Space Complexity
Intro to Arrays
Questions

No. of iterations → Less → Better

↓

Big O → Time Complexity

Space Complexity → Big O Notation (worst case space)

O( )

↓

memory space taken by your algo

↓

input space + space taken by algo + output space

func (int n) { → 4 B (input space)

int x 4 B
int y 4 B
long z 8 B

} space taken by your algo

Total program space = 20 B

Space Complexity → 16 B → SC: O(1)

1. func (int N) {

```
int arr[10]   40 B
int x          4 B
int y          4 B
long z         8 B
int[] arr = new int[N]   4N B
```

}

int → 4 B long → 8 B

Space = 4N + 56

O(N)

2. func (int N) {

```
int x          4 B
int y          4 B
long z         8 B
int[] arr = new int[N]   4N B
long[][] l = new long[N][N]   8N^2
```

}

int → 4 B lon → 8 B

Total = 8N^2 + 4N + 16

SC: O(N^2)

3.

```
int maxArr(int arr[], int N){
    int ans = arr[0]
    for(i from 1 to N-1){
        ans = max(ans, arr[i])
    }
    return ans
}
```

UB

UB

space = 8 B

SC : O(1)

Auxiliary space → extra space taken by algo

Intro to arrays → list of same type of elements

int marks

int score[5] → size

score → name of array

int → data type

ind

0	1	2	3	4
99	96	40	89	31

score

arr → index

score[0]

int score[N]

ind → 0 to N-1

int arr[N]

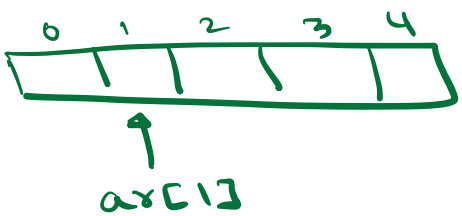
1 ind → O(1)
N ind → N × O(1)
= O(N)

```
for (ind=0 ; ind < N ; ind++) {
    print (arr[ind])
}
```

arr[N] → segmentation fault
→ index out of bound

Time complexity to access ith index in array of size N

arr[5]



TC: O(1)

int arr[5] = <5, -4, 8, 9, 10>
0 1 2 3 4
1st 2nd 3rd 4th 5th

arr[0] + arr[4]

1. Given an array of size N, find max element

N = 5 arr = <0, 1, 3, 5, 7> ans = 9

Max ele → <2, 5, 1, 4, 8, 0, 8, 1, 3, 8>
ans = 8

// int arr[N]

```
int maxele = arr[0]
for (i = 1 ; i < N ; i++) {
    if (arr[i] > maxele) {
        maxele = arr[i]
    }
}
```

A = <-3, -1, -2> (0, 1, 2)

maxele = -3 -1

N = 6
i < 6

A = <5, 10, 15, -1, 2, 3> (0, 1, 2, 3, 4, 5)

maxele = 5 10 15

int maxele = -∞ → INT_MIN / Integer.MIN_VALUE

```
for (i = 0 ; i < N ; i++) {
    if (arr[i] > maxele) {
        maxele = arr[i]
    }
}
```

N = 6
i < 6

A = <5, 10, 15, -1, 2, 3> (0, 1, 2, 3, 4, 5)

maxele = -∞ 5 10 15

<4, 6, 37

made = 0

Given an array 'arr' of size 'N'. Check if there exists a pair (i,j) such that arr[i] + arr[j] == k and i!=j.

TestCase 1:

Input:

```
N = 5
arr = {9,1,3,5,9}
K = 12
```

0 1 2 3 4

(2, 4)

3 + 9 = 12 True

(0, 2)

9 + 3 = 12

Given ar[5] = {3 5 2 7 3} check if there exists pair [i,j] such that i!=j and ar[i] + ar[j] = 6.

0 1 2 3 4

0, 4

3 + 3 = 6 True

Given ar[3] = {4 2 7} check if there exists pair [i,j] such that i!=j and ar[i] + ar[j] = 8.

0 1 2

0, 0 X i != j

4 + 4 = 8 False

<3, 5, 2, 7, 37 (K)

Brute Force: check all pairs

pair → i, j

TC : $O(N^2)$
SC : $O(1)$

```
for (i=0 ; i<N ; i++) {
    for (j=0 ; j<N ; j++) {        0 to N-1
        // fixing with (i,j)

        if (i!=j && arr[i] + arr[j] == K)
            return true
    }
}
return false
```

