In-Lab – 1(page No:230)

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *left, *right;
};
struct node* insert(struct node* root, int value)
{
   struct node* newnode = (struct node*)malloc(sizeof(struct node));
   if (newnode == NULL)
      return root;

   newnode->data = value;
   newnode->left = newnode->right = NULL;
   if (root == NULL)
      return newnode;
   struct node *temp = root, *pre = NULL;
   while (temp != NULL)
   {
      pre = temp;
      if (value <= temp->data)
         temp = temp->left;
      else
         temp = temp->right;
   }
   if (value < pre->data)
      pre->left = newnode;
   else
      pre->right = newnode;

   return root;
}
int heightAndBalance(struct node *root, int *isBalanced)
{
   if (root == NULL)
      return 0;

   int leftHeight = heightAndBalance(root->left, isBalanced);
   int rightHeight = heightAndBalance(root->right, isBalanced);
```

```
    if (abs(leftHeight - rightHeight) > 1)
        *isBalanced = 0;

    return 1 + (leftHeight > rightHeight ? leftHeight : rightHeight);
}
int isBalanced(struct node *root)
{
    int isBalancedFlag = 1;
    heightAndBalance(root, &isBalancedFlag);
    return isBalancedFlag;
}
int main()
{
    int n,value;
    scanf("%d", &n);
    struct node* root = NULL;
    for (int i = 0; i < n; i++)
    {
         scanf("%d", &value);
         root = insert(root, value);
    }
    printf("%s\n", isBalanced(root) ? "Yes" : "No");
    return 0;
}
```

In-Lab – 2(page No: 232)

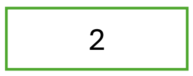
```
#include <stdio.h>
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i)
    {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i = n - 1; i > 0; i--)
    {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        heapify(arr, i, 0);
    }
}
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main()
{
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    heapSort(arr, n);

    printArray(arr, n);
    return 0;
}
```

In-Lab – 3(page No: 234)

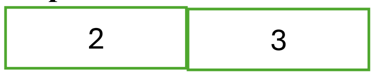
```
#include <stdio.h>
int arr[100];
int size = 0;

void swap(int *x, int *y)
{
   int temp = *x;
   *x = *y;
   *y = temp;
}

void insert(val)
{
   arr[size] = val;
   size++;
   int cur_index = size-1;
   while(cur_index!=0){
      int parent_index = (cur_index-1)/2;
      if(arr[parent_index]>arr[cur_index])
      {
         swap(&arr[parent_index], &arr[cur_index]);
      }
      cur_index = parent_index;
   }
}
void Heapify(int index)
{
    int left_child = 2*index+1;
    int right_child = 2*index+2;
    if(left_child<size && arr[left_child]<arr[index])
    {
       swap(&arr[index], &arr[left_child]);
       Heapify(left_child);
    }
    if(right_child<size && arr[right_child]<arr[index])
    {
       swap(&arr[index], &arr[right_child]);
       Heapify(right_child);
    }
}
void delete_from_heap(int index)
{
    printf("deleting the element %d\n", arr[index]);
    swap(&arr[index], &arr[size-1]);
    size--;
    Heapify(index);
}

