ARRAYS - 1

Good Evening

Agenda for Today

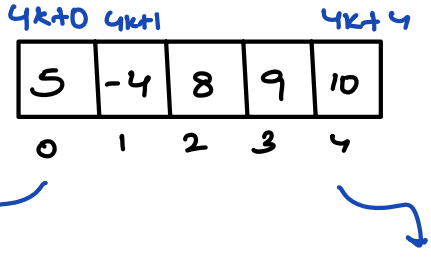
Basics of Array

01. Greater element
02. Check pair
03. Reverse array
04. Rotate an array

ARRAYS BASICS

Array → Collection of same data
↳ Contigous memory alocation

```
int ar[5] =
```



ar → 4K

1st ele + 5th ele = ar[0] + ar[4]

ar[4] ↓

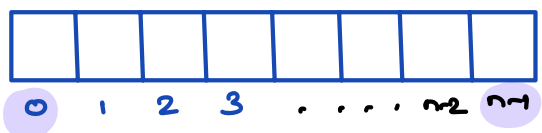
4000 + 4 x 4 byte

= 4000 + 16

= 4016

To access ar[i] in an array → O(1)

```
int ar[n] =
```



ar[-1], ar[n], ar[n+1] → Error

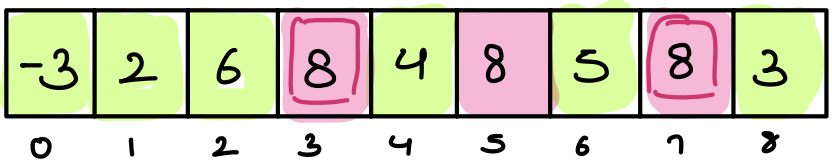
Print Array

passing array as parameter

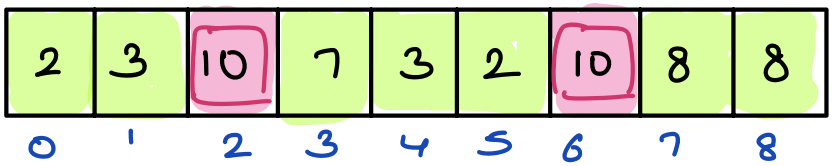
```
void print (int [] ar)
{
    int n = ar.length;
    for (int i=0; i<n; i++){
        print (ar[i]);
    }
}
```

TC : O(n)
SC : O(1)

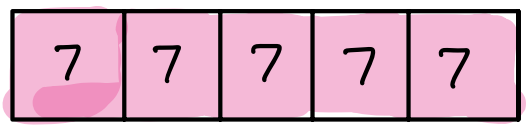
01. Given N elements of array. Count the no. of ele having atleast one element greater than itself.

ar[9] = 

Ans = 6

ar[9] = 

Ans = 7

ar[ ] = 

Ans = 0

Observation → Max of the array is not going to contribute

Steps → 01 Get the max ele

02 Count the freq of max ele

03. Ans = Total no. of ele - count :

```
int countgreater(int[] ar, int n)
{
    // max of the arr
    int maxi = ar[0];
    for (i=1; i<n; i++) {
        if (ar[i] > maxi) {
            maxi = ar[i];
        }
    }

    // count the freq of max ele
    for (i=0; i<n; i++) {
        if (ar[i] == maxi)
            c = c+1;
    }

    return n-c;
}
```

TC = O(n)

SC = O(1)

TODO → Find the maximum ele & the frequency of max ele in just one loop

02. Given N array elements. Check (True/False) if there exists a pair (i, j) such that ar[i] + ar[j] == K & i != j

Note :- i & j are indexes, K is some value

ar[ ] = | 3 | -2 | 1 | 4 | 3 | 6 | 8 |
0 1 2 3 4 5 6

K = 10 → True

ar[ ] = | 2 | 4 | -3 | 7 |
0 1 2 3

K = 4 → True

ar[ ] = | -2 | 3 | 4 |
0 1 2

K = 6 → False

ar[1] + ar[1]
3 + 3 = 6 i != j

arr[ ] = | 3 | 5 | 2 | 7 | 3 |
0 1 2 3 4

K = 6 → True

Simplest / Brute force

↳ Check for all the pairs, get the sum & compare if sum == K

arr[ ] = | | | | |
0 1 2 3

All pairs

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)
(0,3)	(1,3)	(2,3)	(3,3)

