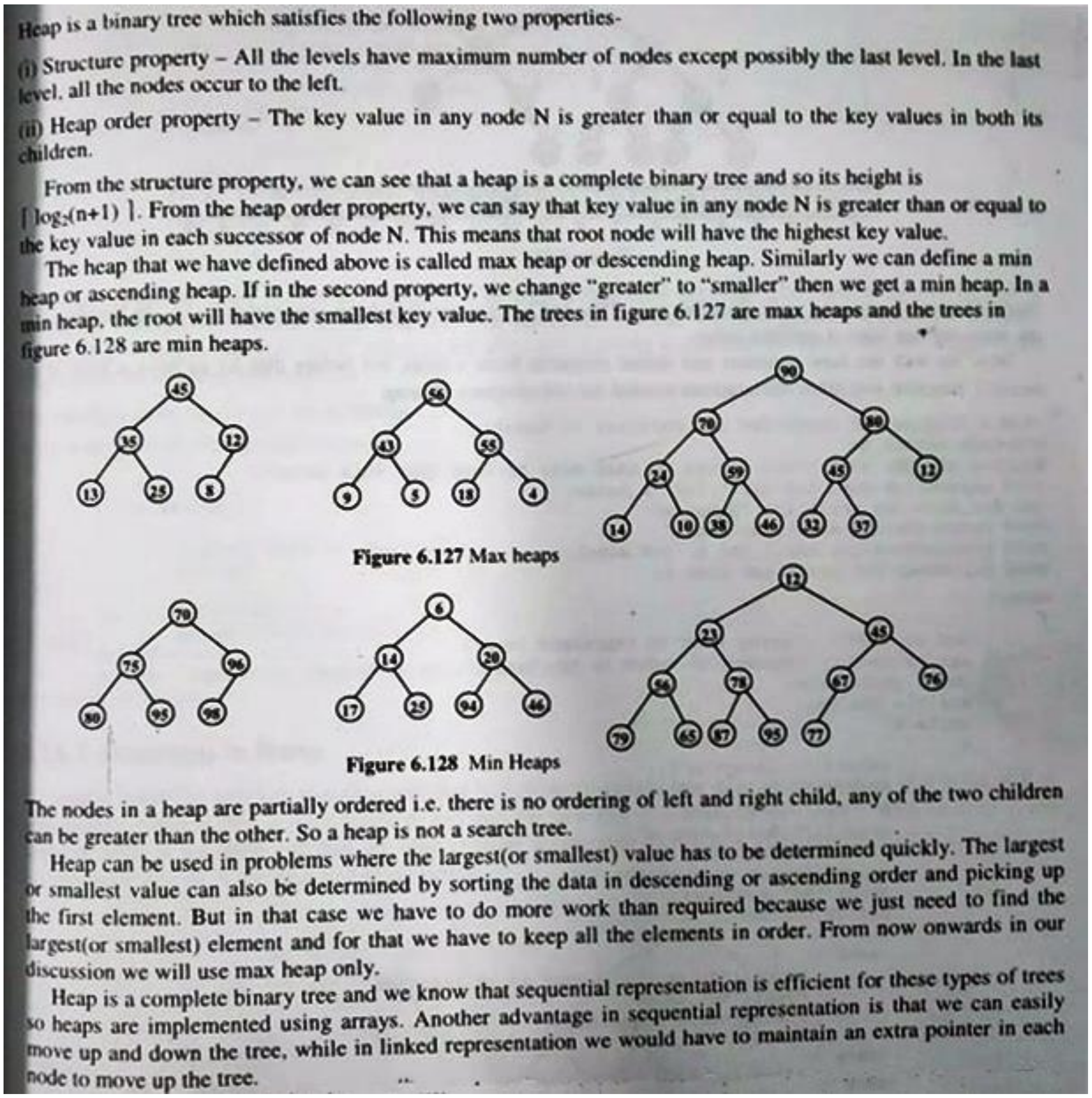
Heap



Heap is a binary tree which satisfies the following two properties-

(i) Structure property – All the levels have maximum number of nodes except possibly the last level. In the last level, all the nodes occur to the left.

(ii) Heap order property – The key value in any node N is greater than or equal to the key values in both its children.

