant in duration.

We use the MIN-HEAP-INSERT method to insert new tasks. In this case, we increase the heap size by one for insertion. We next call HEAP-DECREASE-KEY after inserting an initial value of infinity for the new task (A, A.heap-size, new-task). This modifies the heap so that the new task is placed properly within it. HEAP-DECREASE-KEY takes an upward path towards the root from the end of the heap in this example until we locate the ideal place for the current task, hence running time is $O(\log n)$. As a result, MIN-HEAP-INSERT running time is equal to HEAP-DECREASE-KEY time, $O(\log n)$.

Once the MIN-HEAP has been constructed, determining the subsequent task to be completed using HEAP-EXTRACT-MIN would take $O(\log n)$ time. We would need $O(\log n)$ time to insert the sew task into the heap using MIN-HEAP-INSERT for integrating a new task.

psuedo code

```
procedure MIN-HEAPIFY(A, i):
$l \leftarrow Left(i)$
$r \leftarrow Right(i)$
**if** $l \leq |A|$ and $A[l] \leq A[i]$ **then**
    $min \leftarrow l$
**else**
    $min \leftarrow i$
**end if**
**if** $r \leq |A|$ and $A[r] \leq A[min]$ **then**
    $min \leftarrow r$
**end if**
    // Update min - heap tree
**if** $min \neq i$ **then**
    Exchange A[i] and A[min]
    MIN-HEAPIFY(A, min)
```

```
end if
end procedure
```

//

```
procedure HEAP-EXTRACT-MIN(A):
if $A.heap\_size \leq 1$ then
    error ”underflow”
end if
$min \leftarrow A[1]$
$A[1] \leftarrow A[A.heap\_size]$
$A.heap\_size = A.heap\_size - 1$
MIN-HEAPIFY(A, 1)
return min
end procedure
```

//

```
procedure HEAP-DECREASE-KEY(A, i, key):
if $key \geq A[i]$ then
    error ”new key is larger than the current key”
end if
$A[1] \leftarrow key$
while $i \geq 1$ and $A[PARENT(i) \geq A[i]$ do
    Exchange A[i] and A[PARENT(i)]
    $i \leftarrow PARENT(i)$
end while
end procedure
```

//

```
procedure MIN-HEAP-INSERT(A, key):
$A.heap\_size = A.heap\_size + 1$
$A[A.heap\_size] = \infty$
HEAP-DECREASE-KEY(A, A.heap_size, key)
end procedure
```

The running time to construct a total MIN-HEAP is $O(n \log n)$.

Insertion of a new task and deletion takes $O(\log n)$ time.