Assignment-2

Question - 1 :-

Algo:- 1) int low = 0 , high = arr.length - 1

2) while (low < high)

mid = (low + high) / 2

if A[mid] < A[mid + 1]

Set high = mid

else

set low = mid + 1

end while

3) return A[low]

Time complexity : O(log n)

Example : Let arr = [10, 8, 7, 5, 1, 6, 9] (0 1 2 3 4 5 6), (expected ans is 1)

① mid = $\frac{0+6}{2} = 3$

A[mid] < A[mid + 1] (false)

So , now low = 4 , high = 6

② mid = $\frac{4+6}{2} = 5$

A[5] < A[6] [true]

So, now ~~low~~ high = 5

and low = 4.

③ mid = $\frac{5+4}{2} = 4$

A[4] < A[5] [true]

So now high = 4

and low = 4

④ while (low < high) (false)

So, return A[low] (which is 1)

which is our expected ans.

Question-2 :-

(1) Given array = [28, 52, 17, 35, 24, 48, 11, 20, 17, 30]

(1) Start with j = 1, A[j] = 52.

A[1] > A[0]

no swap required.

So array = [28, 52, 17, 35, 24, 48, 11, 20, 17, 30]

(2) now j = 2,

A[2] < A[1]

swap them

A[1] < A[0]

swap them.

So, now array = [~~28, 52~~, 17, 28, 52, 35, 24, 48, 11, 20, 17, 30]

(3) Now j = 3

A[3] < A[2]

swap them.

now array = [17, 28, 35, 52, 24, 48, 11, 20, 17, 30]

(4) Now j = 4.

shift A[4] to A[1]

now array = [17, 24, 28, 35, 52, 48, 11, 20, 17, 30]

(5) Now j = 5

~~shift~~ A[5] < A[4], so swap

Now array = [17, 24, 28, 35, 48, 52, 11, 20, 17, 30]

(6) Now j = 6

shift A[6] to A[0] As A[6] < A[0].

Now array = [11, 17, 24, 28, 35, 48, 52, 20, 17, 30]

(7) Now j = 7

shift A[7] to A[2] as A[7] < A[3]

Now array = [11, 17, 20, 24, 28, 35, 48, 52, 17, 30]

(8) Now j = 8
Shift A[8] to A[2] as A[8] < A[3]
now array = [11, 17, 17, 20, 24, 28, 35, 48, 52, 30]

(9) Now j = 9
Shift A[9] to A[6] as A[9] < A[7].
now array = [11, 17, 17, 20, 24, 28, 30, 35, 48, 52]

(2) ~~Recursive version of Insertion Sort~~
worst case : $O(n^2)$
Best case : $O(n)$
Avg case : $O(n^2)$

(7) Recursive version of Insertion Sort

insertionsort (arr, n)
loop from i = 1 to n-1
pick element arr[i] and insert it in it's suitable position.

Code:-
```
insertionsort (int arr[], int n).
{ if (n<=1)
    return;
  insertionsort(arr, n-1);
  int last = arr[n-1];
  int j = n-2;
  while (j>=0 && arr[j] > last)
  { arr[j+1] = arr[j];
    j--;
  }
  arr[j+1] = last;
}
```

Worst case : $T(n) = T(n-1) + O(n)$.
$\rightarrow T(n) = O(n^2)$

Best case : $T(n-1) + O(1)$

$T(n) = O(n)$

Avg case : $O(n^2)$

So it has the same Time complexity as iterative version.

(iii) Verify the online nature of the INSERTION SORT algorithm with an example.

→ Insertion sort is considered an "Online" algorithm because it can handel the upcoming data very effeciently.

→ Example we have an initially empty array, and we reciever elements one by one : 7, 4, 5, 2, 10

1. initial array
2. insert 7 : [7]
3. insert 4 : [4, 7]
4. insert 5 : [4, 5, 7]
5. insert 2 : [2, 4, 5, 7]
6. insert 10 : [2, 4, 5, 7, 10]

Question - 3

i) Write the recursive version of the BUBBLE SORT algorithm.

```
static void bubbleSort (int arr[], int n)
{
    if (n == 1)
        return;
    int count = 0;
    for (int i = 0; i < n-1; i++)
        if (arr[i] > arr[i+1])
        { swap(arr[i], arr[i+1]
        ; count++;
```

```
if (count == 0)
    return;
bubblesort (arr, n-1);
}
```

ii) Formulate the recurrence relation for its running times in the worst and the best cases.

