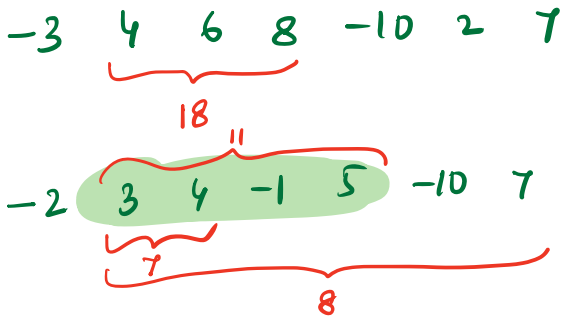
Q1) Given an array, find the max sum of contiguous elements of the array.

Direct I
Google
Linked In
Amazon

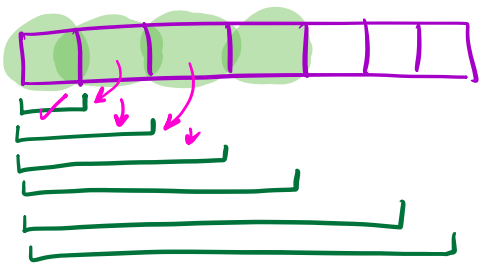


no. of subarrays → $\frac{n(n+1)}{2}$

Brute force → check all subarrays

```
ans = -∞/ar[0]
for start in [0, n-1]:          ——→ O(n)
    for end in [start, n-1]:    ——→ O(n)
        sum = 0
        for idx in [start, end]:    } O(n)
            sum += ar[idx]
        ans = max(ans, sum)
```

$O(n^3)$ T.C.
$O(1)$ S.C.



```
ans = -∞ / ar[0]
for start in [0, n-1]:
    sum = 0
    for end in [start, n-1]
        sum += ar[end]      // stores the sum [start, end]
        ans = max(ans, sum)
```

O(n²) T.C.
O(1) S.C.

Test case 1 :- 3 4 9 1 2

All +ve ⇒ choose entire array

Test case 2 :- [-ve] [+ve] [-ve]

Sum of +ve elements

Test case 3 :- [+ve] [-ve] [+ve]

[4 6 8] -10 2 7

[3 4 -1 5]

At every index i :- find max sum subarray ending at index i.

	-12	5	6	7	-3	2	-10	-12	8	12	21	-4	7
sum	-12	5	11	18	15	17	7	-5	8	20	41	37	44
ans	-12	5	11	18	18	18	18	18	18	20	41	41	44

$$sum[j] = \max_{i \in [0,j]} (ar[i] + ar[i+1] + \dots + ar[j])$$

$$sum[j] = \max(sum(ar[0] \dots ar[j]), sum(ar[1], \dots, ar[j]), sum(ar[2], \dots, ar[j]), \dots, sum(ar[j]));$$

$$= ar[j] + \max(sum(ar[0], \dots, ar[j-1]), sum(ar[1], \dots, ar[j-1]), \dots, sum(ar[j-1]), 0)$$

$sum[j-1]$

$$\boxed{sum[j] = ar[j] + \max(sum[j-1], 0)}$$

$\max(a+b, a+c, a)$

$= a + \max(b, c, 0)$

$$\boxed{sum[j] = ar[j] + \max(sum[j-1], 0)}$$

	-3	7	2	-6	-4	-2	8	-9
sum →	-3	7	9	3	-1	-2	8	-1

→ 9

$+8 + \max(-2, 0)$

$-9 + \max(8, 0)$

$2 + \max(7, 0)$

$-6 + \max(9, 0) = 3$

$-4 + \max(3, 0) = -1$

$-2 + \max(-1, 0) = -2$

```
sum = 0
ans = ar[0]
for(i=0; i<n; i++){
    sum = ar[i] + max(sum, 0);
    ans = max(ans, sum);
}
```

O(n) T.C.
O(1) S.C.

Kadane's Algorithm

```
sum = 0
ans = ar[0]
s = 0, sc = 0, e = 0
for(i=0; i<n; i++){
    if(sum < 0){
        sum = 0;
        sc = i; // start index of current subarray
    }
    sum += ar[i];
    if(ans < sum){
        ans = sum;
        s = sc;  // start index of best subarray till now
        e = i;   // end index of best subarray till now
    }
}
return (s,e);
```

[Break till 10:52 PM]

Q2) Given an array of size n, find the smallest +ve no. which is NOT present in the array.

