Let $arr[5]$:

0	1	2	3	4

Let $arr[n]$:

→ Access i^{th} index : $arr[i]$ → $O(1)$

→ Given an Array, N, print

```
T=0; Tc N; T++{
    print(arr[T])
}
```

→ time

→ space

1) Given an array, count no. of elements here odd sum + element

Approach: Brute Force Method

0	1	2	3	4	5
1	2	3	4	5	6

2) $arr[7]$: { 3, -2, 5, 3, 7, 8, -1 }

Algo:
1) Select element
2) Take element
3) Move ahead to count

Approach: Nested Loop

Step: No. of elements less

Step: Max. element

Step: Max. sum

0	1	2	3	4	5	6
1	1	1	1	1	1	1

Step: Find Max. sum

Step: Find Max. Prev

Boundaries:

Max = ___ ; for i=0;

```
T=0; Tc N; T++{
    if(max < arr[T]){
        max = arr[T]
    }
    if(max == arr[T]){
        cnt
    }
}
```

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

max = arr[0] | cnt = 1

```
T=0; Tc N; T++{
    if(max < arr[T]){
        max = arr[T]
    }
    if(max == arr[T]){
        cnt
    }
}
return N-C
```

$arr[7] = \{-3, -4, -5, -1\}$

max = -3

cnt = 1

Step: Take 1 loop

Expr: $O(N)$

3) Given N-array elements, check if two equal or pair (T/F)

Input Value $arr[5] = arr[5] \rightarrow$ TC $\{1, 2, 3\}$

Pick Value Check vs Extra Space

$arr[7] = \{3, -2, 1, 5, 3, 6, 8\}$

$K = 10$; $\{ 7, 3 \}$

$K = 9$ → Not possible

Approach
1) Read
2) Observations
3) Logic

Logic: Fix any pair check if sum is K. $\{-4\} \rightarrow \{-4\} \rightarrow O(n^2)$

$arr[5] = \{2, 1, 3, 5\}$

Pairs:

2	[2, 1]	[2, 3]
1	[1, 3]	[1, 5]
3	[3, 5]	[3, 5]
5	[5, 3]	[5, 5]

```
T=0; Tc N; T++{
    T2=T; Tc N; T2++{
        if(arr[T]+arr[T2]==K){
            return True
        }
    }
}
return False
```

→ Iterations:

	1	2	Sum
0	[1, N-1]		Sum N-1 : Nested
1	[2, N-2]		→ $(N-1)(N-1)$
2	[3, N-3]		N-2 → $N^2 - N$
			$\Rightarrow \frac{N^2 - N}{2}$
N-1	[N, N-N]		0 → $N^2 - N$
			→ $O(N^2)$

→ Time Complexity: $O(N^2)$

4) Given an $arr[n]$ Take K, among n $\Rightarrow$ $O(n^2)$

$arr[7] = \{2, 1, 3, 5, 7, 4, 6\}$

$K = 9$

$K = 3$

$K = 10$

Approach: $\{2, 1, 3, 5, 7, 4, 6\}$

Idea

Step: 2 9 N-2

Step: 1 8 N-3

3 7 N-4

→ $O(n^2)$

$arr[7] = \{2, 1, 3, 5, 7, 4, 6\}$

→ Recursion: $arr[n] = \{$

Sorting Left Right

$arr[7] = \{2, 1, 3, 5, 7, 4, 6\}$

Step: $Cal(N, arr[0], 0)$

Try ...

$a = 0$

$b = n$

$a = Temp$

$a = a^b$

$b = a^b$

$a = a^b$

→ $O(1)$

Total = 3

Total = 2

Subarrays: Continuous part of an array is called a subarray

→ [1, 2, 3] should be continuous

Algo: Take single element is a subarrays

Algo: Entire array is also a subarray

Algo: It should always contain element

$arr[n] = \{1, 2, 3, 4, 5, 6\}$

Input: $\{1, 3, 4, 5, 6\}$ → Not continuous

Input: $\{2, 4\}$ →

Input: $\{1, 2, 3\}$ →

Input: $\{1, 3, 4, 5, 6\}$ → Not continuous

5) Given N array elements

→ Take p subarrays

→ Take p subarrays

→ Take p subarrays

Approach: N Elements from $\{1, 2\}$

$\{1, 2\}$ $\{1, 3\}$ $\{1, 4\}$ $\{1, 5\}$ $\{1, 6\}$

$\{2, 3\}$ $\{2, 4\}$ $\{2, 5\}$ $\{2, 6\}$

$\{3, 4\}$ $\{3, 5\}$ $\{3, 6\}$

$\{4, 5\}$ $\{4, 6\}$

$\{5, 6\}$

$\{6\}$

→ Total subarrays = $\frac{(N)(N+1)}{2}$

6) $arr[7]$: { 10, 12, 9, 2, 3, 2 }

$\{10\}$ $\{10, 12\}$ $\{10, 12, 9\}$ $\{10, 12, 9, 2\}$ $\{10, 12, 9, 2, 3\}$ $\{10, 12, 9, 2, 3, 2\}$