From the structure property, we can see that a heap is a complete binary tree and so its height is $\lceil \log_2(n+1) \rceil$. From the heap order property, we can say that key value in any node N is greater than or equal to the key value in each successor of node N. This means that root node will have the highest key value.

The heap that we have defined above is called max heap or descending heap. Similarly we can define a min heap or ascending heap. If in the second property, we change "greater" to "smaller" then we get a min heap. In a min heap, the root will have the smallest key value. The trees in figure 6.127 are max heaps and the trees in figure 6.128 are min heaps.

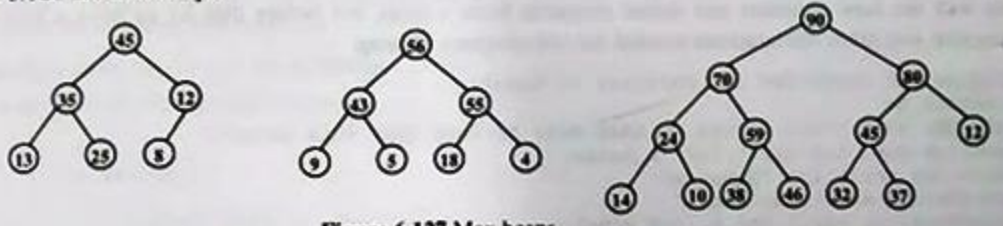


Figure 6.127 Max heaps

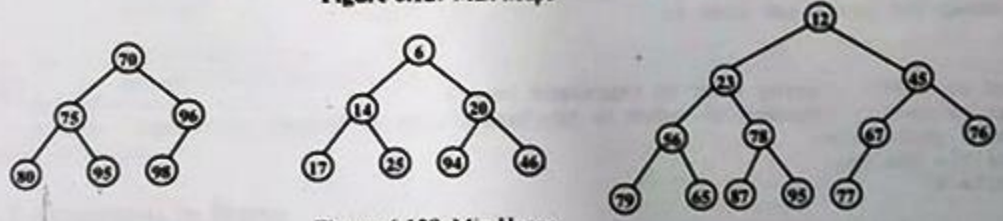


Figure 6.128 Min Heaps

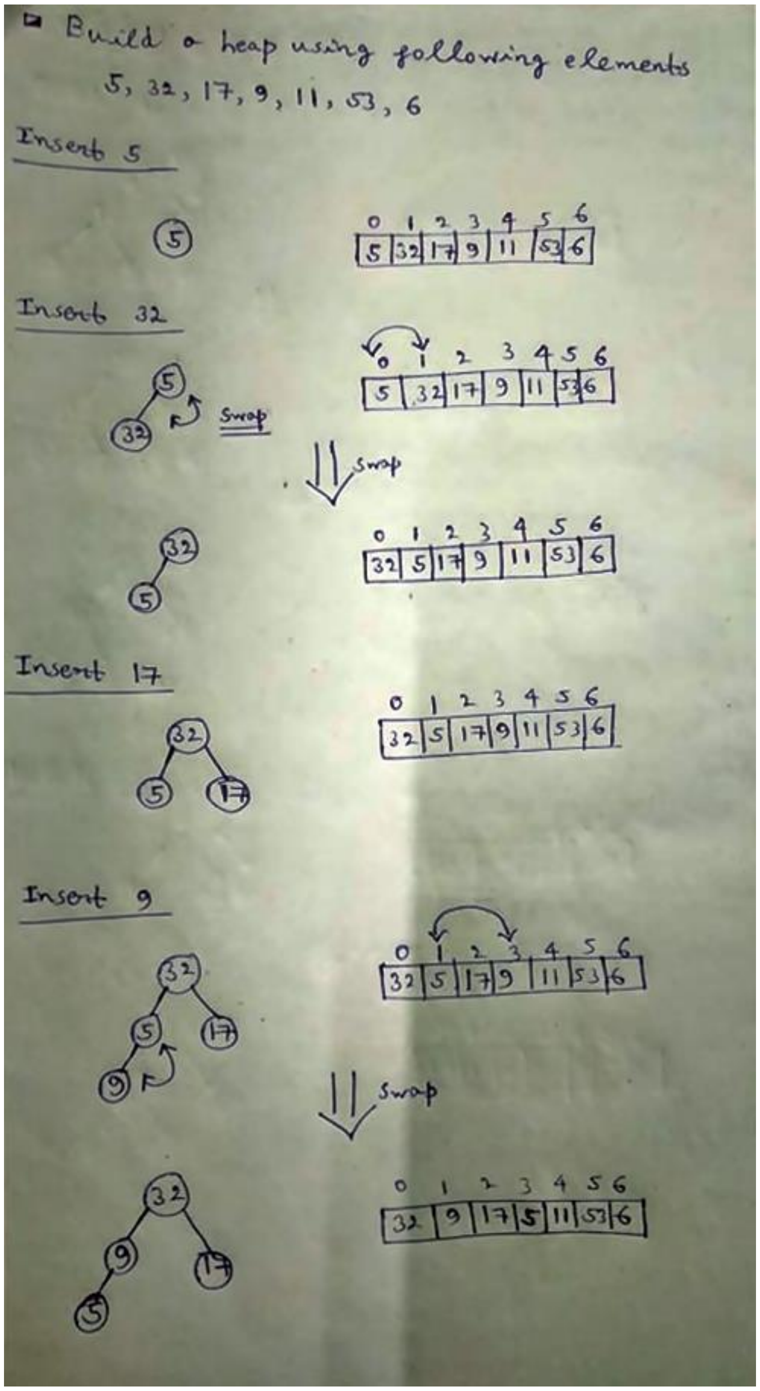
The nodes in a heap are partially ordered i.e. there is no ordering of left and right child, any of the two children can be greater than the other. So a heap is not a search tree.