Best case: $T(n) = T(n-1) + O(1)$

$\rightarrow T(n) = O(n)$

Worst case: $T(n) = T(n-1) + O(n)$

$\rightarrow T(n) = O(n^2)$

Avg case: $T(n) = T(n-1) + O(n)$

$= O(n^2)$

(iii) BUBBLESORT is stable or not ? justify your answer with a suitable example.

→ Bubblesort is stable.

→ A sorting algorithm is said to be stable if two element after swap is also a sorted pair.

→ It always maintain the relative order.

→ Example. Let an initial array [23, 15, 23*, 40].

1st step → [15, 23, 23*, 40].

2nd step [15, 23, 23*, 40].

3rd step [15, 23, 23*, 40].

So you can see the element 23 with a star is not swapped and present in its relative position to 23.

So Bubble sort is a stable algorithm.

Question-4: (i) Recursive version of the SELECTION SORT algorithm.

```
static void recursiveSelectionSort (int[] a, int n,
                                    int index)
{
    if (index == n)
    return ;
    int k = minIndex (a, index, n-1);
    if (k != index)
    {  swap (a[k], a[index]);
    }
      recurseSelectionSort (a, n, index+1);
}
static int minIndex (int a[], int i, int j)
{
    if (i == j)
      return i;
    int k = minIndex (a, i+1, j);
    if (a[i] < a[k])
       return i ;
      return k;
}
```

(ii) Formulate the recurrence relation for its running times in the worst and the best cases.

Best case: $T(n) = T(n-1) + O(1)$
$= O(n^2)$

worst case: $T(n) = T(n-1) + O(1)$
$= O(n^2)$

Avg case: $T(n) = T(n-1) + O(n)$
$= O(n^2)$

(iii) SELECTION SORT is Stable or not?

→ Selection Sort is not a Stable sorting algorithm. because it swaps according pivot's position

(iv) Can you observe any resemblence with the operation of HEAPSORT algorithm: Explain with example:

→ HEAPSORT algorith works some what like selection SORT concept.

→ Both are Unstable sorting.

→ For example: Let an array

[4, 2, 7, 1, 9, 5, 3]

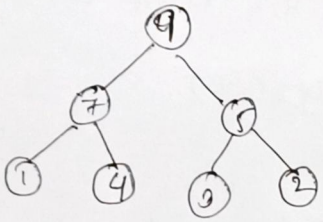
Selection SORT (1st Pass)

The minimum element is 1.

Swap it with 4.

Now array [1, 2, 7, 4, 9, 5, 3]

HEAP SORT 1st Pass



Swap the root (9) with last element 3)

Now array [3, 2, 7, 1, 4, 5, 9].

Question-5 : Design an efficient algorithm to check if k^{th} largest element in an n element heap is smaller than or equal to a given value x. Your algo should has O(k).

Algo:-

1) initialize c=0, r
2) Read k and x
3) for int i = 0 to n-1 do
   check is ith element of heap is less than or equal x
   if yes
   count = count + 1
   result = heap[i]
   check is count is equal to k
   if yes then break the loop
   i=i+1
   end for
4) check if count < k
   Print (Not enough element to search)
   else
   Print (result)

Time complexity :-

Best case : O(1)
worst case : O(K)

Question-6 :- Suppose we are comparing the implementations of Insertion sort and Heap Sort on the same machine. For input size n, Insertion sort runs in $8n^2$ steps, while Heap sort runs in $64 n \log n$ steps for which value of n, Insertion Sort outperforms Heap sort?

As Per Given instructions

$$8n^2 < 64 n \log n$$

$$n^2 < 8 n \log(n)$$

$$n < 8 * \log_2(n)$$

$$n/8 < \log n.$$

$$\Rightarrow 2^{n/8} < n.$$

So, $2 \leq n \leq 43$ insertion sort beats merge sort.

Question-7 :- In a binary max-heap containing 100 elements, if the root is at Position one.

i) Find the Position of the Parent of a node i that is Present at Position 19 and check if i is a left or right child of its parent.

$$\text{Parent node} = \left\lfloor \frac{19}{2} \right\rfloor$$

$$\Rightarrow i = 9.$$

As i is odd it is a right child.

(ii) What is the height of the node i?

