```cpp
//In priority queue it takes O(logn) time for deleting/inserting elements.
//to make priority queue from an array it takes O(n)-> if buit using step down mtd or O(nlogn)-> if built using stl
#include<bits/stdc++.h>
using namespace std;
class Node{
    public:
    int data;
    Node* next;
    Node(int val){
        data = val;
        next = NULL;
    }
};

void merge(int arr[], int s,int mid, int e){
    int a1 = mid-s+1;
    int b1 = e-mid;
    int a[a1], b[b1];
    for(int i=0; i<a1; i++){
        a[i] = arr[i+s];
    }
    for(int i=0; i<b1; i++){
        b[i] = arr[mid+1+i];
    }
    int i=0, j=0 ,k=s;
    while(i<a1 && j<b1){
        if(a[i]<=b[j]){
            arr[k++] = a[i++];
        }else{
            arr[k++] = b[j++];
        }
    }
    while(i<a1){
        arr[k++] = a[i++];
    }
    while(j<b1){
        arr[k++] = b[j++];
    }
    return;
}

void mergeSort(int arr[], int s, int e){
    if(s>=e){
        return;
    }
    int mid = (s+e)/2;
    mergeSort(arr,s,mid);
    mergeSort(arr,mid+1,e);
    merge(arr,s,mid,e);
}
```

```
Node* mergeLL(Node* head1, Node* head2){
    if(head1==NULL and head2==NULL){
        return NULL;
    }
    Node *dummy = new Node(0);
    Node *tail = dummy;
    while(head1 and head2){
        if(head1->data > head2->data){
            tail->next = head2;
            head2 = head2->next;
        }
        else{
            tail->next = head1;
            head1 = head1->next;
        }
        tail = tail->next;
    }
    if(head1){
        tail->next = head1;
    }
    if(head2){
        tail->next = head2;
    }
    Node* head = dummy->next;
    delete dummy;
    return head;
}


Node* mergeKLL(Node* arr[],int k){
    Node* temp = arr[0];
    for(int i=1; i<k; i++){
        temp = mergeLL(temp,arr[i]);
    }
    return temp;
}


Node* mergeKLL1(Node* arr[], int k){
    priority_queue<int, vector<int> , greater<int>> pq;
    for(int i=0; i<k; i++){
        Node* temp = arr[i];
        while(temp){
            pq.push(temp->data);
            temp = temp->next;
        }
    }
    Node* dummy = new Node(0);
    Node* temp = dummy;
    while(!pq.empty()){
        temp->next = new Node(pq.top());
        pq.pop();
        temp = temp->next;
    }
```

```
    Node* head = dummy->next;
    delete dummy;
    return head;
}

int main(){
    // //find sum of elements between k1^th and k2^th smallest element
    int arr[] = {7,10,4,3,20,15,8,5};
    int k1 = 3, k2 = 6, n = 8;
    priority_queue<int> pq1; //Max-Heap
    priority_queue<int> pq2; //Max-Heap
    for(int i=0; i<k1; i++){
        pq1.push(arr[i]);
    }
    for(int i=0; i<k2; i++){
        pq2.push(arr[i]);
    }

    for(int i=k1; i<n; i++){
        if(pq1.top()>arr[i]){
            pq1.pop();
            pq1.push(arr[i]);
        }
    }

    for(int i=k2; i<n; i++){
        if(pq2.top()>arr[i]){
            pq2.pop();
            pq2.push(arr[i]);
        }
    }
    int sum1 = 0, sum2 = 0;
    while(!pq1.empty()){
        sum1 += pq1.top();
        pq1.pop();
    }
    pq2.pop(); //to remove 1st ele from the second pq
    while(!pq2.empty()){
        sum2 += pq2.top();
        pq2.pop();
    }
    cout<<"Sum is: "<<sum2-sum1<<endl;


    //kth smallest element in a stream
    int arr[] = {1,2,3,4,5,6,7};
    int k = 4 , size = 7;
    vector<int> ans;
    priority_queue<int,vector<int>,greater<int>> pq; //Min-Heap
    for(int i=0; i<k-1; i++){
        ans.push_back(-1);
        pq.push(arr[i]);
    }
    for(int i=k-1; i<size; i++){
```

```
        pq.push(arr[i]);
        ans.push_back(pq.top());
        pq.pop();
    }
    for(auto ele: ans){
        cout<<ele<<" ";
    }


    //merge sort
    int arr[] = {1,5,2,7,100,56,4,7};
    int n = sizeof(arr) / sizeof(arr[0]);
    mergeSort(arr,0,n-1);
    for(int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    }


    //merge 2 sorted linkedlist
    Node* head1 = new Node(1);
    head1->next = new Node(10);
    head1->next->next = new Node(11);
    head1->next->next->next = new Node(15);
    head1->next->next->next->next = new Node(20);

    Node* head2 = new Node(2);
    head2->next = new Node(3);
    head2->next->next = new Node(4);
    head2->next->next->next = new Node(5);
    head2->next->next->next->next = new Node(6);

    Node* temp = mergeLL(head1,head2);
    while(temp){
        cout<<temp->data<<"->";
        temp = temp->next;
    }
    cout<<"NULL"<<endl;


    //merge k sorted linked list
    //Method-1
    Node* ans = mergeKLL(Node* arr[], int size);

    //Method-2 : Using Heap(Priority Queue) O(nklognk)
    Node* ans = mergeKLL1(Node* arr[], int start, int end);

    //Method-3 : using merge Sort
    Node* ans = mergeKLL2(Node* arr[], int size)

    //Method-4 : Using Heap(Priority Queue) O(nklogk)
    Node* ans = mergeKLL3(Node* arr[], int start, int end);

    return 0;
}
```