STACK

Q1) https://leetcode.com/problems/maximal-rectangle/submissions/1308889649/

```
class Solution {
   private:
   int area(vector<int>&heights)
   {
      int n=heights.size();
      int ans=0;
      stack<int>st;
      for(int i=0;i<=n;++i)
      {
         while(!st.empty() and (i==n ||
heights[st.top()]>=heights[i]))
         {
            int height=heights[st.top()];
            st.pop();
            int width;
            if(st.empty()) width=i;
            else width=i-st.top()-1;

            ans=max(ans,width*height);
         }
         st.push(i);

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

   }
};
```

Q2) https://leetcode.com/problems/sum-of-subarray-minimums/description/

```
class Solution {
public:
    vector<int> getNSL(vector<int>& arr, int n)
{
      vector<int> result(n);
      stack<int> st;

      for(int i = 0; i<n; i++) {
         if(st.empty()) {
            result[i] = -1;
        } else {
            while(!st.empty() && arr[st.top()] >
arr[i]) //strictly less
                st.pop();

            result[i] = st.empty() ? -1 : st.top();
        }
        st.push(i);
    }

    return result;
}

//This is just we are finding next smaller to
each element to right
//Similar : Leetcode-84
vector<int> getNSR(vector<int>& arr, int n)
{
    vector<int> result(n);
    stack<int> st;

    for(int i = n-1; i>=0; i--) {
        if(st.empty()) {
            result[i] = n;
        } else {
            while(!st.empty() && arr[st.top()] >=
arr[i]) //non-strictly less
                st.pop();

            result[i] = st.empty() ? n : st.top();
        }
            st.push(i);
        }

        return result;
    }

    int sumSubarrayMins(vector<int>& arr) {
        int n = arr.size();

        vector<int> NSL = getNSL(arr, n); //Next
smaller to left
        vector<int> NSR = getNSR(arr, n); //Next
smaller to right


        long long sum = 0;
        int M = 1e9+7;
        for(int i = 0; i<n; i++) {
            long long d1 = i - NSL[i]; //distance to
nearest smaller to left from i

            long long d2 = NSR[i] - i; //distance to
nearest smaller to right from i

            /*
                we have d1 numbers in the left and
d2 numbers in the right
                i.e. We have d1 options to start from
the left of arr[i]
                and d2 options to end in the right of
arr[i]
                so the total options to start and end
are d1*d2
            */
            long long total_ways_for_i_min =
d1*d2;

            long long sum_i_in_total_ways  = arr[i]
* (total_ways_for_i_min);


            sum  = (sum +
sum_i_in_total_ways)%M;
        }


        return sum;


    }


};
```

Q3) https://leetcode.com/problems/evaluate-reverse-polish-notation/description/

```
class Solution {
public:
    int evalRPN(vector<string>& tokens) {
        stack<int> st;
        int result = 0;

        unordered_map<string, function<int (int,
int)> > mp = {
            {"+", [](int a, int b) {return a + b; } },
            {"-", [](int a, int b) {return a - b; } },
            {"*", [](int a, int b) {return (long)a *
(long)b; } },
            {"/", [](int a, int b) {return a / b; } },
        };

        for(const string& s:tokens) {
            if(s == "+" || s == "-" || s == "*" || s ==
"/") {
                int b = st.top();
                st.pop();
                int a = st.top();
                st.pop();

                result = mp[s](a, b);
                st.push(result);
            } else {
                st.push(stoi(s));
            }
        }
        return st.top();
    }
};
```

Q4) https://leetcode.com/problems/simplify-path/description/

```
class Solution {
public:
 string simplifyPath(string path) {
 string token = "";

 stringstream ss(path);
 stack<string> st;

 while(getline(ss, token, '/')) {
 if(token == "" || token == ".")
 continue;

 if (token != "..")
 st.push(token);

 else if (!st.empty())

 st.pop();

 }


 string result = "";


 while(!st.empty()){ // add all the stack
elements

 result="/"+st.top()+result;

 st.pop();

 }


 if(result.length()==0) // if no directory or
file is present

 result="/"; // minimum root directory must
be present in result


 return result;

 }

};
```