Heap can be used in problems where the largest(or smallest) value has to be determined quickly. The largest or smallest value can also be determined by sorting the data in descending or ascending order and picking up the first element. But in that case we have to do more work than required because we just need to find the largest(or smallest) element and for that we have to keep all the elements in order. From now onwards in our discussion we will use max heap only.

Heap is a complete binary tree and we know that sequential representation is efficient for these types of trees so heaps are implemented using arrays. Another advantage in sequential representation is that we can easily move up and down the tree, while in linked representation we would have to maintain an extra pointer in each node to move up the tree.

Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata

Build a max-heap from input elements (Build_Heap or Heapify):



Build a heap using following elements
5, 32, 17, 9, 11, 53, 6

Insert 5

0	1	2	3	4	5	6
5	32	17	9	11	53	6

Insert 32

0	1	2	3	4	5	6
5	32	17	9	11	53	6

Swap

0	1	2	3	4	5	6
32	5	17	9	11	53	6

Insert 17

0	1	2	3	4	5	6
32	5	17	9	11	53	6

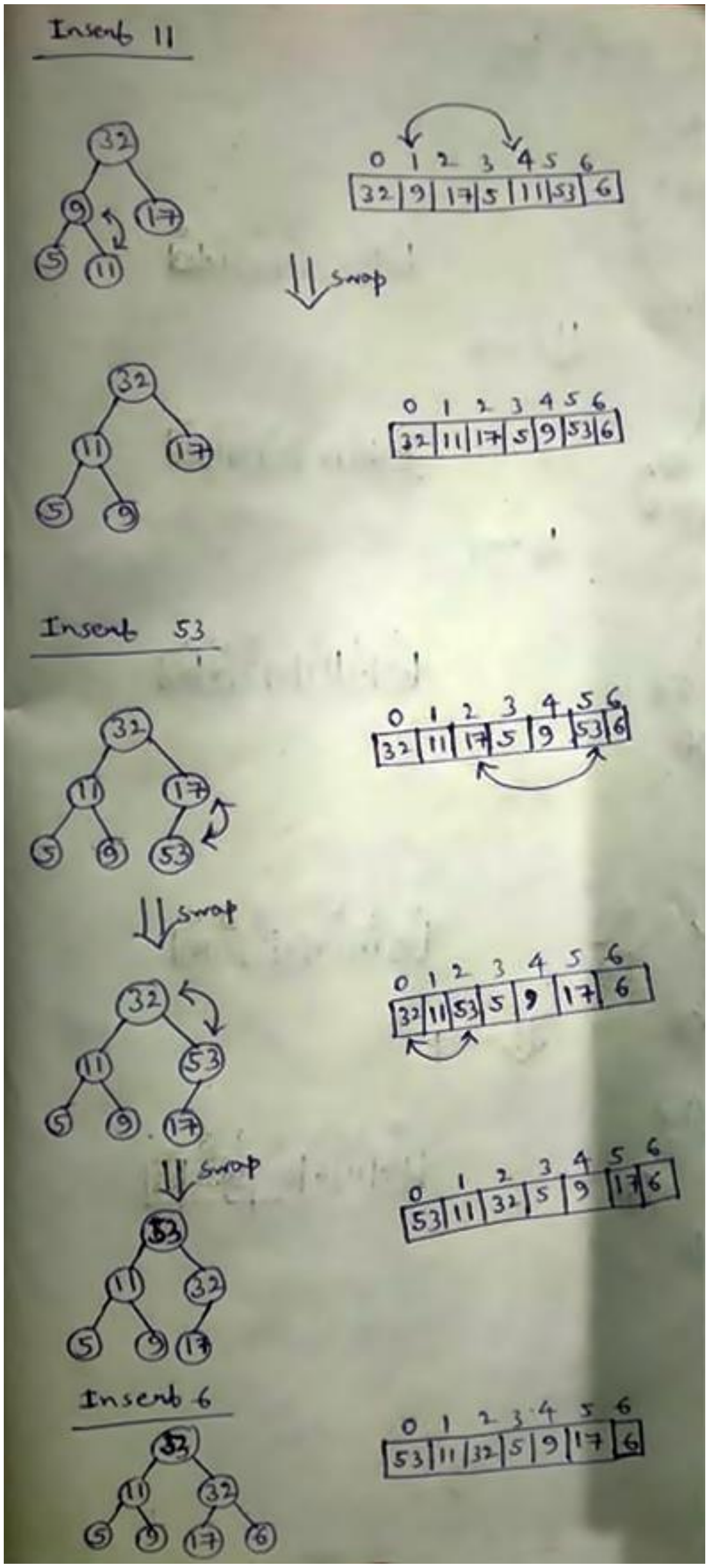
Insert 9

0	1	2	3	4	5	6
32	5	17	9	11	53	6

Swap

0	1	2	3	4	5	6
32	9	17	5	11	53	6

Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata



Insert 11

0	1	2	3	4	5	6
32	9	17	5	11	53	6

swap

0	1	2	3	4	5	6
32	11	17	5	9	53	6

Insert 53

0	1	2	3	4	5	6
32	11	17	5	9	53	6

swap

0	1	2	3	4	5	6
32	11	53	5	9	17	6

swap

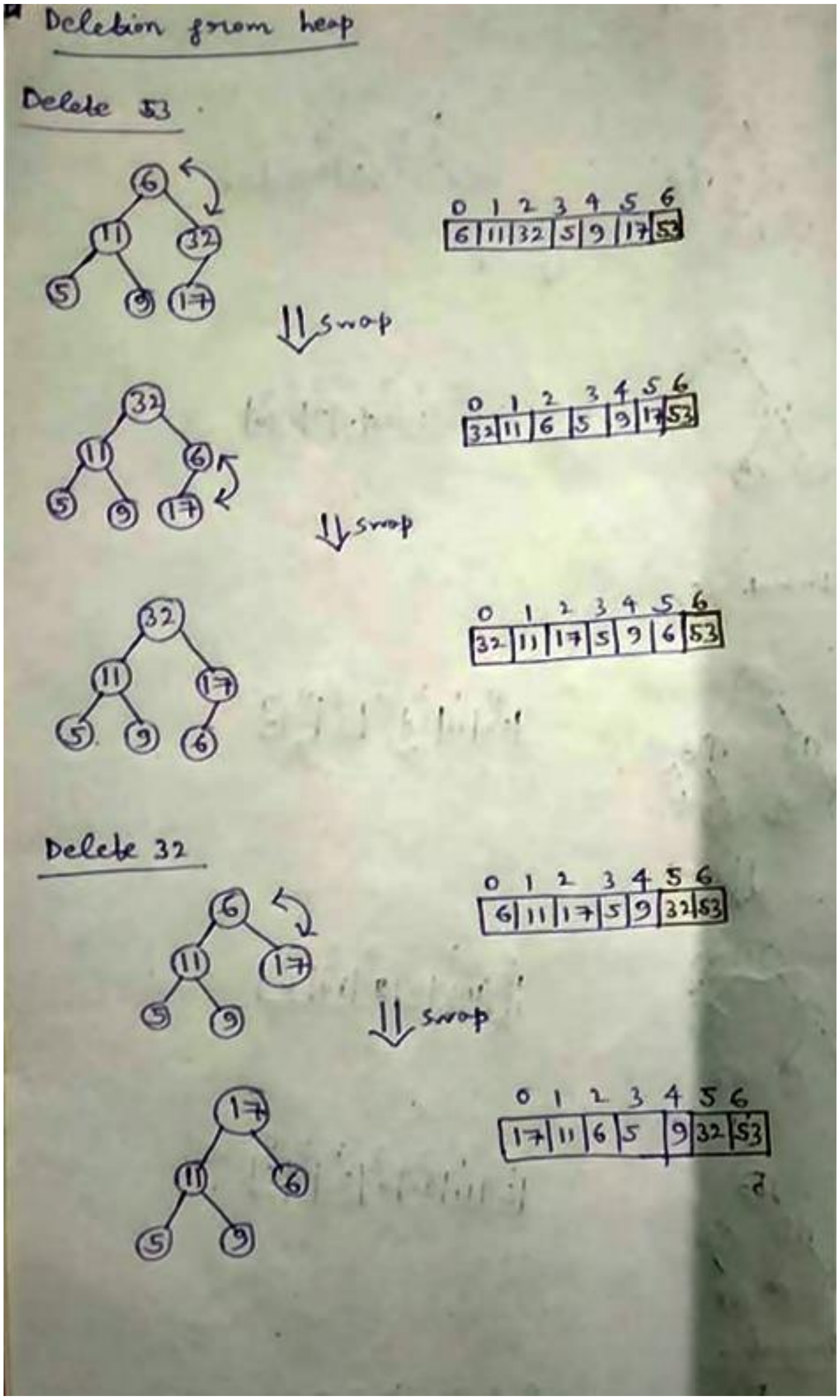
0	1	2	3	4	5	6
53	11	32	5	9	17	6

Insert 6

0	1	2	3	4	5	6
53	11	32	5	9	17	6

Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata

Deletion of elements from max-heap:



0	1	2	3	4	5	6
