EASY :

1. Reverse K elements of a queue.

```
void solve(queue<int> &q,int i,int n)
{
   if(i==n)
   return;

   int val=q.front();
   q.pop();
   i++;
   solve(q,i,n);
   q.push(val);
}
queue<int> modifyQueue(queue<int> q, int k)
{
   solve(q,0,k);
   int n=q.size()-k;
   while(n--)
   {
      int val=q.front();
      q.pop();
      q.push(val);
   }
   return q;
   // add code here.
}
```

2. First Circular Tour that visits all petrol pumps

```
class Solution{
  public:

   int tour(petrolPump p[],int n)
   {
     int currsum=0;
     int totalsum=0;
     int pos=-1;
     for(int i=0;i<n;i++)
     {
        currsum=currsum+p[i].petrol-p[i].distance;
        totalsum=totalsum+p[i].petrol-p[i].distance;
        if(currsum<0)
        {
```

```
            currsum=0;
            pos=i;
        }
    }
    if(totalsum<0)
    return -1;
    else
    return (pos+1)%n;
  }
};
```

MEDIUM :

1. LRU Cache

```
class LRUCache
{
  private:
  public:
  unordered_map<int,int>mp;
  unordered_map<int,int>priority;
  queue<int>q;
  int length;
  void add_queue(int key)
  {
    q.push(key);
    priority[key]++;
  }
  LRUCache(int cap)
  {
    length=cap;
  }
  int get(int key)
  {
    auto it=mp.find(key);
    if(it==mp.end())
    return -1;
    add_queue(key);
    return it->second;
  }

  void set(int key, int value)
  {
    if(mp.size()<length)
    {
```

```
            mp[key]=value;
            add_queue(key);
            return;
        }
        auto it=mp.find(key);
        if(it!=mp.end())
        {
            it->second=value;
            add_queue(key);
            return;
        }
        while(true)
        {
            int cur=q.front();
            q.pop();
            priority[cur]--;
            if(priority[cur]==0)
            {
                mp.erase(cur);
                break;
            }
        }
        mp[key]=value;
        add_queue(key);
    }
};
```

2. Rotten Oranges

```
class Solution {
public:

    int orangesRotting(vector<vector<int>>& grid)
    {

        int p=0;
        int count=0;
        int time=0;
        int n=grid.size();
        int m=grid[0].size();
        queue<pair<int,int>>q;

        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
```

```
            if(grid[i][j]==2)
            q.push({i,j});

            if(grid[i][j]!=0)
            p++;
        }
    }

    int arr_x[4]={0,0,1,-1};
    int arr_y[4]={1,-1,0,0};
    while(!q.empty())
    {
        int s=q.size();
        count+=s;
        while(s--)
        {
            int x=q.front().first;
            int y=q.front().second;
            q.pop();
            for(int i=0;i<4;i++)
            {
                int next_x=x+arr_x[i];
                int next_y=y+arr_y[i];
                if(next_x<0 || next_y<0 || next_x>=n || next_y>=m || grid[next_x][next_y]!=1)
                continue;
                grid[next_x][next_y]=2;
                q.push({next_x,next_y});
            }
        }
        if(!q.empty())
        time+=1;
    }
    if(count==p)
    return time;
    return -1;
  }
};
```

HARD :

1. Trapping Rain Water

```
class Solution {
public:
  int trap(vector<int>& height)
```

```
    {
        int n=height.size();

        vector<int>pre(n);
        vector<int>post(n);

        pre[0]=height[0];
        for(int i=1;i<n;i++)
        pre[i]=max(pre[i-1],height[i]);

        post[n-1]=height[n-1];
        for(int i=n-2;i>=0;i--)
        post[i]=max(post[i+1],height[i]);

        long long int sum=0;
        for(int i=0;i<n;i++)
        sum+=(min(pre[i],post[i])-height[i]);

        return sum;
    }
};
```

2. Sliding window maximum

```
class Solution {
public:
    priority_queue<pair<int,int>, vector<pair<int,int>>,greater<pair<int,int>> > st;
    int b=0;
    int maxx=INT_MIN;
    vector<int>ans;
    vector<int> maxSlidingWindow(vector<int>& nums, int k)
    {
        int n=nums.size();

       for(int i=0;i<min(k,n);i++)
       {
          if(maxx<nums[i])
          {
             maxx=nums[i];
             b=i;
          }
       }
       st.push({maxx,b});
       ans.push_back(maxx);
       for(int i=k;i<n;i++)
       {
```

```
        if(nums[i]>st.top().first)
        {
            while(!st.empty())
            {
                st.pop();
            }
            st.push({nums[i],i});
            ans.push_back(nums[i]);

        }

        else if(i-st.top().second<k)
        {
            ans.push_back(st.top().first);


        }
        else
        {
            int maxx=INT_MIN;
            while(!st.empty())
            {
                st.pop();
            }
            for(int j=i-k+1;j<=i;j++)
            {
                st.push({nums[j],j});
                if(maxx<nums[j])
                {
                    maxx=nums[j];
                    b=j;
                }

            }
            ans.push_back(maxx);
            st.push({maxx,b});
        }

    }
     return ans;

  }
};
```