We know height of node n is $[\log n]$

$$\Rightarrow \text{height} = [\log 19]$$
$$= [4.247927]$$
$$= 4$$

(III) Find number no of nodes with two children, one child, and no child.

As the no of nodes is 100 then. the no of leaves. $\lfloor \frac{n}{2} \rfloor$

$= \lfloor \frac{100}{2} \rfloor$

$= 50$

So as leaves has 0 leaves so 50 nodes has zero child.

As n is even so no of 1 child is 1 node.

The number of nodes having 2 children.

$= 100 - (50+1)$

$= 100 - 51$

$= 49$

(IV) Number of nodes in Left subtree and right subtree.

~~Left subtree~~ =

As binary tree is ACBT. So Number of

left subtree = 63

right subtree = 36

(V) Position of the Parent of fifth node at a height = 3.

at height 3 the node number. $= 2^3$ to $2^4 - 1 = 15$

fifth node present at $= 8 + 5 - 1 = 12$.

Parent of the node is $= \lfloor \frac{12}{2} \rfloor$

$= 6.$

(8) A binary heap contains 100 elements.

(i) Position of the third leaf node.

the position of leaves are $\lfloor \frac{n}{2} \rfloor + 1$ to n.

So the position of third leaf

$= \lfloor \frac{n}{2} \rfloor + 1 + 2$

$= \lfloor \frac{100}{2} \rfloor + 3$

$= 53$

(ii) Total number of leaves.

Total number of leaves $= \lfloor \frac{n}{2} \rfloor = 50$

(iii) Height of the heap

height of the heap $= \lfloor \log_2 100 \rfloor$

$= \lfloor 6.643 \rfloor$

$= 6.$

(iv) No of nodes at height 3.

~~No of nodes = 2 3 8~~

No of nodes at height 3 is 8

9) Derive a formula to compute the height of the k^{th} largest element in a binary max-heap. Using that compute 35th largest element of a heap [60..1]

$$K = \left(\frac{1 - 2^n}{-1} \right) + 1$$

For 35th largest element.