Q5) https://leetcode.com/problems/validate-stack-sequences/description/

```
class Solution {

public:

    bool validateStackSequences(vector<int>&
pushed, vector<int>& popped) {

        stack<int> st;

        int n = pushed.size();

        int i = 0, j = 0;

        while(i < n && j<n) {


            st.push(pushed[i]);


            while(!st.empty() && j < n && st.top()
== popped[j]) {

                st.pop();

                j++;

            }

            i++;


        }


        return st.empty();

    }

};
```

Q6)

https://leetcode.com/problems/132-pattern/

```
class Solution {

public:

    bool find132pattern(vector<int>& nums) {

        int n  = nums.size();

        int num3 = INT_MIN;

        stack<int>st;


        for(int i = n-1; i >= 0; i--) {

            if(nums[i] < num3)

                return true;


            while(!st.empty() && nums[i] > st.top())
{
```

```
                num3 = st.top();
                st.pop();
            }
            st.push(nums[i]);
        }

        return false;
    }
};
```

2 POINTERS

Q1)

https://leetcode.com/problems/reverse-vowels-of-a-string/description/

```
class Solution {
public:
    bool isVowel(char &ch) {
        return ch =='a' || ch == 'e' ||
            ch =='i' || ch == 'o' ||
            ch == 'u' ||
            ch =='A' || ch == 'E' ||
            ch =='I' || ch == 'O' ||
            ch == 'U';
    }
    string reverseVowels(string s) {
        int n = s.length();
        int i = 0;
        int j = n-1;

        while(i < j) {
            if(!isVowel(s[i])) i++;

            else if(!isVowel(s[j])) j--;

            else {
                swap(s[i], s[j]);
                i++;
                j--;
            }
        }
        return s;
    }
};
```

Q2)

https://leetcode.com/problems/reverse-words-in-a-string/

```
//toclass Solution {
public:
    string reverseWords(string s) {
        stringstream ss(s);
        string token = "";

        string result = "";

        while(ss >> token) {
            result = token + " " + result;
        }

        return result.substr(0, result.length()-1);
    }
};
//2 pointer way

class Solution {
public:
    string reverseWords(string s) {
        //story
        //1. reverse whole string

        reverse(s.begin(), s.end());
        int i = 0;
        //hero honge hamare l and r jo revrese karenge words ko
        int l = 0, r = 0;
        int n = s.length();

        while(i < n) {
            while(i < n && s[i] != ' ') { //i ko agar char dikha to r ko dega and i++ and r++
                s[r] = s[i];
                r++;
                i++;
            }

            if(l < r) { //l     r
                reverse(s.begin()+l, s.begin()+r);

                s[r] = ' ';
                r++;

                l = r;
            }

            i++; //y eto badhta rahega
        }

        s = s.substr(0 , r-1);

        return s;

    }
};
```

Q3)

https://leetcode.com/problems/number-of-subsequences-that-satisfy-the-given-sum-condition/

```
class Solution {
public:
    int M = 1e9+7;
    int numSubseq(vector<int>& nums, int target) {
        int n = nums.size();

        sort(begin(nums), end(nums));

        vector<int> power(n);
        power[0] = 1;

        for(int i = 1; i<n; i++) {
            power[i] = (power[i-1] * 2) % M;
        }
        int l = 0, r = n-1;
        int result = 0;
        while(l <= r) {

            if(nums[l] + nums[r] <= target) {
                int diff = r-l;
                result = (result % M + power[diff]) % M;
                l++;
            } else {
                r--;
            }
        }


        return result;
    }
};
```

Q4) https://leetcode.com/problems/minimize-maximum-pair-sum-in-array/description/

```
class Solution {
public:
    int minPairSum(vector<int>& nums) {
        sort(begin(nums), end(nums));

        int maxResult = 0;
        int i = 0, j = nums.size()-1;

        while(i < j) {
            int sum = nums[i] + nums[j];

            maxResult = max(maxResult, sum);
            i++;
            j--;
        }

        return maxResult;

    }
};
```