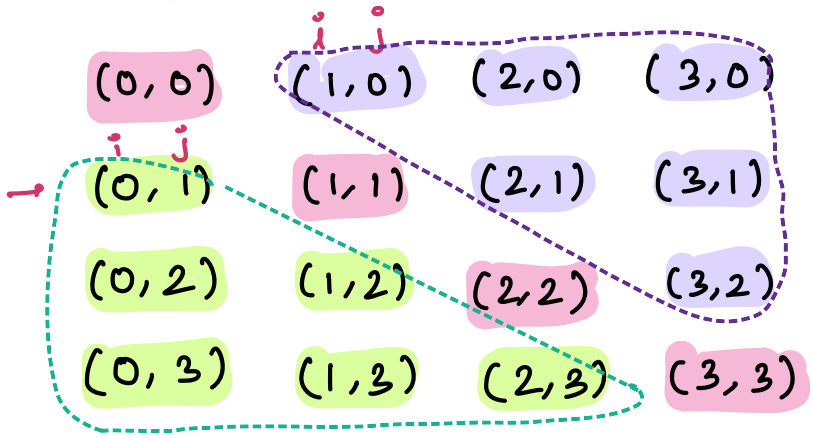
Idea → 1

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

TC = O(n²)
SC = O(1)

Idea 2 → Only look for the ans in upper half or the lower half

All pairs



i	j			
1	0	=	ar[1] +	ar[0]
0	1	=	ar[0] +	ar[1]

} same sum → duplicates pair

```
boolean sum (int [] ar, int k)
{
    for (i=1 ; i<n; i++)
    {
        for (j=0; j<i; j++)
        {
            if (ar[i] + ar[j] == k)
                return true;
        }
    }
    return false;
}
```

i	j	no of iter
1	[0 0]	1
2	[0 1]	2
3	[0 2]	3
.	.	.
.	.	.
.	.	.
n-1	[0 n-2]	n-1

Total iterations $= \frac{(n-1)*(n-1+1)}{2}$

$= \frac{(n-1)*n}{2}$

$= \frac{n^2-n}{2} \simeq O(n^2)$

All idea

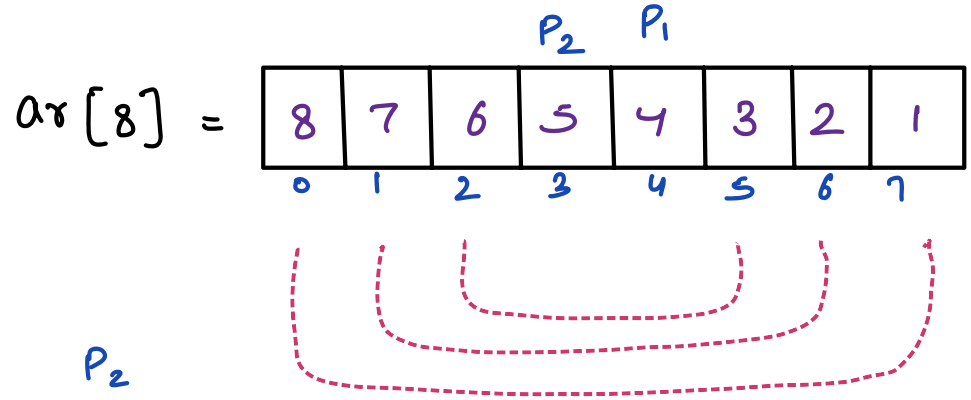
01. All pairs → $O(n^2)$
02. Optimised pair → $O(n^2)$
03. HashMap → $O(n)$
04. Sort + BS → $O(n \log n)$
05. Sort + 2 pointer → $O(n \log n)$

} Future

03 Given an array, **Reverse** the entire array

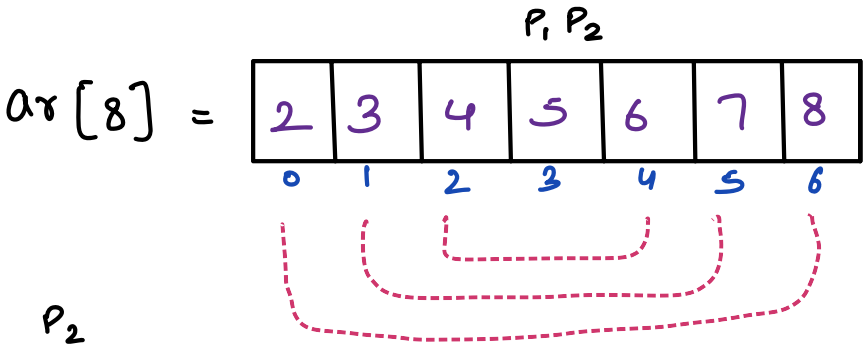
Note :- Change the given array

Expected S.C = O(1)



