Stacks

Questions:

1. Nearest greater to left
2. Nearest greater to right (Next largest element)
3. Nearest smaller to left (Nearest Smaller element)
4. Nearest smaller to right
5. Stock span problem
6. Maximum Area of rectangle in Histogram
7. Maximum Area of Rectangle in Binary Matrix
8. Rain water trapping
9. Minimum element in the stack with extra space
10. Minimum element in the stack with O(1) space
11. The celebrity problem
12. Longest Valid parenthesis
13. Iterative TOH

Tricky Questions to above part: https://leetcode.com/problems/next-greater-element-ii/

https://leetcode.com/problems/next-greater-element-iii/

Conceptual

1. Rain water trapping
2. Implementing a minimum stack with Extra space and without extra space
3. Implementing a stack using Heap
4. The celebrity Problem
5. Longest valid Parathesis
6. Iterative TOH

Identification

1) Stack problems usually have array
2) If brute force is O(n2) and j loop is dependent on i then this version will have improvised sol in stack

Concept:

Next Largest Element aka nearest greater to right

Nearest Greater to Right

// https://www.youtube.com/watch?v=NXOOYYwpbg4&list=PL_z_8CaSLPWdeOezg68SKkeLN4-
//T_jNHd&index=2

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
vector<int> nextgreater(vector<int> arr){
   vector<int> ans;
   stack<int> st;
   int n=arr.size();
   for(int i=n-1;i>=0;i--){
      if(st.size()==0){
         ans.push_back(-1);
      }
      else if(st.size()>0 and st.top()>arr[i]){
         ans.push_back(st.top());
      }
      else if(st.size()>0 and st.top()<=arr[i]){
         //pop elements
         while(st.size()>0 and st.top()<=arr[i]){
            st.pop();
         }
         if(st.size()==0){
            ans.push_back(-1);
         }
         else if(st.top()>arr[i]){
            ans.push_back(st.top());
         }

      }
      st.push(arr[i]);
   }
   reverse(ans.begin(),ans.end());
   return ans;
}

int main() {
  vector<int> arr={2,4,4,7};
  vector<int> ans=nextgreater(arr);
  for(int i=0;i<4;i++){
     cout<<ans[i]<<endl;
  }
   return 0;
}
```

Nearest Greater to Left

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
vector<int> nextgreater(vector<int> arr){
    vector<int> ans;
    stack<int> st;
    int n=arr.size();
    for(int i=0;i<n;i++){
        if(st.size()==0){
            ans.push_back(-1);
        }
        else if(st.size()>0 and st.top()>arr[i]){
            ans.push_back(st.top());
        }
        else if(st.size()>0 and st.top()<=arr[i]){
            //pop elements
            while(st.size()>0 and st.top()<=arr[i]){
                st.pop();
            }
            if(st.size()==0){
                ans.push_back(-1);
            }
            else if(st.top()>arr[i]){
                ans.push_back(st.top());
            }

        }
        st.push(arr[i]);
    }

    return ans;
}

int main() {
   vector<int> arr={2,4,4,7};
   vector<int> ans=nextgreater(arr);
   for(int i=0;i<4;i++){
      cout<<ans[i]<<endl;
   }
    return 0;
}
```

Nearest Smaller to Left

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
vector<int> nextsmaller(vector<int> arr){
   vector<int> ans;
   stack<int> st;
   int n=arr.size();
   for(int i=0;i<n;i++){
//Three conditions, if stack is empty
      if(st.size()==0){
         ans.push_back(-1);
      }
//If element in stack is less than the element in question
      else if(st.size()>0 and st.top()<arr[i]){
         ans.push_back(st.top());
      }
//If the stack top is greater than the element then simply pop
      else if(st.size()>0 and st.top()>=arr[i]){
         //pop elements
         while(st.size()>0 and st.top()>=arr[i]){
            st.pop();
         }
//Semi-condition after popping the elements if the stack size is 0 then simply push -1
         if(st.size()==0){
            ans.push_back(-1);
         }
//Else remove push top element in the array
         else if(st.top()<arr[i]){
            ans.push_back(st.top());
         }