Q5)

https://leetcode.com/problems/boats-to-save-people/description/

```
class Solution {
public:
    int numRescueBoats(vector<int>& people, int limit) {
        sort(people.begin(),people.end());
        int n=people.size();
        int i=0,j=n-1;
        int count=0;
        while(i<=j){
            int sum=people[i]+people[j];
            if(sum>limit){
                count++;
                j--;
            }else{
                count++;
                i++;
                j--;
            }

        }
        return count;
    }
};
```

Q6) https://leetcode.com/problems/count-the-number-of-good-partitions/solutions/4384369/count-non-overlapping-intervals/

```
class Solution {
public:
    int M = 1e9 + 7;
    int numberOfGoodPartitions(vector<int>& nums) {
        int n = nums.size();

        unordered_map<int, int> last_index;
//number, last index


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

            last_index[nums[i]] = i;

        }


        int i = 0;

        int j = max(0, last_index[nums[0]]);


        int result = 1;

        while(i < n) {

            if(i > j) { //we found one partition

                result = (result*2)%M;

            }


            j = max(j, last_index[nums[i]]);

            i++;

        }


        return result;

    }

};
```

Q7) https://leetcode.com/problems/minimum-length-of-string-after-deleting-similar-ends/

```
class Solution {

public:

    int minimumLength(string s) {

        int n = s.length();

        int i = 0, j = n-1;

        int count=

        while(i < j && s[i] == s[j]) {

            char ch = s[i];


            while(i < j && s[i] == ch) {

                i++;

            }


            while(j >= i && s[j] == ch) {

                j--;

            }


        }


        return j-i+1;

    }

};
```

SLIDING WINDOW

Q1)COUNT OCCURENCES OF ANAGRAMS

```
class Solution{
public:
  bool allZero(vector<int>& count) {
    return count==vector<int>(26, 0);
  }

        int search(string pat, string txt) {
          int k = pat.size();
          vector<int> count(26, 0);
          for(char &ch : pat) {
            count[ch-'a']++;
          }
          int i = 0, j = 0;
          int n = txt.size();
          int result = 0;
          while(j < n) {
            int idx = txt[j]-'a';
            count[idx]--;

            if(j - i + 1 == k) {
              if(allZero(count)) {
                result++;
              }

              count[txt[i]-'a']++;
              i++;
            }
            j++;
          }
          return result;
        }
};
```

Q2) **https://leetcode.com/problems/minimum-size-subarray-sum/description/**

```
class Solution {
public:
  int minSubArrayLen(int target, vector<int>& nums) {
      int n = nums.size();
      int i = 0, j = 0;

      int sum = 0;
      int minL = n+1;

      while(j < n) {
        sum += nums[j];

        while(sum >= target) {
          minL = min(minL, j-i+1);
          sum -= nums[i];
          i++;
        }

        j++;
      }
      return minL == n+1 ? 0 : minL;
  }
};
```

Q3) **First negative integer in every window of size k**

```
typedef long long ll;

vector<ll> printFirstNegativeInteger(ll A[], ll N,
ll K) {


    deque<ll> deq;

    vector<ll> result;


    for(ll i = 0; i<K; i++) {

      if(A[i] < 0)

        deq.push_back(i);

    }


    for(int i = K; i<N; i++) {

      if(!deq.empty()) {

        result.push_back(A[deq.front()]);

      } else {

        result.push_back(0);

      }


      while(!deq.empty() && deq.front() < i-
K+1) {

        deq.pop_front();

      }


      if(A[i] < 0)

        deq.push_back(i);

    }


    if(!deq.empty())

      result.push_back(A[deq.front()]);

    else

      result.push_back(0);


    return result;


  }
```

Q4) **https://leetcode.com/problems/minimum-window-substring/description/**

```
class Solution {

public:

  string minWindow(string s, string t) {

    int n = s.length();

    map<char, int> mp;


    for(char &ch : t) {

      mp[ch]++;

    }


    int requiredCount = t.length();

    int i = 0, j  = 0;

    int minStart  = 0;

    int minWindow = INT_MAX;

    while(j < n) {
          char ch_j = s[j];
          if(mp[ch_j] > 0)
             requiredCount--;

          mp[ch_j]--;

          while(requiredCount == 0) { //try to
shrink the window
             if(minWindow > j-i+1) {
                minWindow = j-i+1;
                minStart  = i;
             }

             char ch_i = s[i];
             mp[ch_i]++;
             if(mp[ch_i] > 0)
                requiredCount++;
             i++;
          }

          j++; //Don't ever forget this :-)
       }

       return minWindow == INT_MAX ? "" :
s.substr(minStart, minWindow);
    }
};
```