6	11	32	5	9	17	53

0	1	2	3	4	5	6
32	11	6	5	9	17	53

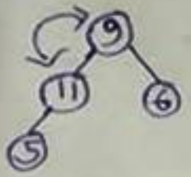
0	1	2	3	4	5	6
32	11	17	5	9	6	53

0	1	2	3	4	5	6
6	11	17	5	9	32	53

0	1	2	3	4	5	6
17	11	6	5	9	32	53

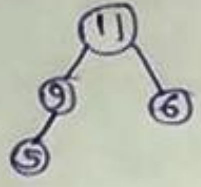
Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata

Delete 17



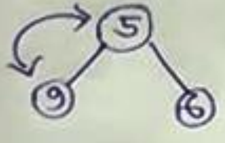
0	1	2	3	4	5	6
9	11	6	5	17	32	53

⇓ swap



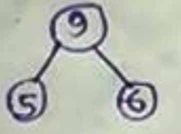
0	1	2	3	4	5	6
11	9	6	5	17	32	53

Delete 11



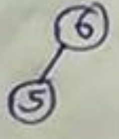
0	1	2	3	4	5	6
5	9	6	11	17	32	53

⇓ swap



0	1	2	3	4	5	6
9	5	6	11	17	32	53

Delete 9



0	1	2	3	4	5	6
6	5	9	11	17	32	53

Delete 6

0	1	2	3	4	5	6
5	6	9	11	17	32	53

Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata

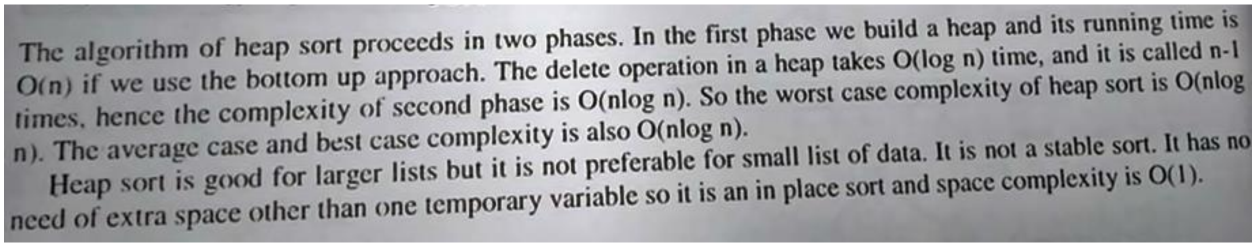
Heap Sort:

Heap sort works in two phases:

Phase 1: Build a max heap from the elements of the input array.

