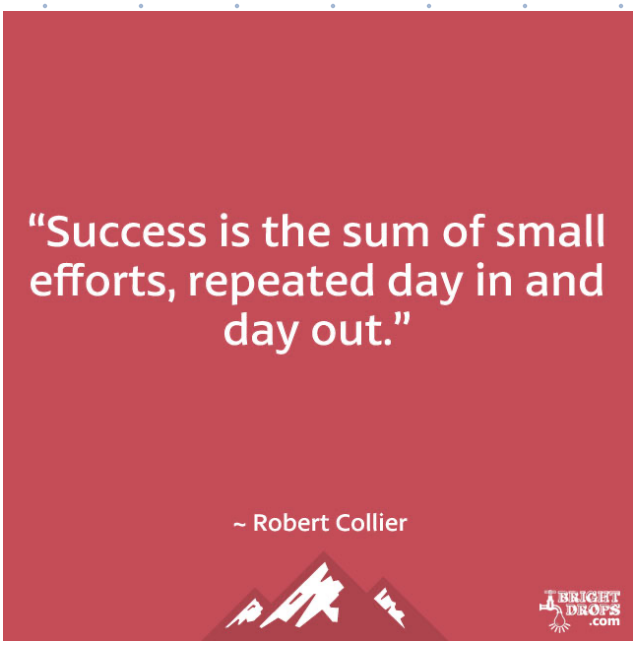
Sliding Window & Contribution Technique



Good Evening

AGENDA:

01. Max sum out of all possible subarr
02. Find sum of all subarr sums (Google & Fb)
03. Max subarr sum with length k

01. Given an arr of integers, find sum of all possibble subarrays of array & maintain the maximum sum.

$$A[] = \{1, 2, 3\}$$
$$\quad\;\; 0 \quad 1 \quad 2$$

sub arr		sum
[0 0]	= {1}	1
[0 1]	= {1, 2}	3
[0 2]	= {1, 2, 3}	6
[1 1]	= {2}	2
[1 2]	= {2, 3}	5
[2 2]	= {3}	3

Ans = 6

BF → Find all the subarrays, iterate & get sum of subarray → maintain the overall maximum.

```
ans = -∞
for (i=0; i<n; i++){
    for (j=i; j<n; j++){
        sum = 0
        for (k=i; k≤j; k++){
            sum += A[k];
        }
        if (sum > ans) ans = sum;
    }
}
```

TC: $O(n^3)$
SC: $O(1)$

* Optimisation 1 : Use prefix sum Approach

```
if (st==0)      sum = pf[end]

else            sum = pf[end] - pf[st-1];
```

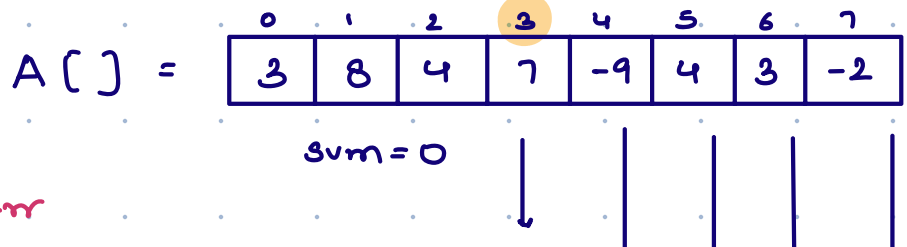
* Construct pf[n] array.

```
ans = -∞
for (i=0; i<n; i++){
    for (j=i; j<n; j++){
        sum=0
        if (i==0)   sum = pf[j];
        else        sum = pf[j] - pf[i-1];
        if (sum>ans) ans = sum;
    }
}
```

TC: $O(n^2)$
SC: $O(n)$

* Idea 3 → Use carry forward Approach

Q Given A[N], print subarr sums starting from index 3.



[3 3] = 7

[3 4] = -2

[3 5] = 2

[3 6] = 5

[3 7] = 3

sum = 0 + 7

sum = 7 + (-9)

sum = -2 + 4

sum = 2 + 3

sum = 5 + (-2)

```
void printsum( int []A, int i )        → 3
|  int sum = 0
|  for (j = i : j < n ; j++) {
|  |  sum += A[j]
|  |  print (sum);
|  }
}
```

$i = 3$ → printing sum of all subarr starting from idx = 3

$i = 4$ → printing sum of all subarr starting from idx = 4

$i = 2$ → printing sum of all subarr starting from idx = 2

```
void printsum( int [] A)
    ans = -∞
    for(i=0; i<n; i++){
        int sum=0
        for(j=i : j<n ; j++){
            sum += A[j]
            if (sum > ans) ans = sum;
        }
    }
}
```