$N = 5$, $K = 8$

0	1	2	3	4
3	5	2	7	37

i	j	arr[0]+arr[0]
0	0	3 + 3 == K

(i, j)

i,j				
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

i	j
0	1→4
1	2→4
2	3→4
3	4→4

i	j
0 ,	1
3 + 5	

i	j
1 ,	0
5 + 3	

Optimized → Either go to upper Δ pairs or go to lower Δ pairs

i j

i+1 → N-1

SC : O(1)

```
for (i=0 ; i<N ; i++) {
    for (j=i+1 ; j<N ; j++) {
        // fixing diff (i,j)
        if (arr[i] + arr[j] == K)
            return true
    }
}
```

0 to N-2

N = 5

i = 4 j = 5

0 1 2 3 (i)

(j)

i = N-1 j = N loop breaks

N + N

i	j [i+1 → N-1]	
0	[1 → N-1]	N-1
1	[2 → N-1]	N-2
2	[3 → N-1]	N-3
⋮		+ ⋮
N-2	[N-1 → N-1]	1
N-1	[N] X	

$1 + 2 + \ldots\ldots N-1 = \frac{(N-1)(N)}{2}$

$\frac{(N-1)(N)}{2} = \frac{N^2}{2} - \frac{N}{2}$

TC: $O(N^2)$

SC: $O(1)$

$1 + 2 + \ldots N = \frac{N(N+1)}{2}$

10:50

3. Given an arr, reverse it

i, j → index

	0	1	2	3	4
arr[5] =	~~6~~ 17	~~16~~ 20	15	~~20~~ 10	~~17~~ 6

reverse → 17 20 15 10 6

i: 0 → 1 (+1)

j: 4 → 3 (-1)

i: 2 → 2 (+1)

j: 2 → 2 (-1)

i == j

stop

	0	1	2	3	4	5
arr[6] =	~~6~~ 5	~~16~~ 17	~~15~~ 20	~~26~~ 15	~~17~~ 10	~~5~~ 6
rev →	5	17	20	15	10	6

j i

i	j
0	5
1	4
2	3
3	2

i > j stop

N/2 ite

```
i = 0 , j = N-1
while (i < j) {
    temp = arr[i]
    arr[i] = arr[j]
    arr[j] = temp
    i++      j--
}
```

i ≥ j stop

TC: O(N)
SC: O(1)

i j

~~1~~ 2 2

arr[i] = arr[j]

2 dc → 1 itr 1 dc → 1/2

N dc → N/2 itr

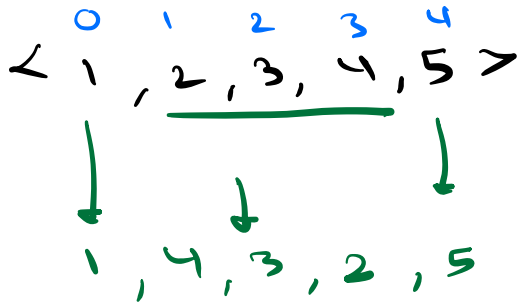
4. Given an arr, reverse part of arr

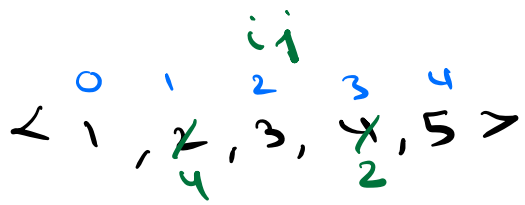
l → r

N = 5

l = 1

r = 3





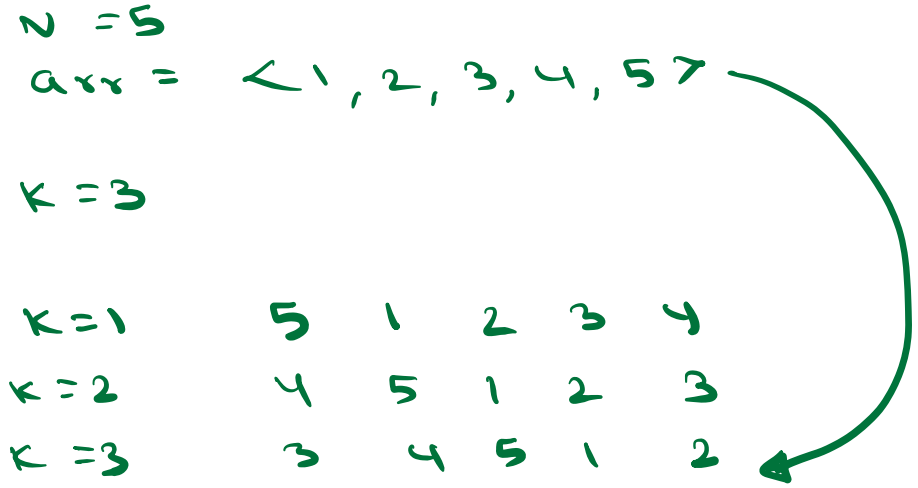

```
void reverse(int arr[], int n, int l, int r) {
    i = l , j = r
    while (i < j) {
        temp = arr[i]
        arr[i] = arr[j]
        arr[j] = temp
        i++      j--
    }
}
```