void print()
{
   for(int i=0; i<size; i++)
   {
            printf("%d ", arr[i]);
         }
         printf("\n");
}
int main(void)
{
         int n;
         scanf("%d", &n);
         for (int i=0; i<n; i++)
         {
            int value;
            scanf("%d", &value);
            insert(value);
         }
         print();
         int x;
         scanf("%d", &x);
         for (int i=0; i<x; i++)
         {
            int index;
            scanf("%d", &index);
            delete_from_heap(index);
            print();
         }
}
```

Post-Lab – 1(page No: 236)

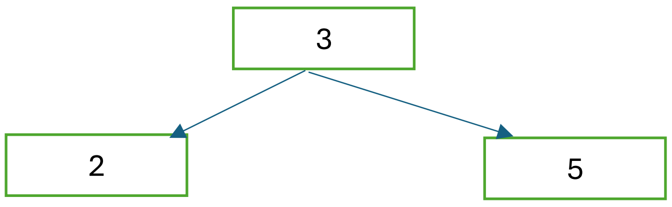
Step1:



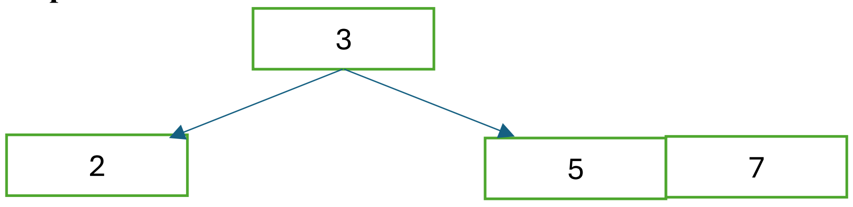
Step2:



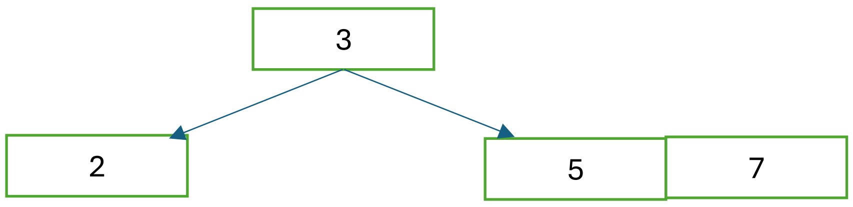
Step3:



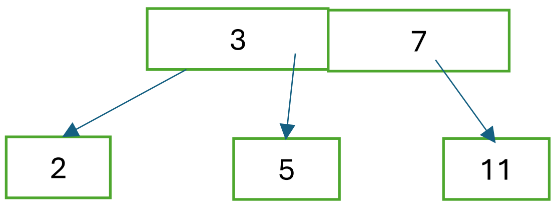
Step4:



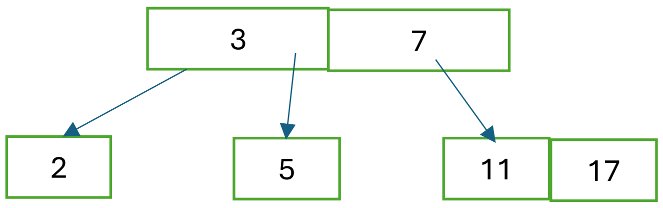
Step5:



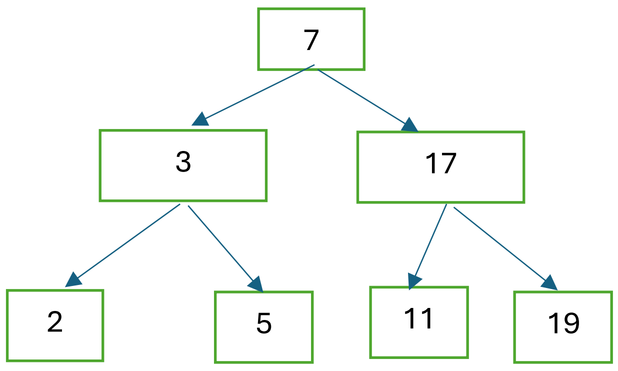
Step6:



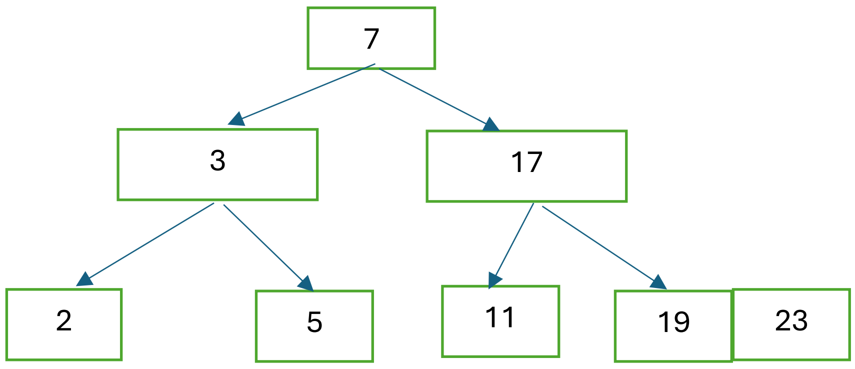
Step7:



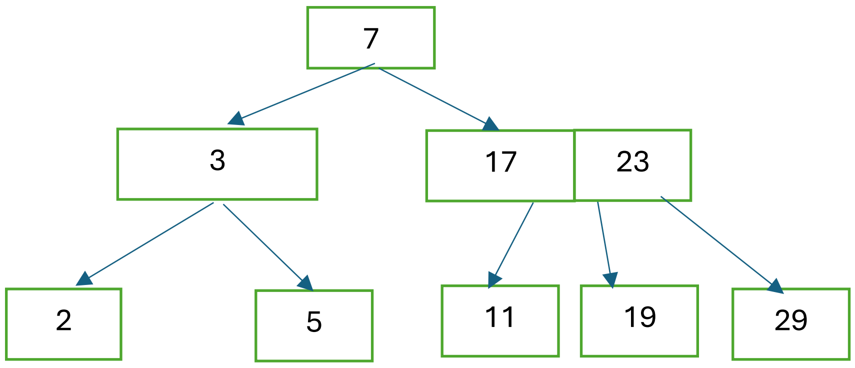
Step8:



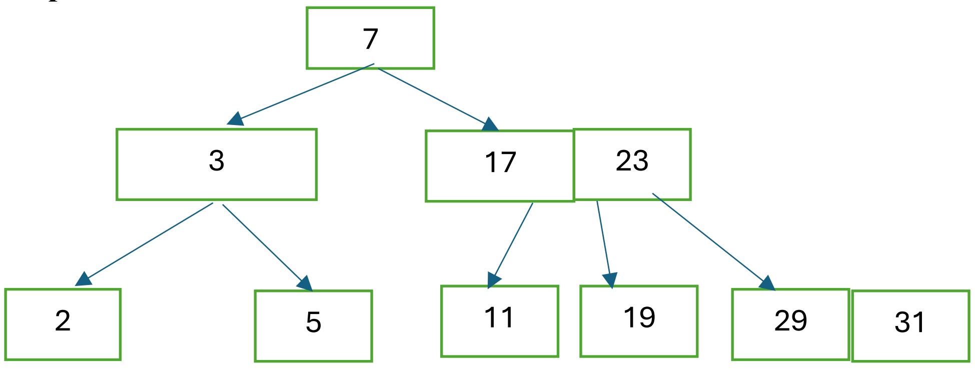
Step9:



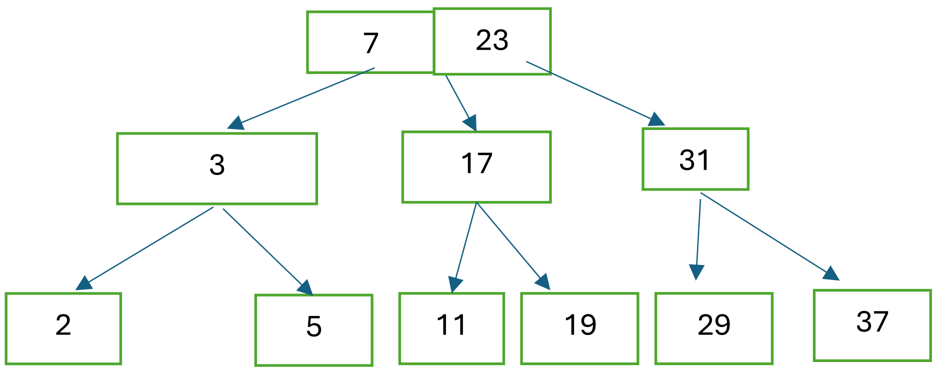
Step10:



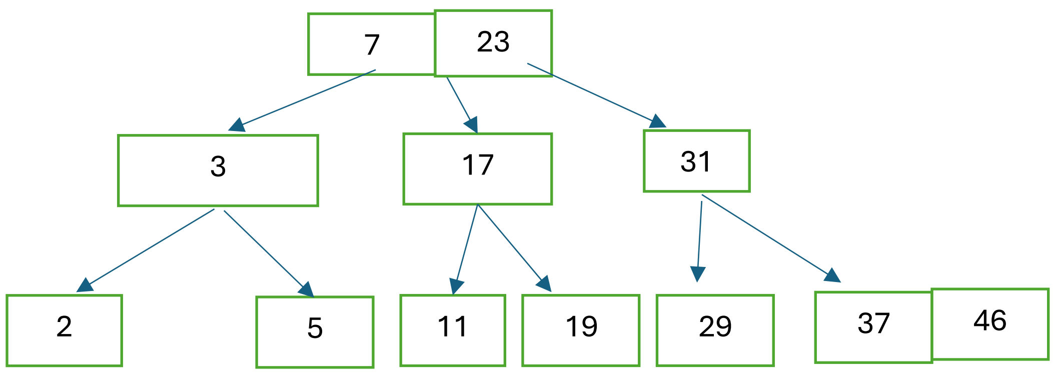
Step11:



Step12:



Step13:



Skill Lab-1(page No:238)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 1000
struct MaxHeap
{
   int size;
   int arr[MAX_SIZE];
};

void swap(int *a, int *b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}

void heapifyUp(struct MaxHeap *heap, int index)
{
   int parent = (index - 1) / 2;
   while (index > 0 && heap->arr[index] > heap->arr[parent])
