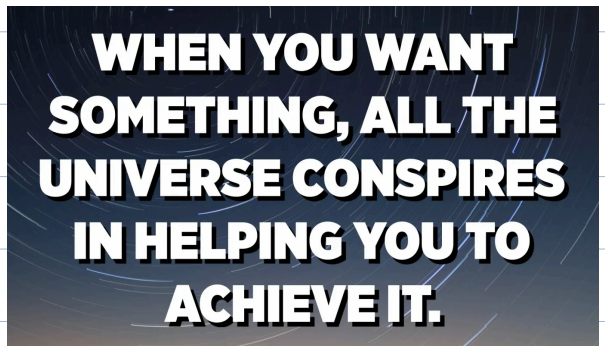
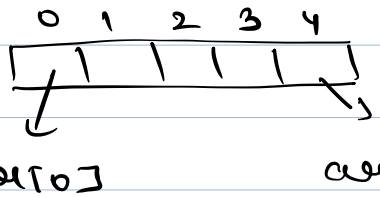
Today's Content

Arrays

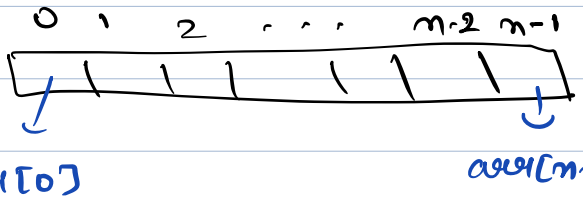
Quote :-



int arr[5];



int arr[n] =




```
void printarr(int arr[]) {
    int n = arr.length;
    for (i=0; i<n; i++) {
        print(arr[i])
    }
}
```

T.C → O(N)

S.C → O(1)

Q1) Given n array elements count no. of elements having 1 element greater than itself.

```
                 0   1   2  3  4  5  6
arr[7] = { -3, -2, 6, 8, 4, 8, 5 3   → 5
           +1  +1  +1    +1    +1

                 0  1  2  3  4  5  6  7
arr[8] = { 2, 3, 10, 7, 3, 2, 10, 8 3   → 6
           +1 +1    +1  +1 +1     +1

                 0  1  2 3
arr[4] = [ 8, 8, 8, 8 ]   → 0
```

Obs :- Largest element of array can't have any element greater than that.

Step :- Iterate & get no. of max elements ↳ c.

Final ans :- Total no. of elements - c

// Pseudo code :-

```
int countGreater(int arr[]) {
    int n = arr.length;                    -> (1)
    int max = arr[0];                      -> (1)
    for (i=1; i<n; i++) {                  -> n-1
        if (arr[i] > max) {
            max = arr[i];
        }
    }

    int freq = 0;
    for (i=0; i<n; i++) {                  -> n
        if (arr[i] == max) {
            freq++
        }
    }

    return n-freq;                         -> 1
}
```

T.C -> 2n+2 -> O(n)

S.C -> O(1).

Todo:- Try to do it in 1 for loop.
Discuss → Next class Doubt session.

p_1 p_2

$-2, 1, 3, 4, 6, 8,$ $k = 7$

Ques) Given N array elements, check if there exists a pair (i,j) such that
arr[i] + arr[j] == k && i != j

0 1 2 3 4 5 6
arr = { 3, -2, 1, 4, 3, 6, 8 }
k = 10 (i = 3, j = 5) return True.

0 1 2 3
arr[] = { 2, 4, -3, 7 }
k = 5 return false.

0 1 2 3
arr[] = { 2, 4, -3, 7 }
k = 8 return false :-

idea 1 :- Make all the pairs and check their sum.

(i,j)

```
bool sum(int[] arr, int k) {
    int n = arr.length;
    for(i=0; i<n; i++) {
        for(j=0; j<n; j++) {
            if (arr[i] + arr[j] == k &&
                i != j)
                return true;
        }
    }
    return false;
}
```

i	j	Total iter
0	0 n-1	n
1	0 n-1	n
2	0 n-1	n
:		:
n-1	0 n-1	n
		n^2

n

T.C → $O(n^2)$.