WC l → 0
    r → n-1

TC: O(N)

SC: O(1)

5. Given an array of size N, rotate the array from R->L K times

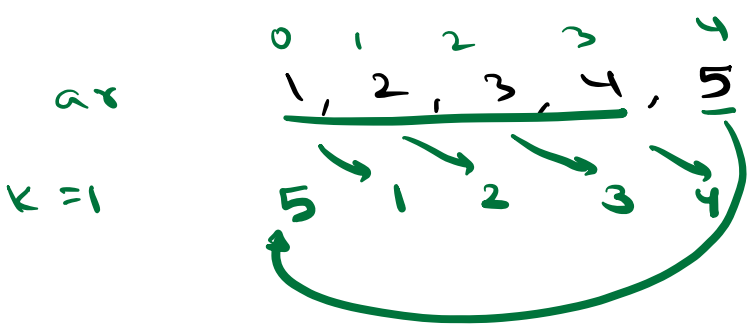


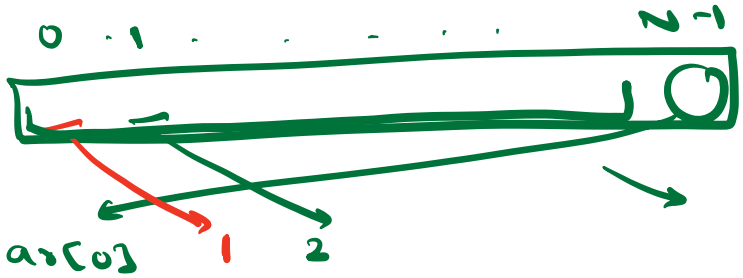
BF : Do the rotation process K times

```
for (cnt=1 ; cnt ≤ K ; cnt++) {
    temp = ar[N-1]
    for (i = N-2; i ≥ 0 ; i--) {
        ar[i+1] = ar[i]
    }
    ar[0] = temp
}
```

K * (N-1)

TC: O(NK)
SC: O(1)





0 1 2
1 2 3

N = 8

7, 6, 5, 8, 9, 2, 5, 10

K = 3

2 5 10 7 6 5 8 9

last 3 → first 3
first 5 → last 5

ar[10] : a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9

K = 3 a_7 a_8 a_9 a_0 a_1 a_2 a_3 a_4 a_5 a_6

---

a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9

↓ Reverse ar

a_9 a_8 a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

1

↓ reverse first K elements

a_7 a_8 a_9 a_6 a_5 a_4 a_3 a_2 a_1 a_0

↓ reverse last N-K elements

a_7 a_8 a_9 a_0 a_1 a_2 a_3 a_4 a_5 a_6

```
void rotate(int ar[], int N, int k){
    k = k % N
    reverse(ar, 0, N-1)
    reverse(ar, 0, k-1)   → first k elements
    reverse(ar, k, N-1)   → rem elements
}
```

TC : O(N)

SC : O(1)

k=0	1,2,3,4	4	8	% 4 = 0
k=1	4,1,2,3	5	9	% 4 = 1
k=2	3,4,1,2	6	10	% 4 = 2
k=3	2,3,4,1	7	11	% 4 = 3
k=4	1,2,3,4			

(5)

4 → 0 → 3

k % n

Dynamic arrays → don't have to declare size

int ar[5] → size is fixed

can't extend space

1. add / append elements
2. resized

↓

access in ind → ar[i] O(1)

Java	C++	Python	C#	JS
ArrayList	vector	List	↓	Array
			Array list	

1. Declare a dynamic array
2. Initialize dynamic array

list<int> l

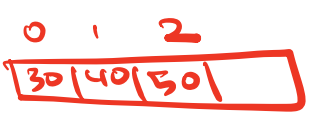
↓

integers

3. Insert elements

l.add(30)
l.add(40)
l.add(50)

→ at the last



(4) Iterate in dynamic list

GFG, Leetcode, Hackerearth

```
for (i=0 ; i<N ; i++) {
    for (j=0 ; j<N ; j++) {
        ...
    }
}

for (k=0; k<N ; k++) {
}
```

```
for( i=0 ; i <2^N ; i++) {
    j=i
    while (j>0){
        j--
    }
}
```

i	j	iter
0	0	0 +
1	1	1 +
2	[2 1]	2
3	[3 1]	
		(
		(
		(+
2^N-1		2^N-1

$1 + 2 + \dots 2^N-1$

$$\frac{2^{N-1}(2^N)}{2} = \frac{4^N - 2^N}{2}$$

4^N