Q5)

https://leetcode.com/problems/contains-duplicate-ii/

```
class Solution {
public:
   bool containsNearbyDuplicate(vector<int>&
nums, int k) {
       int n = nums.size();

       unordered_set<int> st;

       int i = 0, j = 0;

       while(j < n) {

          //step-1
          if(abs(i-j) > k) { //abs(i-j) <= k
             st.erase(nums[i]);
             i++; //shrink
          }

          //past me dekha hai nums[j] ?
          if(st.find(nums[j]) != st.end()) {
             return true;
          }

          st.insert(nums[j]);
          j++;

       }

       return false;

    }
};
```

Q6)

https://leetcode.com/problems/count-subarrays-with-fixed-bounds/description/

```
class Solution {
public:
    long long countSubarrays(vector<int>& nums, int minK, int maxK) {
        long long ans = 0;

        int minPosition = -1;
        int maxPosition = -1;
        int leftBound   = -1;

        for(int i = 0; i < nums.size(); i++){
            if(nums[i] < minK || nums[i] > maxK)
                leftBound = i;//culprit index

            if(nums[i] == minK)
                minPosition = i;
            if(nums[i] == maxK)
                maxPosition = i;

            int count = min(maxPosition, minPosition) - leftBound;

            ans += (count <= 0) ? 0 : count;

        }

        return ans;
    }
};
```

Q7)

https://leetcode.com/problems/maximum-number-of-vowels-in-a-substring-of-given-length/description/

```
class Solution {
public:
    bool isVowel(char &ch) {
        return ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u';
    }

    int maxVowels(string s, int k) {
        int n = s.length();

        int maxV  = 0;
        int count = 0;
        int i = 0, j = 0;

        while(j < n) {

            if(isVowel(s[j]))
                count++;

            if(j-i+1 == k) {
                maxV = max(maxV, count);
                if(isVowel(s[i]))
                    count--;
                i++;
            }

            j++;
        }
        return maxV;
    }
};
```

Q8)

https://leetcode.com/problems/k-radius-subarray-averages/description/

```
//Approach-1 (Using Prefix Array)
class Solution {
public:
    vector<int> getAverages(vector<int>& nums, int k) {
        int n = nums.size();

        if(k == 0)
            return nums;

        vector<int> result(n, -1);

        if(n < 2*k + 1)
            return result;

        vector<long long> prefixSum(n, 0);
        prefixSum[0] = nums[0];

        for(int i = 1; i<n; i++) {
            prefixSum[i] = prefixSum[i-1] + nums[i];

        }

        for(int i = k; i<n-k; i++) {
            int left_idx  = i-k;
            int right_idx = i+k;

            long long sum = prefixSum[right_idx];

            if(left_idx > 0)
                sum -= prefixSum[left_idx-1];


            int avg = sum/(2*k+1);

            result[i] = avg;


        }

        return result;

    }
};
//approach-2 silding window

class Solution {
public:
    vector<int> getAverages(vector<int>& nums, int k) {
        int n = nums.size();

        if(k == 0)
            return nums;
        vector<int> result(n, -1);

        if(n < 2*k + 1)
            return result;

        long long windowSum = 0;

        int left  = 0;
        int right = 2*k;
        int i     = k;

        for(int i = left; i <= right; i++) {
            windowSum += nums[i];
        }



        result[i] = windowSum/(2*k+1);

        i++;
        right++; //Shifting window

        while(right < n) {

            int out_of_window  = nums[left];
            int came_to_window = nums[right];

            windowSum = windowSum -
out_of_window + came_to_window;

            result[i] = windowSum/(2*k+1);
            i++;
            left++;
            right++;

        }
        return result;
    }
};
};
```