TC: $O(n^2)$

SC: $O(1)$

Q Given an array of integers, find total sum of all possible subarrays.

constraints

i'm not allowed to generate all subarrays

$A[] = \{1, 2, 3\}$ (indices 0, 1, 2)

sub arr		sum
[0 0]	= {1}	1
[0 1]	= {1, 2}	3
[0 2]	= {1, 2, 3}	6
[1 1]	= {2}	2
[1 2]	= {2, 3}	5
[2 2]	= {3}	3

total sum =

1+3+6+2+5+3

= 20

* Carry forward technique

```
void printsum( int [] A)
    ans = 0
    for(i=0; i<n; i++){
        int sum = 0
        for(j=i ; j<n ; j++){
            sum += A[j]
            ans = ans + sum;
        }
    }
    return ans;
}
```

TC: $O(n^2)$

SC: $O(1)$

* Optimal solution = Contribution Technique

A[] = {6 8 -1 7}
0 1 2 3

subarr

		sum
[0 0]	= {6}	= 6
[0 1]	= {6, 8}	= 14
[0 2]	= {6, 8, -1}	= 13
[0 3]	= {6, 8, -1, 7}	= 20
[1 1]	= {8}	= 8
[1 2]	= {8, -1}	= 7
[1 3]	= {8, -1, 7}	= 14
[2 2]	= {-1}	= -1
[2 3]	= {-1, 7}	= 6
[3 3]	= {7}	= 7

Sum = 94

A[i] * contribution

6 * 4 = 24

8 * 6 = 48

-1 * 6 = -6

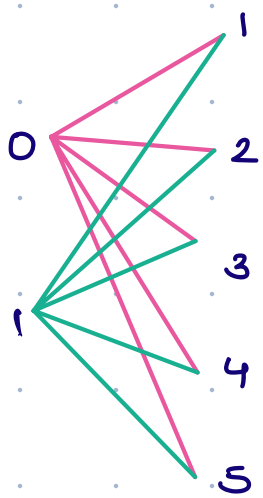
7 * 4 = 28

Sum = 94

Quiz - In how many subarrays, index 1 will be present?

	0	1	2	3	4	5
A[] = {	3	-2	4	-1	2	6 }

st end



[0,1) [0,2] [0,3] [0,4] & [0,5]

[1,1) [1,2] [1,3] [1,4] & [1,5]

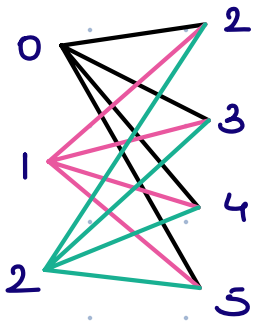
Total subarrays = 2 * 5

= 10

In how many subarrays, index 2 will be present?

	0	1	2	3	4	5
A[] = {	3	-2	4	-1	2	6 }

st end



Total no. of subarr containg idx 2 = 3 * 4 = 12

count = b - a + 1

st [0 - i] (a) end [i n-1] (b)

$i - 0 + 1$ $\quad$ $n - 1 - i + 1$

$(i+1) * (n-i)$

> Total no. of subarr containing idx i $= (i+1) * (n-i)$;

	0	1	2	3
A[] =	6	8	-1	7
$(i+1)$ =	1	2	3	4
$(n-i)$ =	4	3	2	1

Total no. of subarr containing i idx =	4	6	6	4

ans = $6*4 + 8*6 + (-1)*6 + 7*4 = 94$

```
int totalsum (int [] A) {
    ans = 0
    for (i=0; i<n; i++) {
        int count = (i+1) * (n-i);
        ans = ans + A[i] * count ;
    }
    return ans;
}
```

TC: O(n)

SC: O(1)

10:07 pm → 10:17 pm

* Given an integer K, how many subarrays can be formed of size K in A[n]

A[] = { 1 2 3 4 5 }

0 1 2 3 4

K = 1 = 5 (5 - 1 + 1)

K = 2 = 4 (5 - 2 + 1)

K = 3 = 3 (5 - 3 + 1)

K = 4 = 2 (5 - 4 + 1)

} N - K + 1 subarrays

Q Given an array of size N, print st idx & end idx of all the subarr of length k

$A = \{ \underset{0}{10} \quad \underset{1}{20} \quad \underset{2}{30} \quad \underset{3}{40} \quad \underset{4}{5} \}$ $K = 3$

st	end
0	2 (k-1)
1	3
2	4

```
st = 0 , end = k-1
while (end < n) {
    print (st + " " + end);
    st ++;
    end ++;
}
```