S.C → $O(1)$.

(i,j) n=4, 0,1,2,3 i → 0 to 2

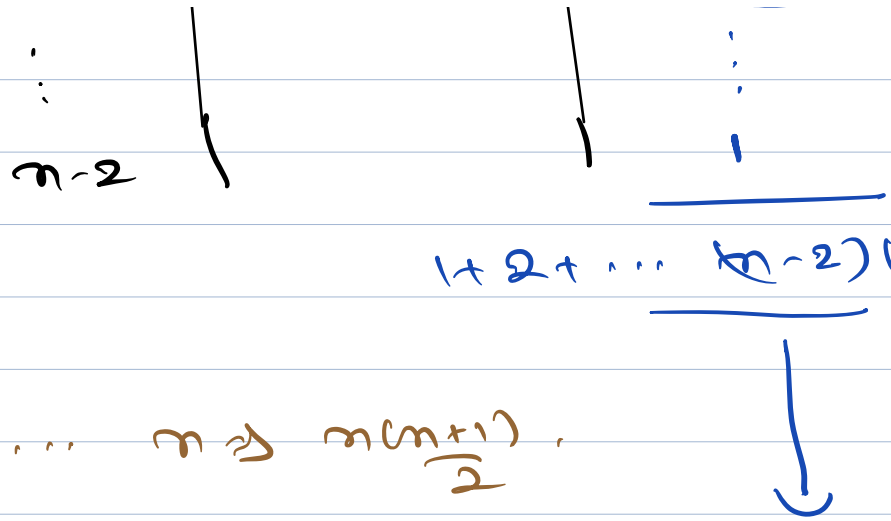
j

i				
	0 0	0 1	0 2	0 3
	1 0	1 1	1 2	1 3
	2 0	2 1	2 2	2 3
	3 0	3 1	3 2	3 3

n → 0 to n-1 i → 0 to n-2

```
bool sum(int[] arr, int k) {
    int n = arr.length;
    for(i=0; i<n-1 ; i++) {
        for(j= i+1 ; j<n; j++) {
            if (arr[i]+ arr[j] == k)
                return true;
        }
    }
    return false;
}
```

i	j	Total iter.
0	1 n-1	n-1
1	2 n-1	n-2
2	3 n-1	n-3



$1+2+\cdots (n-2)(n-1)$

$1+2+\cdots n \Rightarrow \frac{n(n+1)}{2}$.

$\frac{(n-1)(n)}{2}$.

$\Rightarrow \frac{n^2}{2} - \frac{n}{2} \rightarrow O(n^2)$.

S.C $\rightarrow O(1)$.

$\rightarrow$ 3, 3, 4, 5, 3, $k = 6$.

Break 10:01 - 10:11 pm.

$\rightarrow$ Constraint & Space should be constant

Ques) Given an array, reverse entire arr[].

Note:- arr[] itself should change.

```
                0  1  2  3  4  5  6  7
arr[8] = { -1, 4, 7, 6, -2, 7, 8, 10 }
```

↳ { 10, 8, 7, -2, 6, 7, 4, -1 }

Approach :- Two pointer approach :-

```
                    p1
                 p2
                0  1  2  3  4  5  6  7
arr[8] = { -1, 4, 7, 6, -2, 7, 8, 10 }
           10  8  7 -2  6  7  4 -1
```

n =

```
void reverse ( int arr[] ) {

    int p1 = 0; p2 = n-1
    while (p1 < p2) {  → 5 times
        swap ( arr[p1], arr[p2]);
        p1++; p2--;
    }
}
```

100 Bytes

swap:

```
int temp = arr[p1]
arr[p1] = arr[p2]
arr[p2] = temp.
```

T.C → O(n).

