Largest Subarray with 0 Sum:

```
class Solution{
   public:
   int maxLen(vector<int>&A, int n)
   {
      unordered_map<int,int>mp;
      int sum=0;
      mp[sum]=-1;
      int maxLength=0;
      int i=0;
      while(i<n){
         sum+=A[i];
         if(mp.find(sum)==mp.end()){
            mp[sum]=i;
         }
         else{
            int prevInd=mp[sum];
            maxLength=max(maxLength,i-prevInd);
            // if(maxLength<(i-prevInd)){
            //     maxlength=(i-prevInd)
            // }
         }
         i++;
      }
      return maxLength;
   }
};
```

Time:O(n)
Space:O(n)

Kth Smallest Element:

```
class Solution{
    public:
       int kthSmallest(int arr[], int l, int r, int k) {
       priority_queue<int>pq;
       for(int i=l;i<=r;i++){
           pq.push(arr[i]);
           if(pq.size()>k){
               pq.pop();
           }
       }
       return pq.top();
    }
};
```

Time:O(nlogk)
Space:O(k)

Top K Frequent Elements:

```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        priority_queue< vector<pair<int,int>>,greater<pair<int,int>>>pq;
        unordered_map<int,int>mp;
        for(int i=0;i<nums.size();i++){
            mp[nums[i]]++;
        }

        for(auto it:mp){
            int element=it.first;
            int frequency=it.second;
            pq.push({frequency,element});
            if(pq.size()>k){
                pq.pop();
            }
        }
        vector<int>ans;
        while(!pq.empty()){
        push_back(pq.top().second);
            pq.pop();
        }
        return ans;
    }
};
```

Time:O(nlogk)
Space:O(k)

Find K Closest Elements:

```
class Solution {
public:
    vector<int> findClosestElements(vector<int>& arr, int k, int x) {
        priority_queue<pair<int,int>>pq;

        for(int i=0;i<arr.size();i++){
            int diff=abs(arr[i]-x);
            pq.push({diff,arr[i]});
```

```
            if(pq.size()>k){
                pq.pop();
            }
        }
        vector<int>ans;
        while(!pq.empty()){
            int d=pq.top().first;
            int ele=pq.top().second;
            pq.pop();
            ans.push_back(ele);
        }
        sort(ans.begin(),ans.end());
        return ans;
    }
};
```

Time:O(nlogk)
Space:O(k)

Min Cost To connect ropes:

```
class Solution
{
    public:
    //Function to return the minimum cost of connecting the ropes.
    long long minCost(long long arr[], long long n) {
         long long cost=0;
        priority_queue<long long,vector<long long>,greater<long
long>>pq;
        for(int i=0;i<n;i++){
            pq.push(arr[i]);
        }
        while(pq.size()>1){
            long long ele1=pq.top();
            pq.pop();

            long long ele2=pq.top();
            pq.pop();

            cost+=ele1+ele2;
            pq.push(ele1+ele2);
        }
        return cost;
    }
};
```

Time:O(nlogn)
Space:O(n)