      }
      st.push(arr[i]);
   }
//No need to reverse the array
   return ans;
}

int main() {
  vector<int> arr={2,4,4,7};
  vector<int> ans= nextsmaller(arr);
  for(int i=0;i<4;i++){
     cout<<ans[i]<<endl;
  }
   return 0;
}
```

Nearest Smaller to Right

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
vector<int> nextsmaller(vector<int> arr){
    vector<int> ans;
    stack<int> st;
    int n=arr.size();
    for(int i=n-1;i>=0;i--){
//Three conditions, if stack is empty
        if(st.size()==0){
            ans.push_back(-1);
        }
//If element in stack is less than the element in question
        else if(st.size()>0 and st.top()<arr[i]){
            ans.push_back(st.top());
        }
//If the stack top is greater than the element then simply pop
        else if(st.size()>0 and st.top()>=arr[i]){
            //pop elements
            while(st.size()>0 and st.top()>=arr[i]){
                st.pop();
            }
//Semi-condition after popping the elements if the stack size is 0 then simply push -1
            if(st.size()==0){
                ans.push_back(-1);
            }
//Else remove push top element in the array
            else if(st.top()<arr[i]){
                ans.push_back(st.top());
            }

        }
        st.push(arr[i]);
    }
//Need to reverse the array
reverse(ans.begin(),ans.end());
    return ans;
}

int main() {
   vector<int> arr={2,4,4,7};
   vector<int> ans= nextsmaller(arr);
   for(int i=0;i<4;i++){
       cout<<ans[i]<<endl;
   }
    return 0;
}
```

Now below four questions are related to above concept

Stock span

```
class
Solution
            {
                public:
                //Function to calculate the span of stockâ€™s price for all n days.
                //https://www.youtube.com/watch?v=p9T-
            fE1g1pU&list=PL_z_8CaSLPWdeOezg68SKkeLN4-T_jNHd&index=6
                vector <int> calculateSpan(int price[], int n)
                {
                   // This is variation of largest greater to left
                   //rather than taking pair of the number and index in the stack i am
            just pushing the index
                   vector<int> ans;
                   stack<int> st;
                   for(int i=0;i<n;i++){
                       if(st.empty()){
                           ans.push_back(-1); //if no element is present then insert 1
                       }
                       if(prices[st.top()]>prices[i]){
                          ans.push_back(st.top());
                       }
                       else{
                           while(!st.empty() and price[st.top()]<=price[i]){
                               st.pop();
                           }
                           if(st.empty()){
                               ans.push_back(-1);
                           }
                           else{
                               ans.push_back(st.top());
                           }
                       }
                       st.push(i);
                   }
                   for(int i=0;i<n;i++){
                       ans[i]=i-ans[i];
                   }
                   return ans;
                }
            };
```

According to the stock span problem we need to find number of elements (**consecutive)** that are smaller to the current element. Basically, it is translated into nearest greater to the **left**

Maximum Area Histogram

```
class Solution {
public:
   vector<int> nsr(vector<int> arr){
      vector<int> ans;
     stack<int> st;
   int n=arr.size();
   for(int i=n-1;i>=0;i--){
   //Three conditions, if stack is empty
      if(st.empty()){
         ans.push_back(n);
      }
   //If element in stack is less than the element in question
      else if(st.size()>0 and arr[st.top()]<arr[i]){
         ans.push_back(st.top());
      }
   //If the stack top is greater than the element then simply pop
      else if(st.size()>0 and arr[st.top()]>=arr[i]){
         //pop elements
         while(st.size()>0 and arr[st.top()]>=arr[i]){
            st.pop();
         }
         //Semi-condition after popping the elements if the stack size is 0 then simply push -1
         if(st.size()==0){
            ans.push_back(n);
         }
         //Else remove push top element in the array
         else if(arr[st.top()]<arr[i]){
            ans.push_back(st.top());
         }

      }
      st.push(i);
   }
//Need to reverse the array
reverse(ans.begin(),ans.end());
   return ans;


   }
   vector<int> nsl(vector<int> &arr){
      vector<int> ans;
      stack<int> st;
      int n=arr.size();
      for(int i=0;i<n;i++){
   //Three conditions, if stack is empty
      if(st.size()==0){
         ans.push_back(-1);
      }
   //If element in stack is less than the element in question
      else if(st.size()>0 and arr[st.top()]<arr[i]){
         ans.push_back(st.top());
      }
   //If the stack top is greater than the element then simply pop
      else if(st.size()>0 and arr[st.top()]>=arr[i]){
         //pop elements
         while(st.size()>0 and arr[st.top()]>=arr[i]){
            st.pop();
         }
         //Semi-condition after popping the elements if the stack size is 0 then simply push -1
         if(st.size()==0){
            ans.push_back(-1);
            }
            //Else remove push top element in the array
            else if(arr[st.top()]<arr[i]){
                ans.push_back(st.top());
            }

        }
        st.push(i);
    }
//No need to reverse the array
    return ans;


    }

    int largestRectangleArea(vector<int>& heights) {
        int n=heights.size();
        vector<int> left_index=nsl(heights);
        vector<int> right_index=nsr(heights);
        vector<int> width;
        //Now calculate the width array
        for(int i=0;i<n;i++){
            width.push_back(right_index[i]-left_index[i]-1);
        }
        int ans=0;
        //calcualte the area
        for(int i=0;i<n;i++){
            ans=max(ans,heights[i]*width[i]);
        }
       return ans;
    }
};
```