Google

1 2 -3 6 1 4

Ans → 3

1 2 3 4 5 6

Ans → 7

-2, 4, 6, -1, 9, 1, 10, 2, 3

Ans → 5

Approach 1: Check for each +ve integer starting from 1. The first integer not present in the array is the answer.

smallest ans → 1

largest ans → n+1

isPres(1) →

isPres(2) →

isPres(3) →

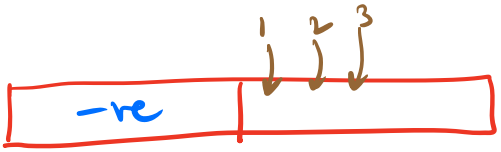
⋮

isPres(n+1) →

```
n = ar.size()
for elem=1; elem<=n; elem++){
    flag = 0
    for(i=0; i<n; i++){
        if(ar[i] == elem) {
            flag=1;
            break;
        }
    }
    if(flag ==1)
        continue;
    return elem;
}
return (n+1);
```

$O(n^2)$ T.C.
$O(1)$ S.C.

Approach 2 :- sort and check the elements one by one.



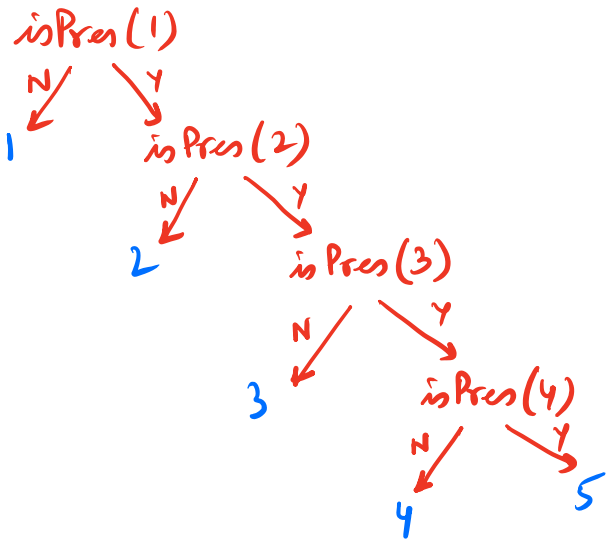
$O(n \log n)$ T.C.

Approach 3 :- HashSet

→ Iterate and store all elements of array
→ Iterate in [1, n+1] and find first missing element.

[O(n) T.C. on average]
O(n) S.C.

n = 4



Approach 4 :- Create a seperate bool array pres[]

pres[x] = T if x ∈ ar
F if x ∉ ar

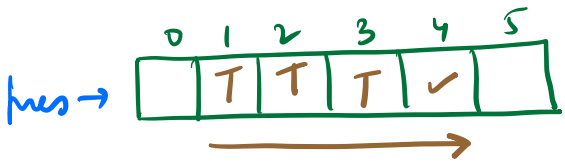
ar[i] > n ⇒ Max (n-1) elements can be present in the range [1, n]

⇓

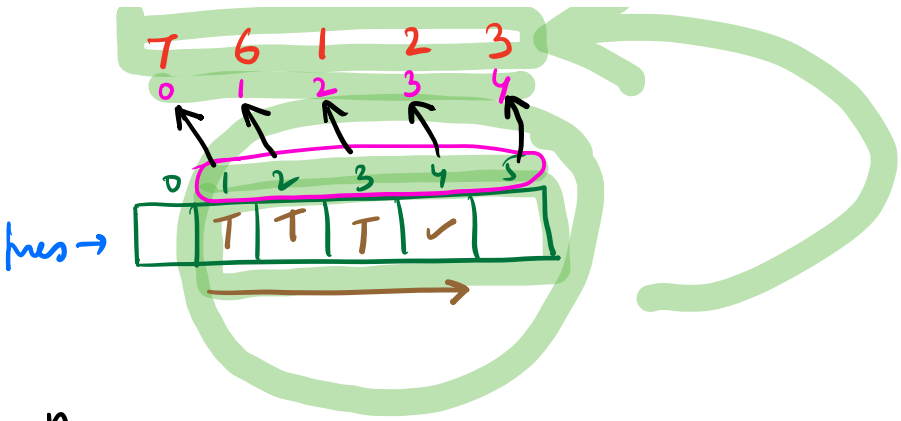
atleast 1 value is absent from [1, n].