P_1	P_2	
0	7	swap
1	6	swap
2	5	swap
3	4	swap
4	3	→ No swap

i > j → stop



P_1	P_2	
0	6	swap
1	5	swap
2	4	swap
3	3	⟶ No

$i == j$ → No swapping

Breaking point $i == j$, $i > j$ } $i \geq j$

⇒ Swap until unless $i < j$

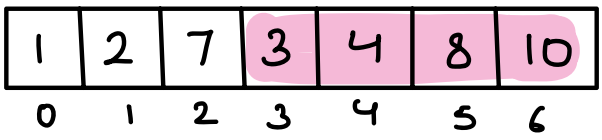
```
void reverse (int [] ar)
    int n = ar.length
    P1 = 0, P2 = n-1
    while (P1 < P2) {
        swap (P1, P2) ⟶ { t = ar[P1]
                          ar[P1] = ar[P2]
                          ar[P2] = t;
        P1++;
        P2--;
    }
}
```

TC = O(n)

SC = O(1)

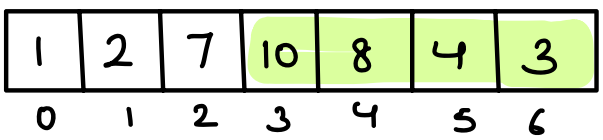
04. Given an array of n elements. Two indicies si & ei are also given. Reverse the array from si to ei

ar[ ] =



si = 3

ei = 6




```
void reversepart(int [] ar, int si, int ei)
    int n = ar.length
    P1 = si, P2 = ei
    while (P1 < P2) {
        swap(P1, P2)  ⟶  { t = ar[P1]
                           ar[P1] = ar[P2]
                           ar[P2] = t;
        P1++;
        P2--;
    }
}
```

TC = O(n)

SC = O(1)

05. Given ar[N] elements, Rotate this array from last to first (anticlockwise) for K no. of times

ar[7] = { 3 -2 1 4 9 6 7 } K=3

0 1 2 3 4 5 6

7-3=4

K=1 = { 7 3 -2 1 4 9 6 }

K=2 = { 6 7 3 -2 1 4 9 }

K=3 = { 9 6 7 3 -2 1 4 }

ar[9] = { 1 2 7 8 4 5 6 7 3 } K=4

K=1 = { 3 1 2 7 8 4 5 6 7 }

K=2 = { 7 3 1 2 7 8 4 5 6 }

K=3 = { 6 7 3 1 2 7 8 4 5 }

K=4 = { 5 6 7 3 1 2 7 8 4 }

Ans → { 5 6 7 3 1 2 7 8 4 }

0 1 2 3 4 5 6 7 8

ar[9] = { 1 2 7 8 4 5 6 7 3 } K=4

as =

5	6	7	3	1	2	7	8	4
0	1	2	3	4	5	6	7	8

With Extra Space

```
for (i=0; i<n-k; i++) {          i=[0  n-k-1]
    ans[i+k] = ar[i];
}
p=0
for (i=n-k; i<n; i++) {
    ans[p] = ar[i];
    p++;
}
```

Expected SC = O(1)