Phase 2: Keep on deleting the root till there is only one element in the tree.

Analysis:



The algorithm of heap sort proceeds in two phases. In the first phase we build a heap and its running time is O(n) if we use the bottom up approach. The delete operation in a heap takes O(log n) time, and it is called n-1 times, hence the complexity of second phase is O(nlog n). So the worst case complexity of heap sort is O(nlog n). The average case and best case complexity is also O(nlog n).

Heap sort is good for larger lists but it is not preferable for small list of data. It is not a stable sort. It has no need of extra space other than one temporary variable so it is an in place sort and space complexity is O(1).

Necessary Algorithms:

Procedure Heap_Sort()

```
Begin
    Step 1: Initialise loop variable i with lb.
    Step 2: Repeat step 3 while i≤ub.
    Step 3: Call procedure Build_Heap() with the array address, i, array element at i, lb, ub.
    [End of step 2 loop]
    Step 4: Store value of ub in i.
    Step 5: Repeat steps 6 - 7 while i≥lb.
    Step 6: Call procedure Del_heap() with array address, i, lb, ub.
    Step 7: Decrement i by 1
        i=i-1
    [End of step 5 loop]
End
```

Procedure Build_Heap()

```
Begin
    Step 1: Set i to lb+size, where size refers to the heap size.
    Step 2: Store the passed data to a[i].
    Step 3: [Heap building]
        Step 3.1: Call procedure parent() by passing value of i.
        Step 3.2: Store the result in p.
        Step 3.3: Repeat steps 4 - 5 while i>lb and a[p]<a[i].
```

Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata

```
        Step 4: Swap a[i] and a[p] using a temporary variable t.
        Step 5: Update i as p.
        Step 6: Go back to step 3.
        [End of step 3 loop]
End
```

Procedure Del_Heap()

```
Begin
        Step 1: [Swap first and last elements of heap]
                Swap a[lb] with a[size-1], using a temporary variable t.
        Step 2: Set i to lb.
        Step 3: Decrement size, due to deletion of one element
                size=size-1.
        Step 4: Repeat the steps 5 - 10.
        Step 5: [Check whether the root has a left child]
                Step 5.1: Call procedure() left by passing value of i and store the result in l.
                Step 5.2: If l ≥size, go to end.
        Step 6: [Check whether the root has a right child]
                Step 6.1: Call procedure right() by passing value of i and store the result in r.
                Step 6.2: If r ≥size
                         then
                                Step 6.2.1: Set max to l.
                                Step 6.2.2: Go to step 8.
        Step 7: Store the maximum of l and r to max.
        Step 8: If a[i]≥a[max], go to end.
        Step 9: Swap a[i] and a[max] using a temporary variable t.
        Step 10: Set i to max.
End
```

Procedure parent()

```
Begin
        Step 1: Compute ceiling value of i/2-1 and store in p.
        Step 2: Return p.
End
```

Procedure left()

```
Begin
        Step 1: Compute 2i+1.
        Step 2: Return the value.
End
```

Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata

Procedure right()

```
Begin
        Step 1: Compute 2i+2.
        Step 2: Return the value.
End
```

An important application of heaps-priority queue:

Priority Queue is an extension of queue data structure with the following properties:

1. Every item has a priority associated with it.
2. The element with highest priority is dequeued (deleted) before other elements with lower priorities.
3. If two elements have the same priority, they are served according to their order in the queue.

If a heap is used to implement priority queue (build a heap with the priority values), the node with the highest priority will always be at the root node. Hence delete operation always deletes node with the highest priority (dequeue operation is similar to deletion from heap in this case). A max heap or a min heap can be used to implement a max priority queue or a min priority queue.

If, priority(i) > priority(j) for all i>j then build max heap [Higher value means higher priority].

If, priority(i) > priority(j) for all i<j then build max heap [Lower value means lower priority].

i and j are integer values.

Reference books:

1. Data Structures Through C In Depth by S.K.Srivastava and Deepali Srivastava
2. Data Structures Using C by Reema Thareja

Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata