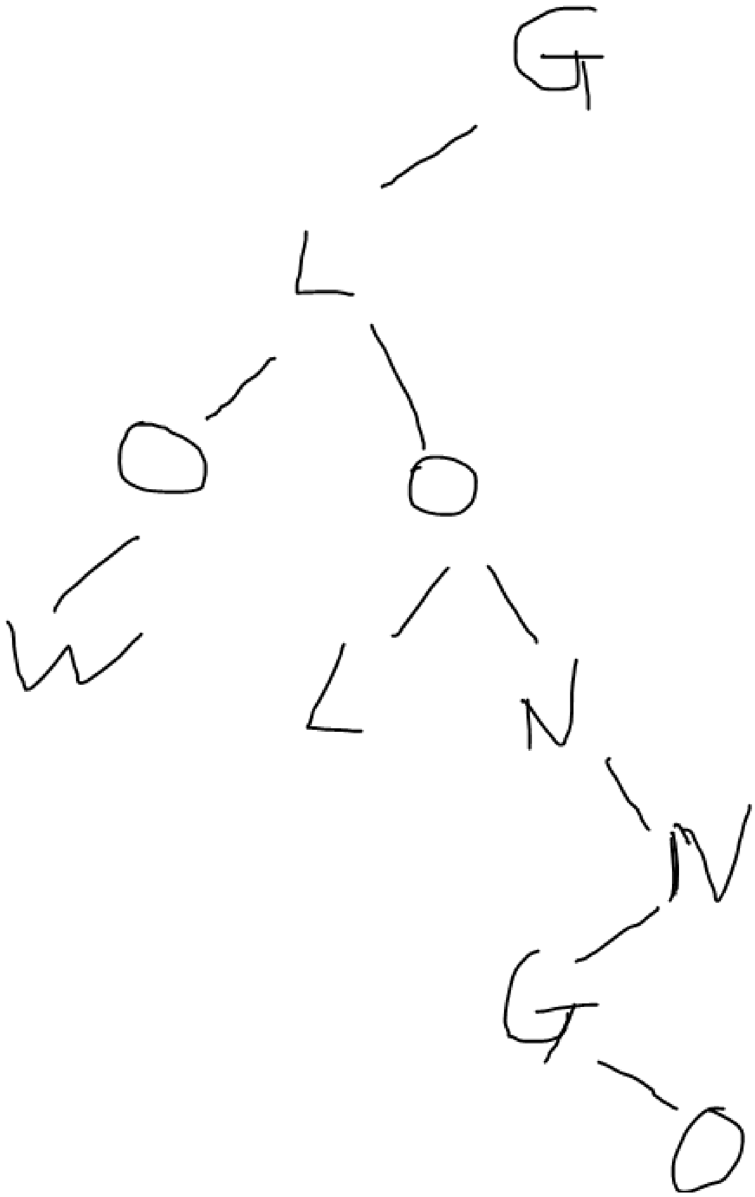
1)

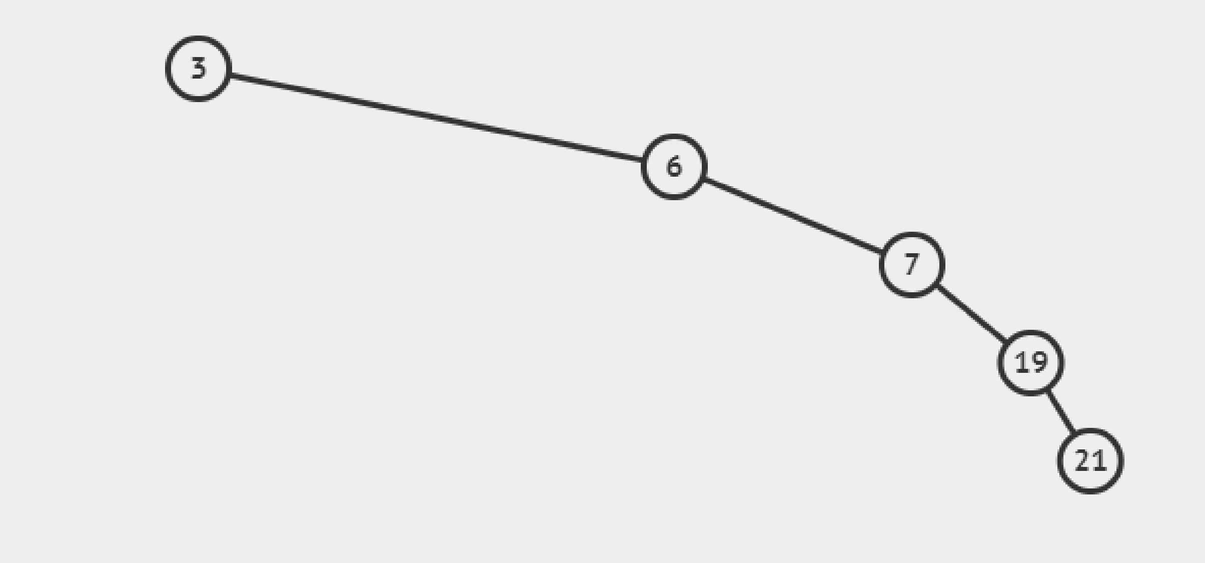
a)



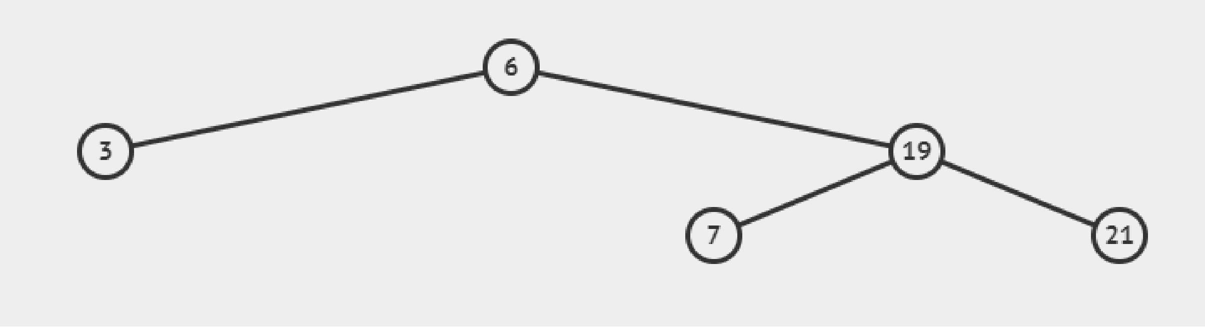
b)

(i)

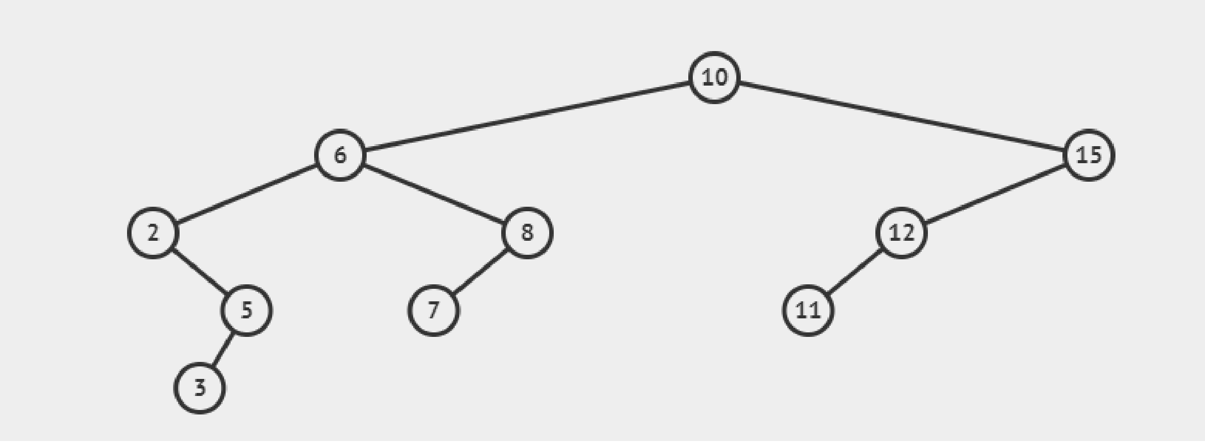
BST



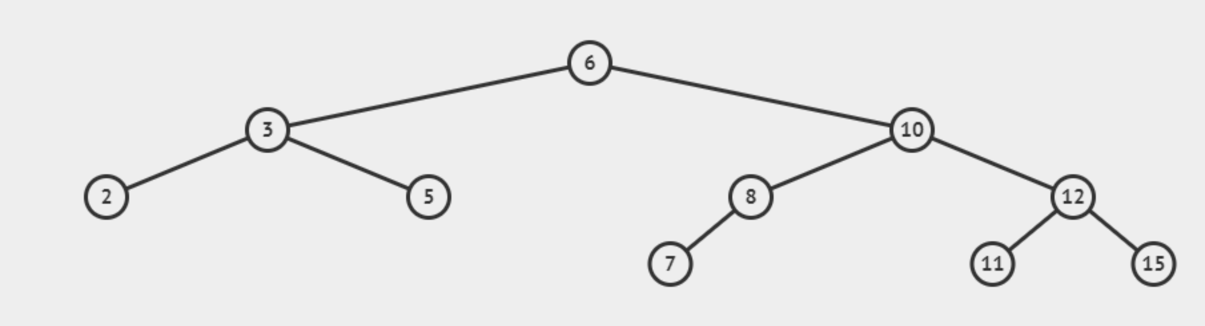
AVL



(ii)
BST



AVL



c)

The worst case is O(n). It is because in a worse case scenario, 1 child is attached to 1 root as it goes downwards, which means the run time of the binary search tree depends on the height, thus giving it a O(n) complexity.

2)

```
//return left child of `A[i]`
function Integer LEFT(Integer i){
   return 2*i + 1
}

//return right child of `A[i]`
function Integer RIGHT(Integer i){
   return 2*i + 2
}

//Recursive function to implement the heapify-down algorithm.
//The node at index `i` and its two direct children
//violates the heap property
function void heapify(Integer[] A, Integer i, Integer size){

   //get left and right child of node at index `i`
   Integer left = LEFT(i)
   Integer right = RIGHT(i)

   Integer smallest = i

   //compare `A[i]` with its left and right child
```

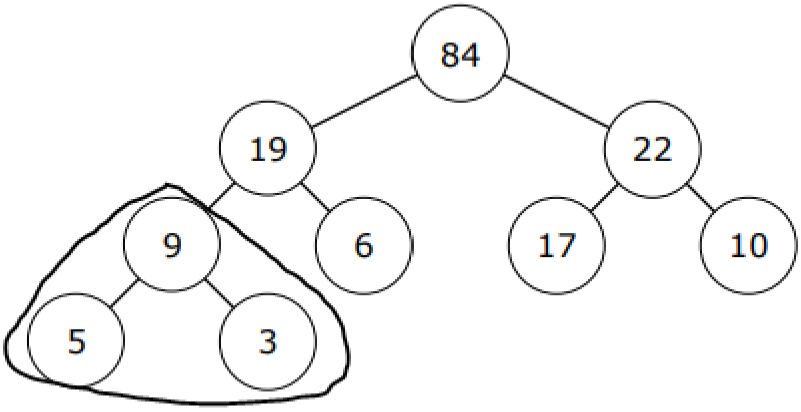
```
    //and find the smallest value
    if left < size && A[left] < A[i] {
        smallest = left
    }
    if right < size and A[right] < A[smallest]{
        smallest = right
    }
    //swap with a child having lesser value and
    //call heapify-down on the child
    if smallest != i {
        swap(A, i, smallest)
        heapify(A, smallest, size)
    }
}

//Utility function to swap two indices in a list
function swap(Integer[] A, Integer i, Integer j):

    Integer[] temp = A[i]
    A[i] = A[j]
    A[j] = temp



//Function to convert a max-heap into a min-heap
function void convert(Integer[] A){

    //Build-Heap: Call heapify starting from the last internal
    //node all the way up to the root node
    i = (A.length - 2) floor division by 2
    while i >= 0 {
        heapify(A, i, len(A))
        i = i - 1
    }

//Convert max-heap into min-heap in linear time
function main() {
    //a list representing the max-heap
    A = [9, 4, 7, 1, -2, 6, 5]

    //build a min-heap by initializing it by the given list
    convert(A)

    print(A)
}
```

b)

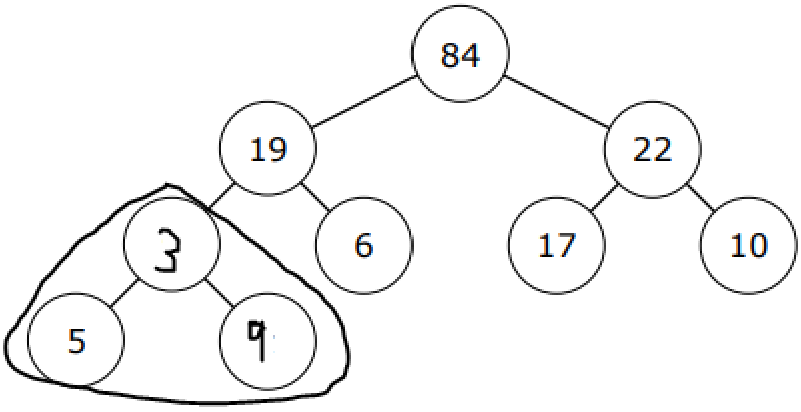
The given array in the question is as follows:

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

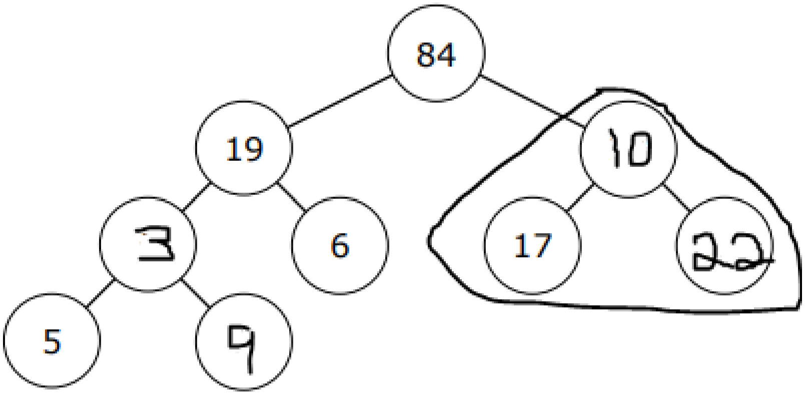
Before:



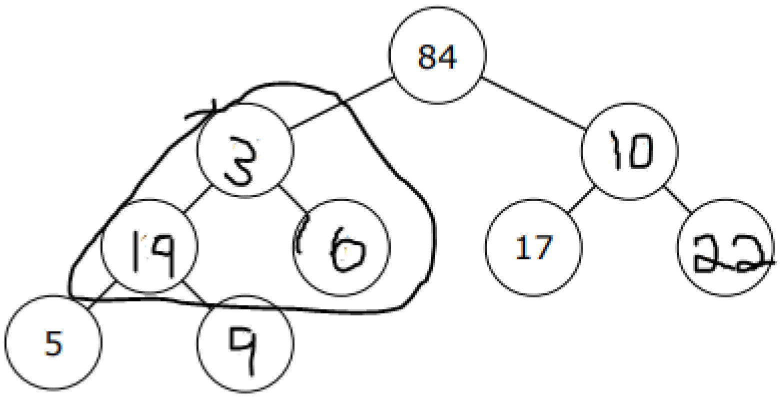
Swap node 3 with 8:



