Balanced Parentheses:

```
#include <iostream>
#include<stack>
using namespace std;

bool isParenthesesBalanced(string s){
    stack<char>st;
    for(int i=0;i<s.size();i++){
        if(s[i]==')'){
            if(!st.empty()){
                st.pop();
            }
            else{
                return false;
            }
        }
        else{
            st.push(s[i]);
        }
    }
    if(st.empty()){
        return true;
    }
    else{
        return false;
    }
}
int main() {
    string s=")(";
    bool flag=isParenthesesBalanced(s);
    cout<<flag;

    return 0;
}
```

Time:O(n)
space:O(n)

Valid Parentheses:

```
class Solution {
public:
    bool isValid(string s) {
         stack<char>st;
        for(int i=0;i<s.size();i++){
```

```
            if(s[i]=='('||s[i]=='{'||s[i]=='['){
                st.push(s[i]);
            }

                else if(s[i]==')'){
                    if(st.size()==0){
                        return false;
                    }
                    else if(st.size()>0&&st.top()!='('){
                        return false;
                    }
                    else{
                        st.pop();
                    }
                }
                else if(s[i]=='}'){
                     if(st.size()==0){
                        return false;
                    }
                    else if(st.size()>0&&st.top()!='{'){
                        return false;
                    }
                    else{
                        st.pop();
                    }
                }
                else if(s[i]==']'){
                     if(st.size()==0){
                        return false;
                    }
                    else if(st.size()>0&&st.top()!='['){
                        return false;
                    }
                    else{
                        st.pop();
                    }
                }
                else{
                    //do nothing
                }
        }
    }
};
```

Time:O(n)

Space:O(n)

Largest Rectangle in Histogram:

```
class Solution {
public:

    vector<int> NSEToTheLeft(vector<int>& heights){
        vector<int>ans;
        stack<pair<int,int>>s;
        for(int i=0;i<heights.size();i++){
            if(s.empty()){
                ans.push_back(-1);
                s.push({heights[i],i});
            }
            else if(!s.empty()&&s.top().first<heights[i]){
                 ans.push_back(s.top().second);
                s.push({heights[i],i});
            }
            else if(!s.empty()&&s.top().first>=heights[i]){
                while(!s.empty()&&s.top().first>=heights[i]){
                    s.pop();
                }
                 if(s.empty()){
                ans.push_back(-1);
                s.push({heights[i],i});
            }
            else if(!s.empty()&&s.top().first<heights[i]){
                 ans.push_back(s.top().second);
                s.push({heights[i],i});
            }
            }
        }
        return ans;
    }

    vector<int> NSEToTheRight(vector<int>& heights){
         vector<int>ans;
        stack<pair<int,int>>s;
        for(int i=heights.size()-1;i>=0;i--){
            if(s.empty()){
                ans.push_back(heights.size());
                s.push({heights[i],i});
            }
            else if(!s.empty()&&s.top().first<heights[i]){
                 ans.push_back(s.top().second);
```

```
                s.push({heights[i],i});
            }
            else if(!s.empty()&&s.top().first>=heights[i]){
                while(!s.empty()&&s.top().first>=heights[i]){
                    s.pop();
                }
                 if(s.empty()){
                ans.push_back(heights.size());
                s.push({heights[i],i});
            }
            else if(!s.empty()&&s.top().first<heights[i]){
                 ans.push_back(s.top().second);
                s.push({heights[i],i});
            }
            }
        }
        return ans;
    }
    int largestRectangleArea(vector<int>& heights) {
        vector<int>vec1=NSEToTheLeft(heights);
        vector<int>vec2=NSEToTheRight(heights);
        reverse(vec2.begin(),vec2.end());

         int maxArea=0;
         for(int i=0;i<heights.size();i++){
            int length=heights[i];
            int breadth=(vec2[i]-vec1[i]-1);
            maxArea=max(maxArea,length*breadth);
         }
         return maxArea;
    }
};
```

Time:O(n^2)
Space:O(n)

Maximal Rectangle:

```
class Solution {
public:
    vector<int> NSEToTheLeft(vector<int>& heights){
        vector<int>ans;
        stack<pair<int,int>>s;
        for(int i=0;i<heights.size();i++){
            if(s.empty()){
                ans.push_back(-1);
```

```
            s.push({heights[i],i});
        }
        else if(!s.empty()&&s.top().first<heights[i]){
             ans.push_back(s.top().second);
            s.push({heights[i],i});
        }
        else if(!s.empty()&&s.top().first>=heights[i]){
            while(!s.empty()&&s.top().first>=heights[i]){
                s.pop();
            }
             if(s.empty()){
            ans.push_back(-1);
            s.push({heights[i],i});
        }
        else if(!s.empty()&&s.top().first<heights[i]){
             ans.push_back(s.top().second);
            s.push({heights[i],i});
        }
        }
    }
    return ans;
}

vector<int> NSEToTheRight(vector<int>& heights){
     vector<int>ans;
    stack<pair<int,int>>s;
    for(int i=heights.size()-1;i>=0;i--){
        if(s.empty()){
            ans.push_back(heights.size());
            s.push({heights[i],i});
        }
        else if(!s.empty()&&s.top().first<heights[i]){
             ans.push_back(s.top().second);
            s.push({heights[i],i});
        }
        else if(!s.empty()&&s.top().first>=heights[i]){
            while(!s.empty()&&s.top().first>=heights[i]){
                s.pop();
            }
             if(s.empty()){
            ans.push_back(heights.size());
            s.push({heights[i],i});
        }
        else if(!s.empty()&&s.top().first<heights[i]){
             ans.push_back(s.top().second);
```

```
                s.push({heights[i],i});
            }
            }
        }
        return ans;
    }
    int largestRectangleArea(vector<int>& heights) {
        vector<int>vec1=NSEToTheLeft(heights);
        vector<int>vec2=NSEToTheRight(heights);
        reverse(vec2.begin(),vec2.end());

         int maxArea=0;
         for(int i=0;i<heights.size();i++){
            int length=heights[i];
            int breadth=(vec2[i]-vec1[i]-1);
            maxArea=max(maxArea,length*breadth);
         }
         return maxArea;
    }
    int maximalRectangle(vector<vector<char>>& matrix) {
        int rows=matrix.size();
        int columns=matrix[0].size();
        int maxArea=0;
        vector<int>vec(columns,0);
        for(int i=0;i<rows;i++){
            for(int j=0;j<columns;j++){
                if(matrix[i][j]=='1'){
                    vec[j]+=1;
                }
                else{
                     vec[j]=0;
                }
            }
            int area=largestRectangleArea(vec);
            maxArea=max(maxArea,area);
        }
        return maxArea;
    }
};
```

Time:O(n^3)
Space:o(n)