S.C → O(1)

```
void reverse ( int arr[] ) {
    int p1 = 0; p2 = n-1
    while ( p1 < p2 ) {          → 5 times
        swap ( arr[p1], arr[p2] );
        p1++; p2--;
    }
}
```

100 Bytes

1 no CVB

int temp = arr[p1]
arr[p1] = arr[p2]
arr[p2] = temp.

T.C → O(n).
S.C → O(1)

```
void reverse ( int arr[] ) {
    int p1 = 0; p2 = n-1
    while ( p1 < p2 ) {          → 5 times
        p1++; p2--;
    }
}
```

100 Bytes

4 B.

int temp = arr[p1]
arr[p1] = arr[p2]
arr[p2] = temp.

T.C → O(n).
S.C → O(1)

Ques) Given N array elements & [si & ei] reverse array from [si, ei]

[si <= ei]

	0	1	2	3	4	5	6	7	8
arr[] = {	-3	4	2	8	7	9	6	2	10 }

p1 = 3, p2 = 7

s=3 void reversepart (int arr[], int s, int e) {

e=7

int p1 = s ; p2 = e

while (p1 < p2) { → 5 times

swap (arr[p1], arr[p2]);

p1++ ; p2--;

}

}

100 bytes

int temp = arr[p1]

arr[p1] = arr[p2]

arr[p2] = temp.

T.C → O(n).

S.C → O(1)

Ques) Given N array elements, Rotate array from last to first by k times → (Google / Amazon)

k = 3,

	0	1	2	3	4	5	6
arr[7] =	3	-2	1	4	6	9	8
k = 1 =>	8	3	-2	1	4	6	9
k = 2 =>	9	8	3	-2	1	4	6
k = 3 =>	6	9	8	3	-2	1	4

	0	1	2	3	4	5	6	7	8
arr[9] =	4	1	6	9	2	14	7	8	3
k=4 =>	14	7	8	3	4	1	6	9	2

k=5

arr => a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12}

f.arr:- a_8 a_9 a_{10} a_{11} a_{12} a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7

1) Reverse the whole array :-

a_{12} a_{11} a_{10} a_9 a_8 a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

reverse reverse.

a_8 a_9 a_{10} a_{11} a_{12} a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7

idea :-

S1 :- Reverse the entire array.

→ reverse part (arr, 0, n-1)

S2 :- Reverse first k elements :-

→ reverse part (arr, 0, k-1);

S3 :- Reverse rest elements

→ reverse part (arr, k, n-1);

		0	1	2	3	4	5
	k = 10; arr[6] =	a_0	a_1	a_2	a_3	a_4	a_5
7 →	k=1;	a_5	a_0	a_1	a_2	a_3	a_4
8 →	k=2;	a_4	a_5	a_0	a_1	a_2	a_3
9 →	k=3;	a_3	a_4	a_5	a_0	a_1	a_2
10 →	k=4;	a_2	a_3	a_4	a_5	a_0	a_1
	k=5;	a_1	a_2	a_3	a_4	a_5	a_0
→	k=6;	a_0	a_1	a_2	a_3	a_4	a_5

k = k % size;

```
void rotateKTimes(int[] arr, int k){
    int n = arr.length;
    k = k % n;
    // reverse the entire array
    reversePart(arr, 0, n-1);
    // reverse first k elements;
    reversePart(arr, 0, k-1);
    // reverse last n-k elements;
    reversePart(arr, k, n-1);
}
```

$n/2$ → reverse entire array

$k/2$ → reverse first k elements

$\frac{n-k}{2}$ → reverse last n-k elements

$$\rightarrow \quad \frac{n}{2} + \frac{k}{2} + \frac{n-k}{2} \quad \Rightarrow \quad \frac{n+k+n-k}{2}$$

$$\frac{2n}{2} \Rightarrow n \quad \therefore \text{T.C} \rightarrow O(n).$$

S.C $\rightarrow O(1)$.