Q Given A[N], print maximum subarr sum of length k

A[ ] =

-3	4	-2	5	3	-2	8	2	-1	4
0	1	2	3	4	5	6	7	8	9

$K = 5$

subarr

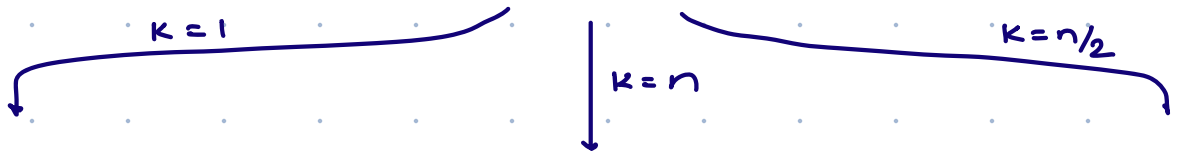
st	end	sum
0	4	7
1	5	8
2	6	12
3	7	16
4	8	10
5	9	11

Ans = 16

Brute force → For every subarray of size k, calculate sum & update maximum.

```
int maximumsum ( int [] A, int k )
{
    st = 0 , end = k-1 , ans = -∞
    while ( end < n ) {
        int sum = 0
        for ( i = st ; i ≤ end ; i++ ) {
            sum += A[i];
        }                                   } k times
        if (sum > ans) ans = sum ;
        st ++ ;
        end ++ ;
    }                                       } n-k+1
}
```

TC: $O(k * n-k+1)$



TC: $O(1 * n-1+1)$

: $O(n)$

TC: $O(n * (n-n+1))$

TC: $O(n)$

TC: $O(\frac{n}{2} * (n-\frac{n}{2}+1))$

: $O(\frac{n}{2} * (\frac{n}{2}+1))$

: $O(\frac{n^2}{4} + \frac{n}{2})$

TC $\leq O(n^2)$

* Approach 2 (prefix sum Approach)

```
int maxsum ( int [] A, int k) {
    → create psum [n]
      s=0 , end= k-1 , ans= -∞
      while ( end < n) {
          int sum=0;
          if ( st == 0) sum= psum[end]
          else sum= psum [end] - psum [st -1];
          if (sum> ans) ans= sum;
          st ++;
          end++ ;
      }
}
```

TC: $O(n)$

SC: $O(n)$

* Idea 3 : Sliding window

Carry forward + Fixed size window

A[ ] =

-3	4	-2	5	3	-2	8	2	-1	4
0	1	2	3	4	5	6	7	8	9

K = 5

st	end	sum
0	4	7
1	5	8 = 7 - A[0] + A[5] 7 - (-3) + (-2)
2	6	12 = 8 - A[1] + A[6] = 8 - 4 + 8 = 12
3	7	16 = 12 - A[2] + A[7] = 12 - (-2) + 2 = 16
.	.	
.	.	
.	.	

slide window = sum - A[st-1] + A[end]

```
int maxsum ( int []A, int k)
{
    // create first window
    sum=0 , ans= -∞
    for ( i=0; i<k; i++){
        sum += A[i];
    }
    if (sum > ans) ans= sum;

    st=1 , end= k
    while (end < n ){
        // slide window
        sum = sum - A[st-1] + A[end]
        if (sum > ans) ans= sum;
        st ++;
        end ++;
    }
    return ans;
}
```

k times

TC: O(n)
SC: O(1)

n-k

TC: O(k+n-k)
: O(n)

Doubts

{10 20 30 40 50} B = 3

0 1 2 3 4

↑ ↑

		maxsum
{10 20 30}	{ }	60
i = B; i--; {10 20}	j = n; j--; {50}	80
i--; Drop {10}	j-- gain {40 50}	100
{ }	{30 40 50}	120

% $10^9 + 7$ → very large prime

we want to have the answer inside int range

- long
  - ↳ % $10^9 + 7$

* 24' June

DSA 1 = 1D

Arrays 2 = 2D

Arrays 3 = Interview problems

Bit Manipulation 1 & 2

Recursion 1 & 2

Maths : Mod & GCD

OOPS : 1 & 2

Hashing 1 & 2

Sorting : 1 & 2

Searching : BS 1, 2 & 3

LL, Stacks(2), Queues

Trees (4 classes) + 1

Maths : Combinatorics & prime no

Backtracking 1 & 2

LL : Interview prob

Hashing : Implementation

2 Heaps & 1 Greedy

4 DP

4 Graph

DSA : Interview problems