   {
      swap(&heap->arr[index], &heap->arr[parent]);
      index = parent;
      parent = (index - 1) / 2;
   }
}
void insert(struct MaxHeap *heap, int value)
{
   heap->arr[heap->size] = value;
   heapifyUp(heap, heap->size);
   heap->size++;
```

```
}
void heapifyDown(struct MaxHeap *heap, int index)
{
    int left, right, largest;
    while (1)
    {
        left = 2 * index + 1;
        right = 2 * index + 2;
        largest = index;

        if (left < heap->size && heap->arr[left] > heap->arr[largest])
            largest = left;
        if (right < heap->size && heap->arr[right] > heap->arr[largest])
            largest = right;

        if (largest != index) {
            swap(&heap->arr[index], &heap->arr[largest]);
            index = largest;
        } else {
            break;
        }
    }
}
void deleteMax(struct MaxHeap *heap)
{
    if (heap->size == 0) return;
    heap->arr[0] = heap->arr[heap->size - 1];
    heap->size--;
    heapifyDown(heap, 0);
}
int getMax(struct MaxHeap *heap)
{
    if (heap->size == 0) return -1;
    return heap->arr[0];
}
int main()
{
    int N;
    scanf("%d", &N);
    struct MaxHeap heap;
    heap.size = 0;

    for (int i = 0; i < N; i++)
    {
        char op;
        int value;
        scanf(" %c", &op);
        if (op == '+')
        {
            scanf("%d", &value);
            insert(&heap, value);
        }
        else if (op == '-')
        {
            deleteMax(&heap);
        }
        printf("%d\n", getMax(&heap));
    }
    return 0;
}
```