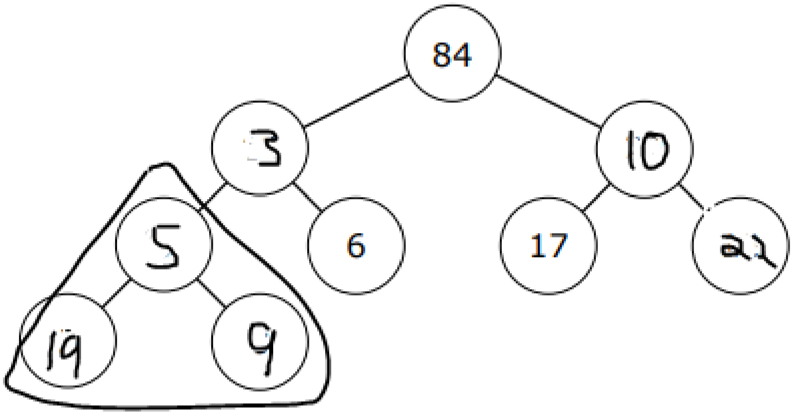
Swap node 2 with 6:



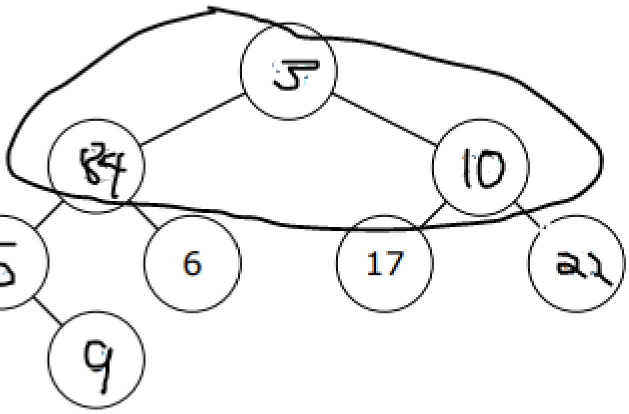
Swap node 1 with 3:



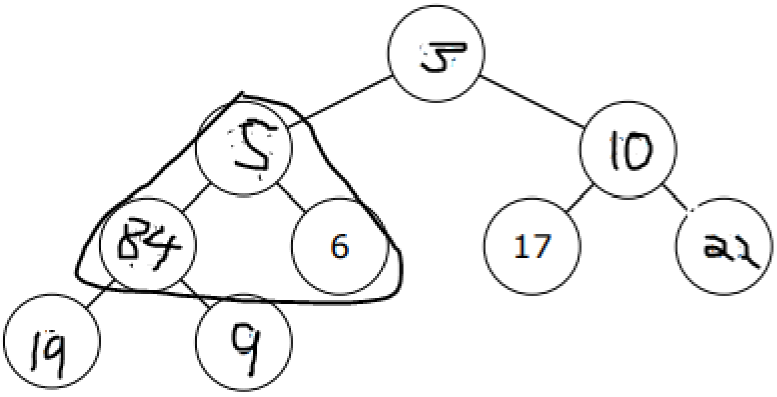
Swap node 3 with 7:



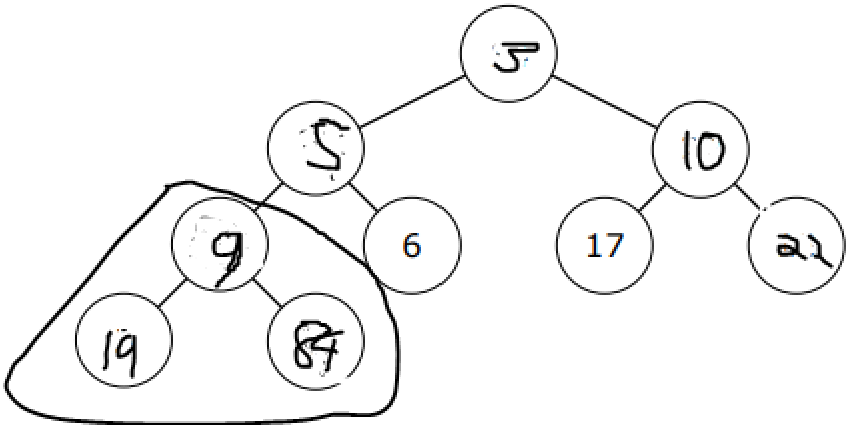
Swap node 0 with 1:



Swap node 1 with 3:



Swap node 3 with 8:



The tree is completed.