$\{12\}$ $\{12, 9\}$ $\{12, 9, 2\}$ $\{12, 9, 2, 3\}$ $\{12, 9, 2, 3, 2\}$

$\{9\}$ $\{9, 2\}$ $\{9, 2, 3\}$ $\{9, 2, 3, 2\}$

$\{2\}$ $\{2, 3\}$ $\{2, 3, 2\}$

$\{3\}$ $\{3, 2\}$

$\{2\}$

→ Total subarrays

N Elements:

	1	2	3	4	5	6	...	N-1	N
0	[1, 1]	[1, 2]	[1, 3]	[1, 4]	[1, 5]	[1, 6]		[1, N-1]	[1, N]
1	[2, 2]	[2, 3]	[2, 4]	[2, 5]	[2, 6]				
2	[3, 3]	[3, 4]	[3, 5]	[3, 6]					
3	[4, 4]	[4, 5]	[4, 6]						
4	[5, 5]	[5, 6]							
5	[6, 6]								
N-1	[N-1, N-1]	[N-1, N]							
N	[N, N]								

→ Total subarrays = $\frac{(N)(N+1)}{2}$

→ Refine step: copy subarrays

$arr[6] = \{ 3, -2, 5, 3, 7, 8 \}$

	0	1	2	3	4	5
$arr[6]$	3	-2	5	3	7	8
	[3, 3]	[3, 2]	[3, 5]	[3, 3]	[3, 7]	[3, 8]
		[1, 2]	[1, 5]	[1, 3]	[1, 7]	[1, 8]
			[2, 5]	[2, 3]	[2, 7]	[2, 8]
				[3, 3]	[3, 7]	[3, 8]
					[4, 7]	[4, 8]
						[5, 8]
						[6, 8]

T=0; 3

3 → 2

3 → 2 → 6

3 → 2 → 6 → 4

3 → 2 → 6 → 4 → 2

→

Given N, print all subarrays

```
T=0; Tc N; T++{
    k=T; Tc N; k++{
        print(arr[T:k])
    }
}
```

→ time

→ space

Given an $arr[7]$. $\{ 3, -2, 5, 3, 7, 8, -1 \}$

Approach:

$arr[N]$ length = N

$Ans[0] = N-1$

$Ans[1] = N-2$

$Ans[2] = N-3$

$Ans[3] = N-4$

$k = N-[N-1]$

$k = N-N+1$

Given N, print all subarrays

```
T=0; Tc N; T++{
    k=T; Tc N; k++{
        print(arr[T:k])
    }
}
```

→ Iterate all elements from T, k

→ Iterate all subarrays

→ Printing all subarrays

→ Time Complexity: $O(N)$ $\Rightarrow$ $O(1)$

→ Given N array elements & $\frac{N}{2}$, K, find subarray $n-2$

Eg: $arr[10] = \{ 3, -2, 5, 3, 7, 8, 9, 4, 1, 3 \}$

$\frac{N}{2} = 5$

$\{n-2\}$

Find Array: $\{ 3, -2, 5, 3, 7, 8, 9, 4, 1, 3 \}$

```
T=0; Tc N; T++{
    while(T<N-2){
        temp = arr[T] & arr[T]
        T++;
    }
}
```

Take $arr[5]$ | Take $arr[N]$

10 Elements | 10 Elements

Dynamic Array:

→ Take 0 size Array

Init Array	Insert Array	Return Array	Capacity	Increase Capacity	Size

Step 1 → Dynamic Array

Take $arr[5]$ → Dynamic array → Create an Array

→ $cap(5)$

→ $sz(0)$

→ Initial Size Array

→ Adding first value element

```
cls : Init(int sz){
    if sz<=0 ->
        sz=1
    base = new int[sz]
}
cls : Insert(int x){
    cls : Insert(int[0], x)
}
cls : Insert(int[] x){
    cls : AddAt(0, x)
}
```

```
T=0; Tc cls.size(); T++{
    print(cls[T])
}
```

→ Ins. array size : 2.0

Output

0, 1, 2, ..., -19

```
void Main(){
    var a = new DynArr(3);
    T=0; Tc N; T++{
        print(a[T])
    }
}
```

Given a Array, Show everything Use

```
void Main(){
    Iterations = data;
    T=0; Tc N; T++{
        data = Insert(arr[T])
    }
    return data;
}
```

→ $O(N)$

//Doubts

1) CP vs Java/Lnd
2) Save back Reminder
3) Comp Language

Int a = 10 ; b = a^2 ; a = a^2 ; b = a^2 ; a = a^2 ; b = a^2 ; a = a^2 : 10 2^2 2^2 2^2 2^2

$arr[5]$: 5 + 4 - 1

→ Scalar Stack

→ Operands & Bracket → Postfix

Infix Exp

→ Postfix

```
T=0; Tc N; T++{
    if(T==N){
        break;
    }
    if(arr[T]==arr[T+1]){
        continue;
    }
    if(arr[T]==arr[T+1]){
        return True
    }
}
```

→ Time Complexity: $O(N)$

```
T=0; Tc N; T++{
    if(T==N){
        break;
    }
    if(arr[T]==arr[T+1]){
        continue;
    }
    if(arr[T]==arr[T+1]){
        return True
    }
}
```

```
T=0; Tc N; T++{
    T2=0; Tc N; T2++{
        if(arr[T]==arr[T2]){
            continue;
        }
        if(arr[T]==arr[T2]){
            return True
        }
    }
}
```

→ $O(N^2)$