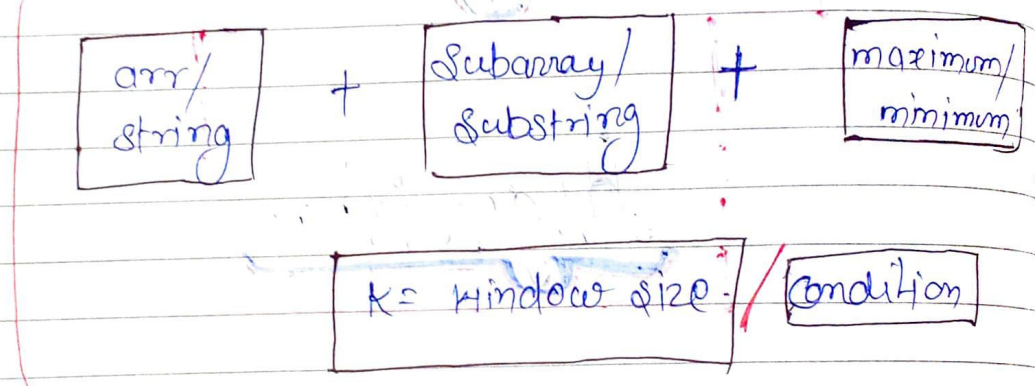
Sliding Window Algorithm

Sliding window technique is useful for solving problems in array or string, especially it is considered as a technique that could reduce the time complexity from $O(n^2)$ to $O(n)$.

Sliding Window Introduction Identification And Types.

It's a continuous block of array which dosen't break.

Identification :



Type

Fix

Given: arr + windowsize

min/max (target)

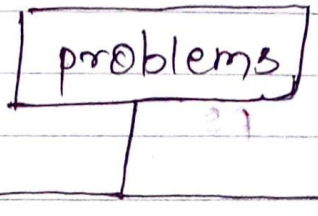
Variable size.

Given: arr + target

min/max (window size)

2.

Sliding Window Problems



Fixed	Variable
(1.) max/min sub array of size k.	(1.) largest/smallest subarray with sum k.
(2.) 1st -ve in every min size of k.	(2.) largest sum-string with k distinct character.
(3.) Count occurence of anagram.	(3.) length of largest substring with no repeeatative characters.
(4.) Max. of all subarray of size k.	(4.) Pick Toy.
(5.) max of min for every min size.	(5.) minimum Window substring.

3. **Max. sum subarray of size k**

PS - IP - OP ⇒

IP	PS	OP
size = 7 arr[] = [2, 5, 1, 8, 2, 9, 1] window size k = 3	return max of all window sum	19

CODE (Sliding Window Maximum)

```
int i = 0;
int j = 0
int sum = 0;
int mx = INT_MIN;

while( j < n.size() )
{
    sum += arr[j];        → Calculation

    if( j - i + 1 < K )
        j++;
```

If window size is less than given window size

When we reached its form ⇒ else if (j - i + 1 == k)

{

mx = max (mx , sum);

sum = sum - arr[i]; → // excluding first element.

maintaining the window size ← i++;
j++;

}

return mx; → // answer

}

Q: First Negative Number in Every Window of Size K

Given an arr.

arr[] : [12, -1, -7, 8, -15, 30, 16, 28]

K = 3

o/p : [-1, -1, -7, -15, -15, 0]

CODE

```
int i = 0;
int j = 0;
```
← Initialising variables

resultant array → `vector<int> res;`

```
deque<int> list;
```
← for storing useful element

```
while( j < arr.size())
{
```
← traversal

calculation ←
```
if(arr[j] < 0)
    list.push_back(arr[j]);
```

```
if(j-i+1 < k)
{
    j++;
    continue;
}
```

matching ← subarray with size k

When it's matched → `else if(j-i+1==k)`

```
{
    if(!list.empty())
        res.push_back(list.front());

    else
        res.push_back(0);
```

Calculating → ans

```
    if(arr[i] < 0)
        list.pop_front();

    i++;
}
j++;
}
```

Sliding → window

```
return res;
```

← returning ans;

6.

Maximum of all Subarray of size K.

Given.

arr[] = [1, 2, 3, 1, 4, 5, 2, 3, 6]

K = 3

o/p → [3, 3, 4, 5, 5, 5, 6]

Sliding Window - Generic Structure.

```
int i = 0;
int j = 0;

while(j < arr.size())
{
    calculation
    if(j - i + 1 < K)
        j++;
    else if(j - i + 1 == K)
    {
        ans
        slid the window
    }
```

Running Code

```
int i=0;
int j=0;

vector<int> res;
deque<int> dq;

while(j<arr.size())
{
    while(!dq.empty() && arr[j] >= arr[dq.back()])
        dq.pop_back();

    dq.push_back(j);

    if(j-i+1<k){
        j++;
        continue;
    }
    if(j-i+1==k)
    {
        res.push_back(arr[dq.front()]);

        while(!dq.empty() && dq.front()<=i-k)
            dq.pop_front();
        i++; j++;
    }
}
return res;
}
```

Calculation

maximum element

ding the window

(2nd variation)

7. Variable Size Sliding Window

Longest Subarray of Sum K

problem Statement :-

arr() = [4, 1, 1, 1, 2, 3, 5]

K = 5 Sum (K) = 5

O/P → 4

CODE

```
int i = 0;
int j = 0;

int Sum = 0;
int mx = INT_MIN;

while (j < arr.size())
{
    Sum += arr[j];   → Calculation
```

1st case ←

```
if (sum < k)
    j++;
```

```
else if (sum == k)
{
    mx = max(mx, j-i+1);
    j++;
}
```

2nd case ← → Calculating ans

```
else if (sum > k)
{
    while (sum > k)
    {
        sum = sum - arr[i];
        i++;
    }
    j++;
}
```

3rd case ← → removing from front

```
}
return mx;
}
```

→ returning ans

Note 1) This approach will only work for positive integer array.

⇒ For negative one +ve integer use unordered_map

8. **Longest Substring With K Unique Characters**

IP: ⇒

S = aabacbebebe

K = 3

O/P = 7

CODE

```
unordered_map<char, int> mp;
int i=0, j=0, mx=0;
while(j < arr.size())
{
   mp[arr[j]]++;
   if(mp.size() < K)
      j++;
   else if(mp.size() == K)
   {
      mx = max(mx, j-i+1);
      j++;
   }
```

```
else if(mp.size() > K)
;

    while(mp.size() > K)
    {
        mp[arr[i]]--;
        if( mp[arr[i]]==0)
            mp.erase(arr[i]);

        i++;
    }
  j++;
}


if( mx==0 || arr.size()<K)
        return -1;

return mx;

}
```

Q.) **Longest Substring With Without Repeating Characters.**

Given ⇒

s = satyam

O/P ⇒ 4

CODE

```
unordered_map<char, int> mp;
int i = 0; j = 0;  mx = INT_MIN;

while(j < s.length())
{

    mp[s[j]]++;

if(mp.size() == j-i+1)
{
    mx = max(mx, j-i+1);
     j++
}
```

```
else if( mp.size() < j-i+1 )
{
    while( mp.size < j-i+1 )
    {
        mp[s[i]]--;
        if( mp[s[i]] == 0 )
            mp.erase( s[i] );

        i++;
    }
j++;
}

return mx;

}
```