ar[10] = { a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 }

K = 4

// Reverse {0, n-1}

ar[10] = { a_9 a_8 a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 }

0 1 2 3 4 5 6 7 8 9

reverse first k ele

reverse these ele

a_6 a_7 a_8 a_9 a_0 a_1 a_2 a_3 a_4 a_5

After rotation { a_6 a_7 a_8 a_9 a_0 a_1 a_2 a_3 a_4 a_5 }

Steps

01. Reverse the complete array = {0, n-1}
02. Reverse the first half = {0 k-1}
03. Reverse the second half = {k n-1}

Fail for cases $k > n$

$n = 6$ $k = 8$ ⟶ $k = k \% n$ → $8 \% 6 = 2$

$k = 0$ ⟶

or [4] = a_0 a_1 a_2 a_3

$k = 1$ = a_3 a_0 a_1 a_2

$k = 2$ → a_2 a_3 a_0 a_1

$k = 3$ → a_1 a_2 a_3 a_0

$k = 4$ → a_0 a_1 a_2 a_3

$k = 5$ → a_3 a_0 a_1 a_2

$k = 6$ → a_2 a_3 a_0 a_1

$k = 7$ → a_1 a_2 a_3 a_0

$k = 8$ → a_0 a_1 a_2 a_3

$0 \rightarrow 4 \rightarrow 8$

$1 \rightarrow 5 \rightarrow 9$

$2 \rightarrow 6 \rightarrow 10$

$k = k \% n$

↓

arr.length

$k = 7$

```
void rotatearr (int[] arr, int k)
{
    int n = arr.length;

    k = k%n

    reversepart (arr, 0, n-1)  ⟶ n/2

    reversepart (arr, 0, k-1)  ⟶ k/2

    reversepart (arr, k, n-1)  ⟶ (n-k)/2
}
```

$$\text{Total iterations} = \frac{n}{2} + \frac{k}{2} + \frac{(n-k)}{2}$$

$$= \frac{n+k+(n-k)}{2}$$

$$= n$$

$$TC = O(n)$$

$$SC = O(1)$$

```
for (i=0 ; i<n ; i++)
|
|   for (j=i-1 ; j≥0 ; j++) {
|   |                          } Infinite no. of times
|   }
}
```

i	j	Iteration
0	0-1	0
	$-1 \geq 0$	

1 $0 \geq 0$

$1 \geq 0$

$2 \geq 0$

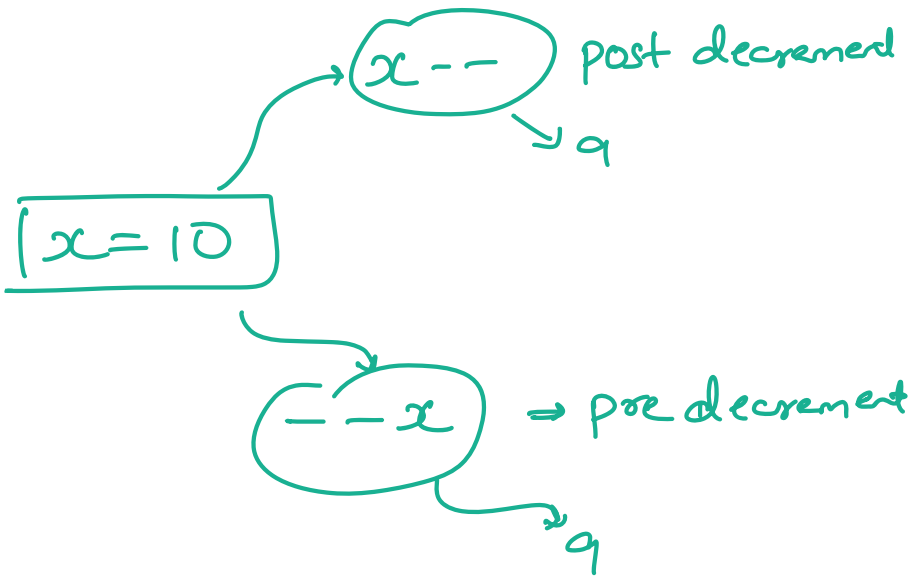
$3 \geq 0$

```
int i=1
while (i<n)
{
    int x=i
    while(x--)>0){
    }
}
```

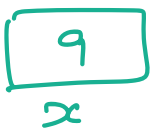
```
while(x--)>0){
}
```

=

```
while(x>0){
    x--;
}
```




```
int x=10
int y=x--;
```



y 10

```
int x = 7
```

x | 6 |

```
int y = --x;
```

y | 6 |

2^x → polynomial

e^x → exponential

```
for (i=0 ; i<n ; i++)
{
    for (j=i ; j<n; j++){
        break;
    }
}
```

O(1)

i	j	iteration
0	0<n	1

```
for (i=0; i<2^n; i++)
    for(j=i; j>0; j--){
        |
        3
    |
    3
```

i	j	iteration
0	[0 -1]	0
1	[1 1]	1
2	[2 1]	2
3	[3 1]	3
⋮		
2^n-1	$[2^n-1\ 1]$	2^n-1

$$1 + 2 + 3 + \cdots + 2^n - 1$$

$$x = 2^n - 1$$

$$1 + 2 + 3 + \cdots + x = \frac{x(x+1)}{2} = \frac{x^2 + x}{2}$$

$$\simeq O(x^2)$$

$$\simeq O(2^n - 1)^2$$

$$\simeq O((2^n)^2 - (1)^2 + 2 + 2^n)$$

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

$$= O(2^{2n}) \simeq O(4^n)$$