$K = 35$

$35 = \left(\frac{1 - 2^n}{-1} \right) + 1$

$\Rightarrow 34 = 2^n - 1$

$\Rightarrow 35 = 2^n$

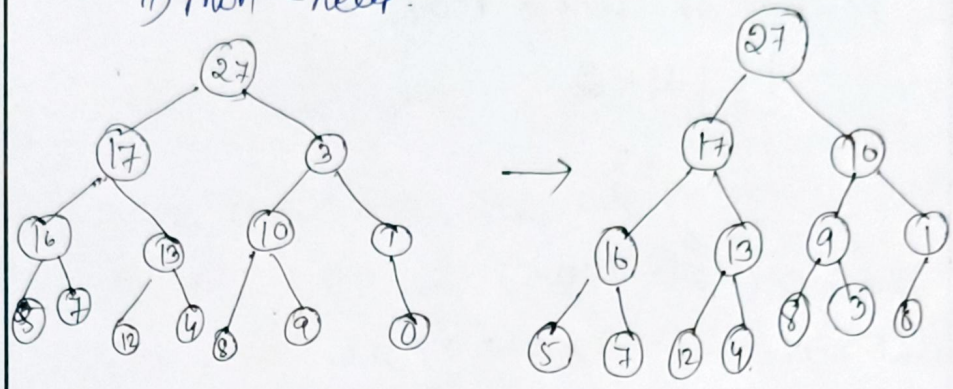
$\Rightarrow n = \log_2(35) = 5.129$

$\Rightarrow n = 5$

So the 35th largest element is present at a height of 5

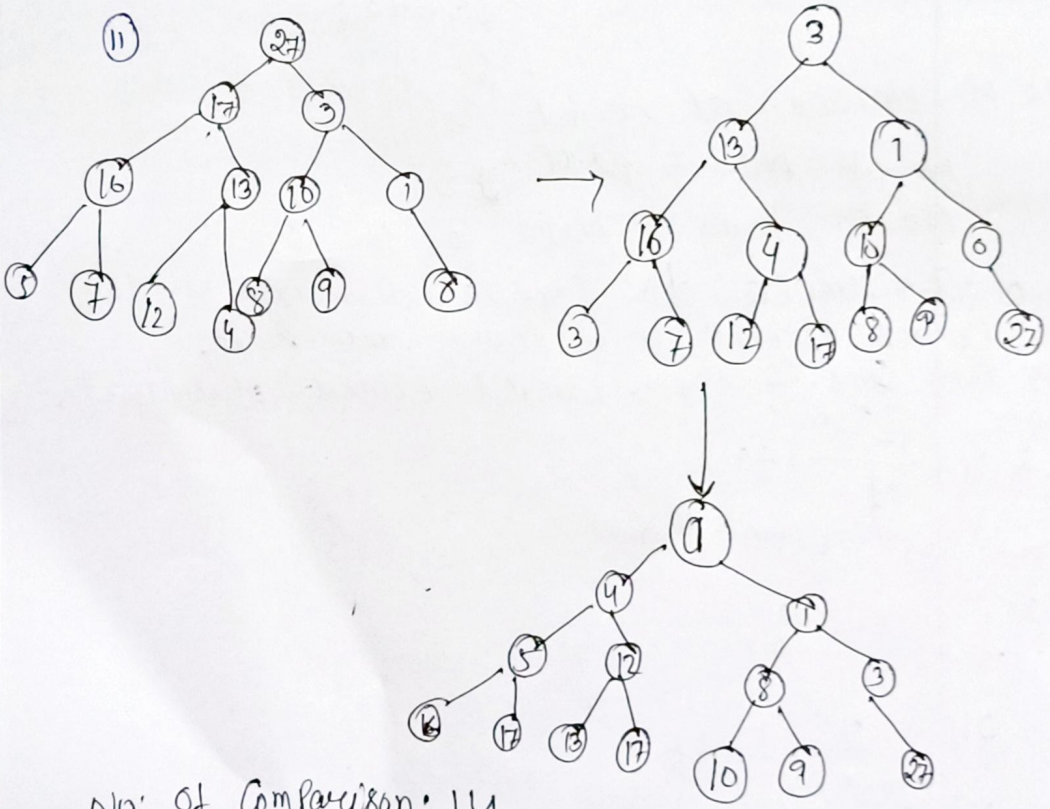
Question:10:. convert the given array A =
<27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0> into a

i) max-heap
ii) min-heap.



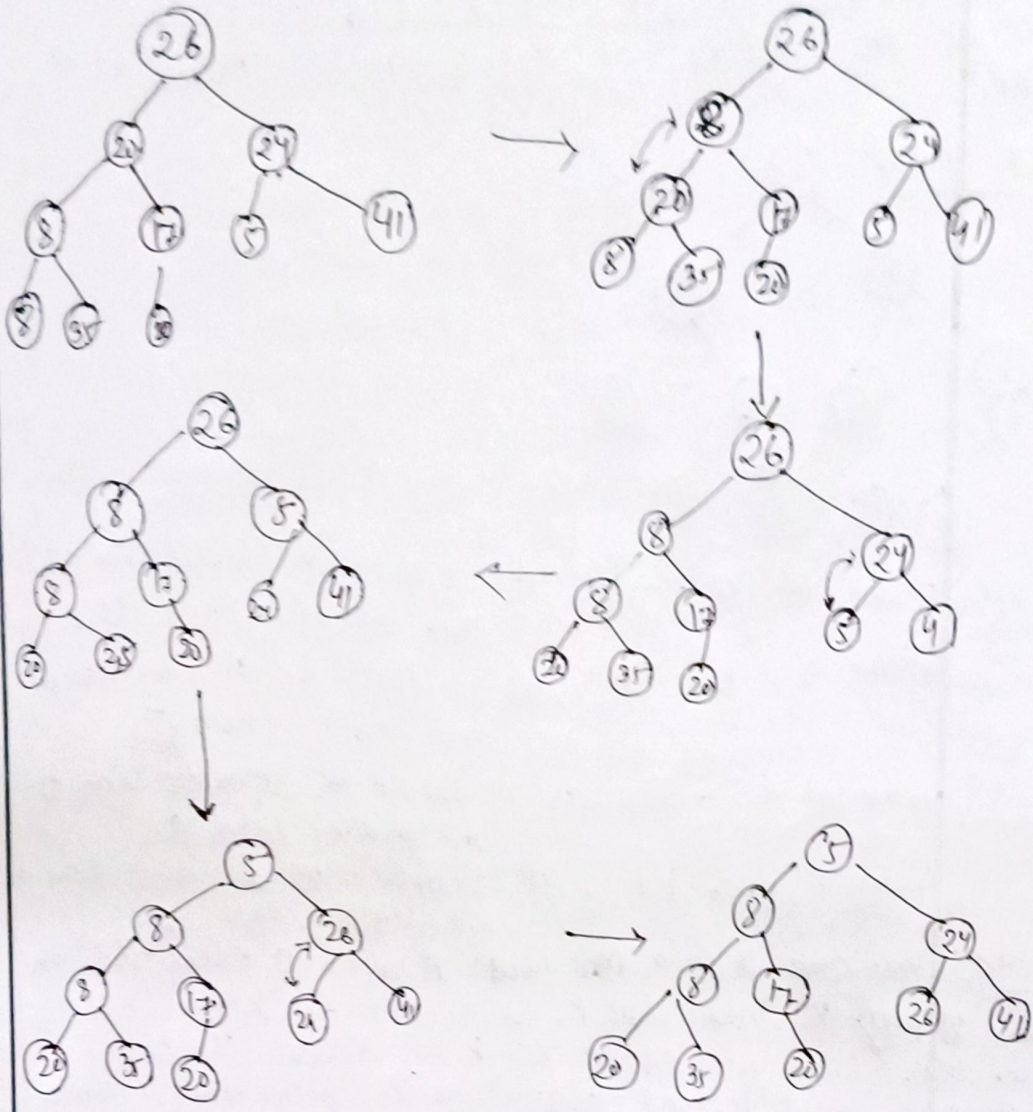
No of comparison = 14
No of max-heap call = 2.