Maximum Area Rectangle in binary matrix

```
    int maximalRectangle(vector<vector<char>>& matrix) {
        int n=matrix.size();
        int m=matrix[0].size();
        vector<int> hist(m,0);
        int ans=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(matrix[i][j]=='1'){
                    hist[j]+=1;
                }
                else{
                   hist[j]=0;
                }
            }
            ans=max(ans,largestRectangleArea(hist));
        }
        return ans;
    }
```

Rain water Trapping

```
class Solution {
public:
    int trap(vector<int>& height) {
        vector<int> lg;
        vector<int> rg;
        int n=height.size();
        //formula is min(maxL,maxR)-arr[i]
        int maxl=INT_MIN, maxr=INT_MIN;
        for(int i=0;i<n;i++){
            maxl=max(maxl,height[i]);
            lg.push_back(maxl);
            maxr=max(maxr,height[n-i-1]);
            rg.push_back(maxr);
        }
        reverse(rg.begin(),rg.end());
        int area=0;
        for(int i=0;i<n;i++){
            area+=min(lg[i],rg[i])-height[i];
        }
        return area;
    }
};
```

Minimum element in the stack with Extra Space

```
#include<vector>
class MinStack {
   //used vector as stack
    vector<int> st1;
    vector<int> st2;
public:
   /** initialize your data structure here. */
   MinStack(){

   }

   void push(int val) {
      //this will also update stack2 that will first check if the
      //top element is  greater than val then it will return min val
      st1.push_back(val);
      if(st2.empty()){
         st2.push_back(val);
      }
      else{
         int n=st2.size();
         //this is important
         if(st2[n-1]>=val){
           st2.push_back(val);
         }
      }
   }

   void pop() {
      //if the size is 0 then return 0
      if(st1.size()==0){
         return;
      }
      int val=st1[st1.size()-1];
      int n=st2.size();
      //this will check if the last element is also the min value
      if(n>0 && st2[n-1]==val){
         st2.pop_back();
      }
      st1.pop_back();
   }

   int top() {
      if(st1.size()==0){
         return 0;
      }
      int t=st1.back();
      return t;
   }

   int getMin() {
      if(st2.size()==0){
         return 0;
      }
      return st2.back();
   }
};
```

Minimum element in the stack with O (1) space

```
#include<vector>
class MinStack {
     vector<int> stack;
     long long int  min;
public:
    /** initialize your data structure here. */
    MinStack(){

    }

    void push(int val) {
        if(stack.size()==0){
            min=val;
            stack.push_back(val);
        }
        else if(val<min){
            stack.push_back((val-min)+ val);
            min=val;
        }
        else{
            stack.push_back(val);
        }
    }

    void pop() {
        int n=stack.size();
        if(n==0){
            return;
        }
        if(stack[n-1]<min){
            //this is an indication that we need to update the min
            min= (min-stack[n-1])+min;
        }
        stack.pop_back();
    }

    int top() {
        int n=stack.size();
        if(n==0){
            return -1;
        }
        else if(stack[n-1]<min){
            //this is an indication that we need to update the min
               return min;
        }
        else  return stack.back();

    }

    int getMin() {
        return min;
    }
};
```