Array Problems (II)

Kadane Algo – Largest sum continuous subarray.

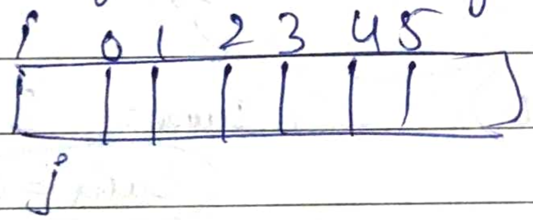
7

-2	-3	4	-1	-2	1	5	-3

Brute Force – ~~take~~ find every subarray & calculate their sum.

→ Maximum sum among them is the answer.

How to find subarrays?




```
for(i=0 →<n)
{  sum=0
for (j=i ⟶ <n)
  {
     sum+ =arr[j]
  }
```

This is not a good approach because of high complexity.

Optimised Approach -

Single traversal → Kadane's Algo

I/P array

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

Sum → tracks value of sum
maxi → tracks max. number

sum = 0
maxi = INT_MIN

I) i = 0
sum = 0 + (-2) := -2

II) Naya answer aya kya?
ans = max (INT_MIN, -2)

III) if (sum < 0) → sum = -2 < 0
sum = 0 → Sum = 0

↳ but why?
because

A	B
-ve	+

↑ if we add -ve part in sum it will decrease the sum
ie sum = A + B
sum = $\underbrace{-A}_{-ve}$ + B

→ thats why sum ko 0 kardo if no. is negative because to prevent sum from decreasing.
∴ sum ko 0 kardo

Page No.:
Date:
YOUVA

dry run →

$i=1$ $sum = 0 + (-3) = -3$

$ans = max(-2, -3) = -2$

$sum = 0$.

$i=2$ $sum = 0 + 4 = 4$

$ans = max(-2, 4) = 4$

$sum => 0$ ✓

$i=3$ $sum = 4$, $ans = 4$

$sum = 4 + (-1) = 3$

$ans = max(4, 3) = 4$

$sum < 0$ X

$i=4$ $sum = 3 + (-2) = 1$

$ans = max(4, 1) \rightarrow 4$

$sum => 0$ ✓

$i=5$

$sum = 1 + 1 = 2$

$ans = max(4, 2) = 4$

$sum \geq 0$ ✓

$i=6$

$sum = 2 + 5 = 7$

$ans = max(4, 7) \rightarrow 7$

$sum >= 0$ ✓

$i=7$

$sum = 7 + (-3) = 4$

$ans = max(7, 4) = 7$

$sum = 4 > 0$ ✓

Now return ans → $\underline{\underline{7}}$

```
int getMaxSubArraySum(int arr[], int n)
{
    int maxSF =INT_MIN; int ans =INT_MIN;
    int maxEH =0;        int sum =0;
for(int i=0; i<n; i++)
  {
    sum = sum + arr[i];
   ans = max(ans, sum);
  if (sum <0)
      sum =0;
  }
  return ans;
}
```

To print Exact Subarray which gives largest sum-

keep track of Starting Index and Ending Index } of window

* Jaha answer store krwa rhe h, wahi start and end index ko set krdo.

<u>Time Complexity</u> → used to show performance of a code.
↳ whether it is fast
↳ whether it is low

T(C) is basically time taken by an algorithm.
→ represented as a function of input.

Different Notations such as –
Big O → $O(n)$ → Upper Bound
Theta → $\Theta(n)$ → Average Case Complexity
Omega → $\Omega(n)$ → Lower Bound

① <u>Big O Notation $O(n)$</u> –

example → $O(n)$ → linear
$O(1)$ → constant
$O(n^2)$ → quadratic
$O(n^3)$ → cubic
$O(\log n)$ → logarithmic

eg →
```
for(i=0; i<n; i++)
{
   cout << arr[i];
}
```

i ↑ ↑ ↑ ↑ ↑ ↑ n

total no. of operation → n

<u>$T(C) \to O(n)$</u>

Page No.:
Date:

* for (int i=0 ; i<10; i++)
{
———
———
}

constant
T(C) = O(1)
because we know ki 10 operations hoge and 10 → constant.

* int i = 101;
while (i--)
{
cout << "mera bhai";
}

T(C) = O(1)

* for(int i = 0 → n)
{
for (j → 0 → n)
{
cout << "babbar";
}
}

i=0 j=0,1,2 --- n
i=1 j=0 ---- n
i=(n-1) j=0 ---- n

$T(C) = O(n^2)$

Why we need T.C.?
→ To do comparison b/w algos.

* for(int i → 0 → <n)
{
for (int j = i → <n)
{
cout → → ~
}
}

$T(C) = O(n^2)$

TC.

Linear search → $O(n)$
Reverse array → $O(n)$
max/min element → $O(n)$

Depict in Big O Notation –

$f(n) = n + 3n \rightarrow 4n \quad O(4n) \rightarrow O(n)$.

$f(n) = \frac{n^2}{4} \rightarrow O\left(\frac{n^2}{\cancel{4}}\right) = O(n^2$

$f(n) = n^3 + \underbrace{2n}_{} \rightarrow n^3 \rightarrow O(n^3)$

discard lower power term

$f(n) = 311 \rightarrow O(1)$

$f(n) = n^3 + \frac{n^3}{5} \rightarrow O(n^3(1 + \frac{\cancel{1}}{5})) \rightarrow O(n^3)$

~~f(n)~~

$f(n) = n^3 + n^6 + n^{2/3} \rightarrow O(n^6)$

$f(n) = n^2 + 1 \rightarrow O(n^2)$

$f(n) = 2n \rightarrow O(\cancel{2}n) \rightarrow O(n)$.

eg

```
int main()
{
for (int 0 → n)
  {
  ______
  ______
  }
for( int 0 → m)
  {
  ______
  ______
  }
}
```

$T(C) = O(n)$

+

$T(C) = O(m)$

Total

$\boxed{O(n+m)}$

This is wrong.
Take only max of m and n

Space Complexity

↓

space taken by program

(1)
```
int i=0
for (i=0; i<=n; i++)
  {
    cout << - - - -
  }
```
SC = O(1)

because we already know size of int ie 4 bytes.

(2)
```
~~for~~ {
    int arr[n]
  }
```
SC → O(n)

because n blocks taken

(3)
```
{
  int arr[n][n]
}
```
SC = $O(n^2)$

(4)
```
{
  int arr[n]
  int arr[m]
}
```
SC = O(n+m)

Note:- for SC →

↳ Jitna bhi space h „sab" "+" kardo. Dont find max.

H/w → T/C and S/C
↳ GFG
↳ Codestudio
↳ Interviewbit

Mock MCQs.

bad

$O(n!)$
$O(n^3)$
$O(n^2)$
$O(n \log n)$
$O(n)$
$O(\log n)$
$O(1)$

good