Q9)

https://leetcode.com/problems/subarray-product-less-than-k/description/

```
class Solution {
public:
    int
numSubarrayProductLessThanK(vector<int>&
nums, int k) {
        if(k <= 1)
            return 0;
        int n = nums.size();
        int count = 0;

        int left = 0;
        int right = 0;
        int prod = 1;

        while(right < n){
            prod *= nums[right];

            while(prod >= k) {
                prod /= nums[left];
        left++;
      }

      count += (right-left)+1;
      right++;
    }
    return count;
  }
};
```

Q10)

https://leetcode.com/problems/length-of-longest-subarray-with-at-most-k-frequency/description/

```
class Solution {
public:
  int maxSubarrayLength(vector<int>& nums, int k) {
    int n = nums.size();

    unordered_map<int, int> mp;

    int i = 0;
    int j = 0;
    int result = 0;

    while(j < n) {

      mp[nums[j]]++;

      while(i < j && mp[nums[j]] > k) {
        mp[nums[i]]--;
        i++;
      }

      result = max(result, j - i + 1);
      j++;
    }

    return result;
  }
};
```

Q11)

https://leetcode.com/problems/count-subarrays-where-max-element-appears-at-least-k-times/description/

```
class Solution {
public:
  long long countSubarrays(vector<int>& nums, int k) {
    int maxE = *max_element(begin(nums), end(nums));

    int n = nums.size();
    int i = 0, j = 0;

    long long result = 0;
    int countMax = 0;

    while(j < n) {
      if(nums[j] == maxE) {
        countMax++;
      }

      while(countMax >= k) {
            result += n-j;

            if(nums[i] == maxE) {
                countMax--;
            }
            i++;
          }
          j++;
        }

        return result;
    }
};

//approach 2

class Solution {
public:
    long long countSubarrays(vector<int>& nums, int k) {
        int maxE = *max_element(begin(nums), end(nums));

        int n = nums.size();

        long long result = 0;

        vector<int> maxIndices;

        for(int i = 0; i < n; i++) {
            if(nums[i] == maxE) {
                maxIndices.push_back(i);
            }

            int size = maxIndices.size();
            if(size >= k) {
                int last_i = maxIndices[size-k];
                result += last_i+1;
            }
        }

        return result;
    }
};
```

Q12)

https://leetcode.com/problems/subarrays-with-k-different-integers/description/

```
class Solution {
public:
    int slidingWindow(vector<int>& nums, int k)
{
        unordered_map<int, int> mp;

        int n = nums.size();
        int i = 0;
        int j = 0;

        int count = 0;

        while(j < n) {
            mp[nums[j]]++;
            while(mp.size() > k) {
                mp[nums[i]]--;
                if(mp[nums[i]] == 0) {
                    mp.erase(nums[i]);
                }
                i++;
            }

            count += (j-i+1);
            j++;
        }

        return count;
    }

    int subarraysWithKDistinct(vector<int>& nums, int k) {
        return slidingWindow(nums, k) - slidingWindow(nums, k-1);
    }
};
```

Q13)

https://leetcode.com/problems/get-equal-substrings-within-budget/description/

```
class Solution {
public:
    int equalSubstring(string s, string t, int maxCost) {
        int n = s.length();

        int maxLen = 0;
        int currCost = 0;

        int i = 0, j = 0;
        while(j < n) {
            currCost += abs(s[j] - t[j]);

            while (currCost > maxCost) {
                currCost -= abs(s[i] - t[i]);
                i++;
            }

            maxLen = max(maxLen, j - i + 1);
            j++;
        }

        return maxLen;
    }
};
```

Q14)
https://leetcode.com/problems/longest-subarray-of-1s-after-deleting-one-element/description/

```
class Solution {
public:
    int longestSubarray(vector<int>& nums) {

        int zeroCount = 0;
        int longestWindow = 0;

        int i = 0;

        for (int j = 0; j < nums.size(); j++) {
            zeroCount += (nums[j] == 0);

            // Shrink the window until the zero counts come under the limit.
            while (zeroCount > 1) {

                zeroCount -= (nums[i] == 0);

                i++;

            }


            longestWindow = max(longestWindow,
j - i);

        }


        return longestWindow;

    }

};
```