T.C. → O(n)
S.C. → O(n) extra space

T 6 1 2 3 n=5



pres[ar[i]] = true;

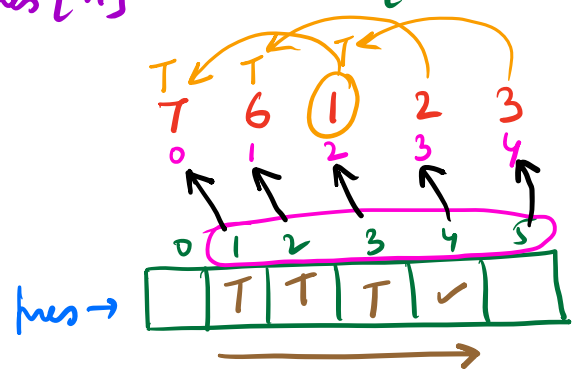


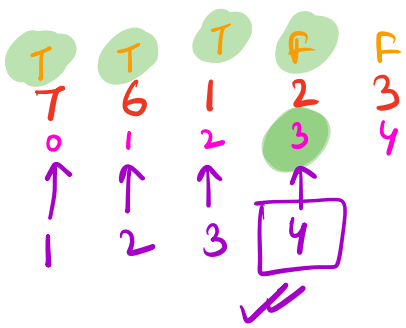
n

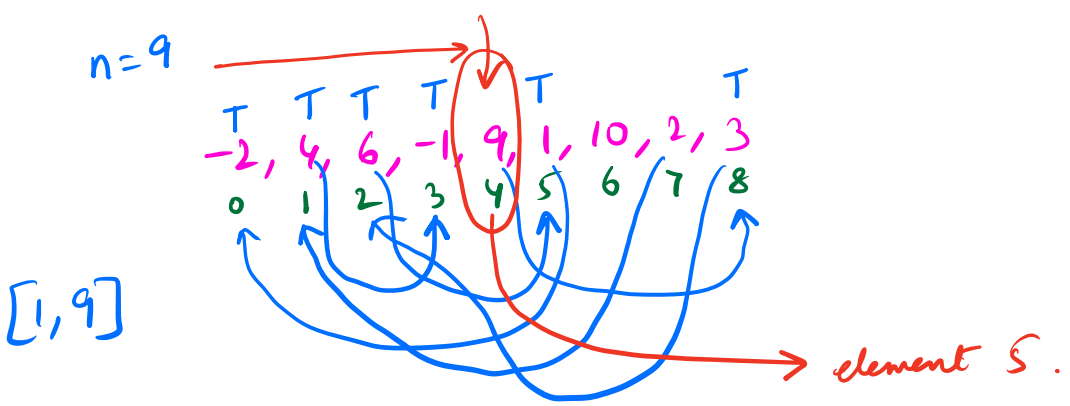
hres[1] → ar[0]
hres[2] → ar[1]
hres[3] → ar[2]

hres[n-1] → ar[n-2]
hres[n] → ar[n-1]

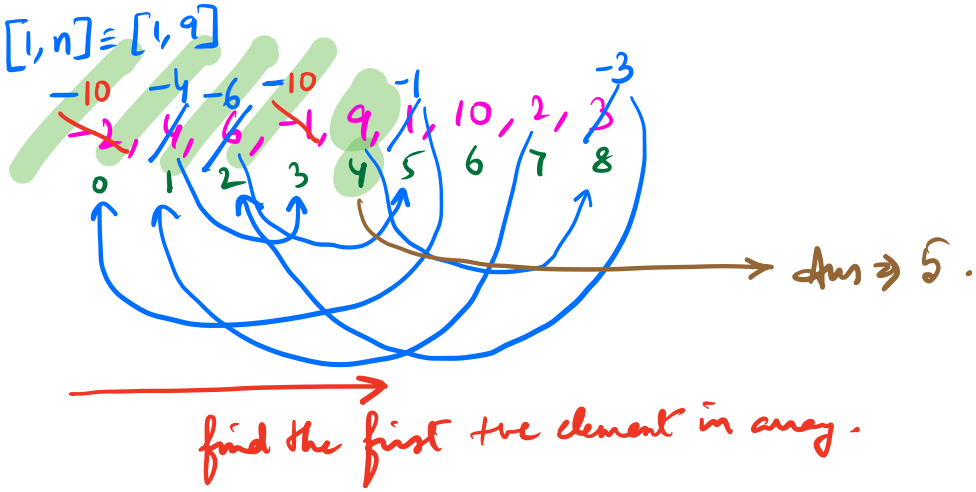
x → hres[x] → ar[x-1]







Use -ve sign to represent T/F.




```
for(i=0; i<n; i++){
    if(ar[i]<=0)
        ar[i]=n+1;
}
for(i=0; i<n; i++){
    if(abs(ar[i])<=n){
        ar[ar[i]-1] = -1*abs(ar[ar[i]-1]);
    }
}
for(i=0; i<n; i++){
    if(ar[i]>0)
        return (i+1);
}
return (n+1);
```