```java
 1 package heap;
 2 // have 3 important functions 1-> buildHeap(int arr[],int n) 2-> heapify(int arr[],int n, int i) 3-> printHeap(int arr[],int n).
 3 // one recursive call inside if block of heapify() function.
 4 //Caution!! In line 22 to 30 we are comparing the array elements(at right or left and largest) but inside if we are changing the index of larger variable
 5 // lastParent = (n/2) - 1; // Left node = (2*i) + 1;  // Right node = (2*i)+2 // Parent(root or sub-root) = (i-1)/2
 6 public class BinaryHeap {
 7 
 8     static void buildHeap(int arr[],int n) {
 9 
10         int lastParent = (n/2) - 1;    // 9th index in this case of our example
11 
12         for(int i = lastParent; i>= 0; --i) {
13             heapify(arr,n,i);  // i will be passed as a largest value to the function hepify
14         }
15     }
16 
17     static void heapify(int arr[],int n,int i) {
18         int largest = i;
19         int left = (2*i) + 1;  // left node of the binary tree
20         int right = (2*i) + 2;  // right node(index) of the binary tree
21 
22         // checks 2 things 1. left node should be less than length of array and 2. if element on the left is greater than that of assumed value at index largest
23         if(left < n && arr[left] > arr[largest]) {
24             largest = left; // if left is greater then assign left to largest variable
25         }
26 
27         // checks 2 things similarly 1. right should be less than that of length of the array. and 2. element on the right is greater than that of assumed value at index of largest
28         if(right < n && arr[right] > arr[largest]) {
29             largest = right;  // if right is greater then assign right to largest variable
30         }
31 
32         //above 2 conditions of if are checking for values of left and right node with it's parent node.
33 
34         //now check if index of root is changed or not. because of the above conditions
35         if(largest!=i) {  // if the index is changed then swap.   // !! swap heap condition
36             int temp = arr[i];
37             arr[i] = arr[largest];
38             arr[largest] = temp;
39 
40             //call heapify recursively to heapify the affected sub - tree
41             heapify(arr,n,largest);
42         }
43 
44         // be careful !! hepify recursive call will be inside swap if condition
45     }
46 
47     //for printing the tree  just as we print array elements
48     static void printHeap(int arr[],int n) {
49 
50         System.out.print("Array representation of heap is: ");
51         for(int i=0; i<n ; i++) {
```

```
            System.out.print(arr[i] + " ");
        }
    }


    public static void main(String[] args) {
    int arr[] = { 1,3,5,4,6,13,10,9,8,15,17 };
    int n = arr.length;  // n is the length of the array (n=11 in our case)

    buildHeap(arr,n);
    printHeap(arr,n);

    }

}

----------------------------- output ----------------------------

Array representation of heap is: 17 15 13 9 6 5 10 4 8 3 1

```