ii)



No: of Comparison: 14
No. of min-heap call: 11.

Question-11: Sort the array A = <26, 20, 24, 8, 17, 5, 41, 8, 35, 20> in descending order using heap sort and find the total number of calls to the MIN-HEAPIFY procedure.



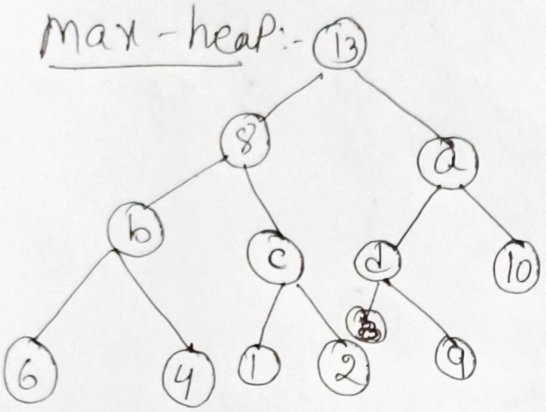
Sorted array = <41, 35, 26, 24, 20, 20, 17, 8, 8, 5>

Question 12:- The following array A contains the distinct positive integers from 1 to 13.

13	8	a	b	c	d	10	6	4	1	2	13	9

What will be the values of a, b, c and d so that it is a max-heap?

Max-heap:-



So the value of a = 12. as it should be small than 13 and greater than 10.

Value of b = 7, as it should be greater than 6 and smaller than 8.

Value of c = 5, as it should be greater than 2 and smaller than 8.

Value of d = 11, as it should be greater than 9 and smaller than 12.

Question-13:- You are given a Min-heap H with n elements and a search key K. Your task is to print all keys in H that are less than or equal to K in sorted order.

i) Purpose an efficient algorithm to solve this problem

ii) Supply a tight bound on the running time of your algorithm

① Read key K.

② If POS >= heapsize
then return
If Heap[POS] >= X.
then return

(3) Print harm [POS]
(4) Call same function (x, left (POS))
(5) Call same function (x, right (POS)).

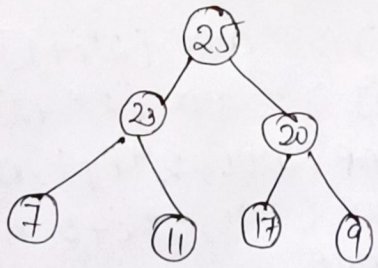
- Initialize the left function
  return (2*i+1)
  Initialize the right function
  return (2*i+2)

(11) Time complexity : O(n)

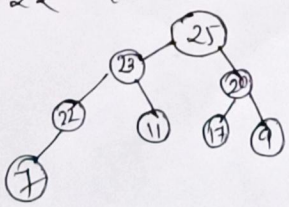
Question :14 :- Define a max-heap and its contiguous representation in an array.
i) Convert the array A = <20,11,25,7,23,17,9> into a max-heap
ii) Explain how you insert a key 22 in the above max-heap.

(1) A max heap is a specialized binary tree that satisfies the max-heap property.
(2) The max heap property states that for any given node in the heap the value of that node is greater than or equal to the value of its children

(i)



(ii) To insert 22 it can be inserted in the place of 7 as $7 \leq 22 \leq 23$



Question-15: Does Heap sort follow stable sorting Property? Justify with an example.

→ Heap sort is not a stable sorting algorithm because during steps the heap sort it doesn't guarantee that the preservation of relative elements stay.

→ For example an array [4*, 3, 2, 4, 1] In this array 4* and 4 are two equal elements but the output will [1, 2, 3, 4, 4*] we can clearly see the relative element 4, 4* has n't stayed in their relative positions.

→ Hence heap sort isn't a stable sorting.

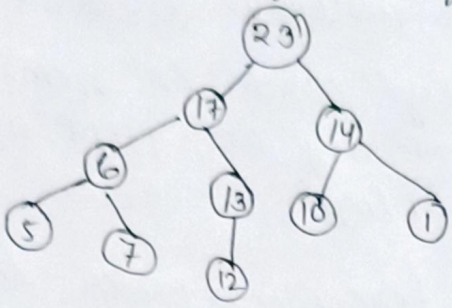
Question-16: write a Pseudocode to check it's a max-heap or not. Then check with values <23, 17, 14, 6, 13, 10, 1, 5, 7, 12> a max-heap? If not then build a max heap from the given array.

Pseudocode:-

* Read n and all elements in the arr.
* Create a boolean function isHeap(int arr[], int i, int n).

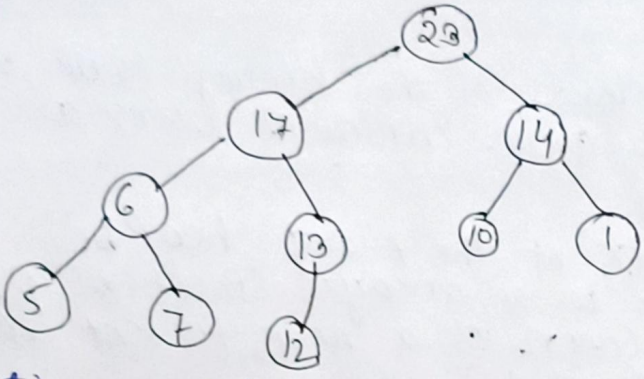
```
if (i >= n-1/2)
    return true
if (arr[i] >= arr[2*i+1] &&
    arr[i] >= arr[2*i+2] &&
    isHeap(arr, 2*i+1, n) &&
    isHeap(arr, 2*i+2, n))
    return true.
return false.
```

(61) Given elements $A = \langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$
Putting these elements in this Pseudocode it has returned false
we can see it by building a tree.

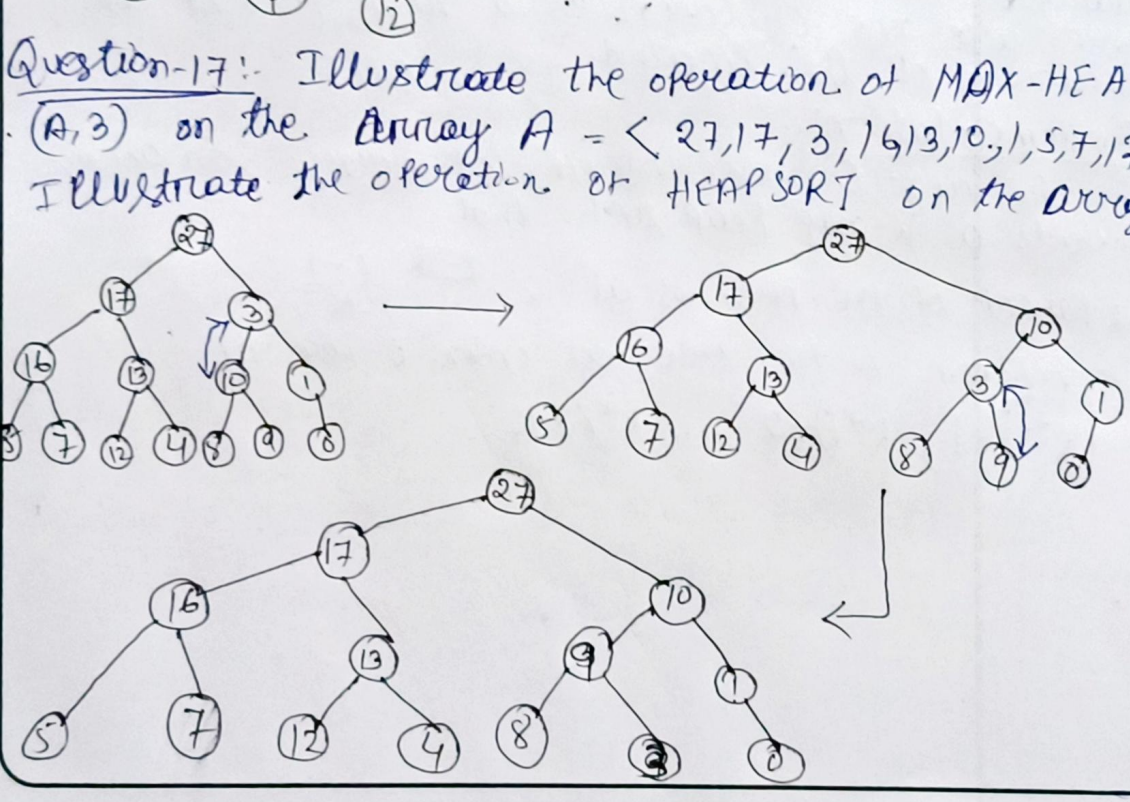


→ It is not a max-heap as $6 < 7$

So the max-heap is.



Question-17:- Illustrate the operation of MAX-HEAPIFY (A, 3) on the Array $A = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$
Illustrate the operation of HEAPSORT on the Array A



Question-18:- What is the running time HEAPSORT algorithm to sord an array of length n is ascending order that is already sorted in the desired order? what will be the running time if the array elements are already sorted in descending order?

→ If array is already sorded in ascending order then the Time complexity of HEAPSORT algorithm is $T(n) = O(n \log n)$.

→ If the array is in descending order then for a number of inputs it also takes $O(n \log n)$ times.

Question-19:- Suppose instead of the binary heap, we implement the Priority queue operations using a K-array ($K \geq 2$) heap.

i) How can the elements of the K-ary heap be arranged implicitly in an array? Specifically, where are the children of a node present at position i of the array located?

→ **Implementation:-**

Assuming 0 based indexing of array; an array represents a K-ary heap such that-

* Parent of the node is at; $\frac{i-1}{K}$
* children of the node at index i are at $(K*i)+1, (K*i)+2 \ldots K*i+K$

(ii) Analyze the Priority que operations (INSERT, EXTRACTMIN) implemented in a k-ary heap using the standard algo. What are the number of comparisons performed in the worst case for these operation.

→ For Insert :-

For K number of nodes there will be be Each Level of the heap requires K-1 comparisons.

So the number of comparisons is : $(k-1) \cdot \log_k n$

Time Complexity : $T(n) = O(\log_k n)$

For EXTRACTMIN :-

Similar to insertion there will be K-1 comparison on $\log_k n$ equals so

the number of comparisions is : $(k-1) \cdot \log_k n$

Time Complexity : $O(\log_k n)$

(iii) Discuss about the optimal value of K to get the best performance on the number of comparisons used in these operation :

→ The Popular method elbow method which is used to determine the optimal value of K to perform the best.

→ The basic idea behind this method is that it plots the various values of cost with changing k.

→ As the value of of K increases, there will be fewer elements in the cluster.

→ Then the optimal value for K would be 4.

N) Suppose we are only interested in minimizing the number of moves performed in the Priority queue operations. What value of K would you choose?

→ As $K \geq 2$, in a increasing manner.

The larger the K will the number of swaps will be reduced. it will be efficient.

(Exception:- in some rare cases ~~some~~, the lesser value of K will take more time if the Priority queue has machinary Pre-defined)

2) Formulate the recurrence relations for BUILD MINHEAP and MIN-HEAPIFY operation and derive the tight upper bounds on the running time of these procedures. BUILD MINHEAP creates a min-heap from the given array while MIN-HEAPIFY recursively maintains the min-heap property

Building a Binary heap = $T(n) = 2T(n/2) + O(n)$

$= \Theta T(n) = O(n)$.

Build a Min-heapify = $T(n) = T(\frac{n}{k}) + O(1)$

$\Rightarrow T(n) = O(n \log_k n)$.