Arrays

Time complexity

Space complexity

(Auxiliary space)

(TLE)

Data Structure

- Array
- Linked lists
- Graphs
- Trees
- Stacks
- Queues

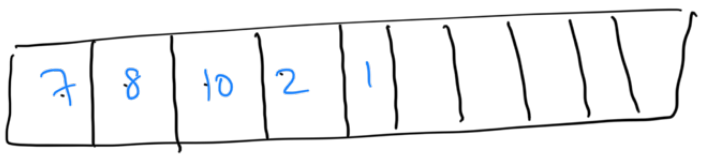
ARRAY

x 10

Array [10]

int

shj

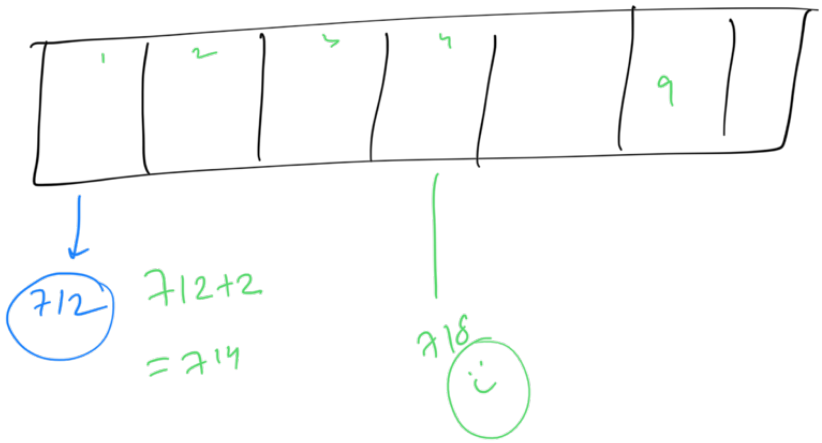


= Size of Array ^ Size of

Start Address

int arr[10]

int → 2 bytes

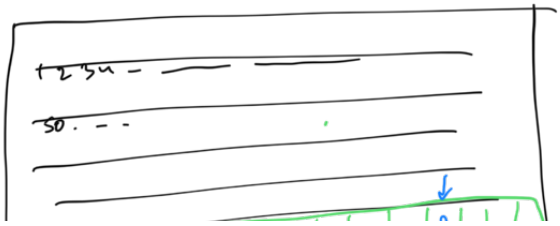


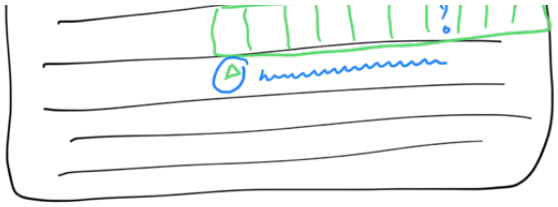
" random access"

Access any element

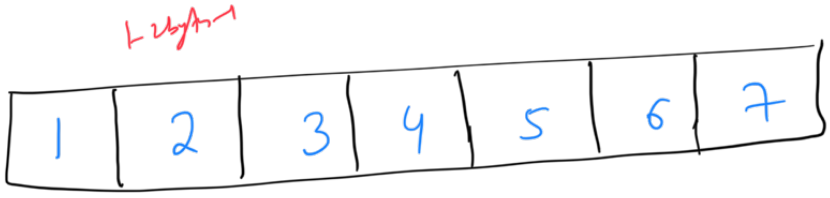
= O(1)

= Constant





Array
— Constant time random access



712

1st → 712

2nd → 714

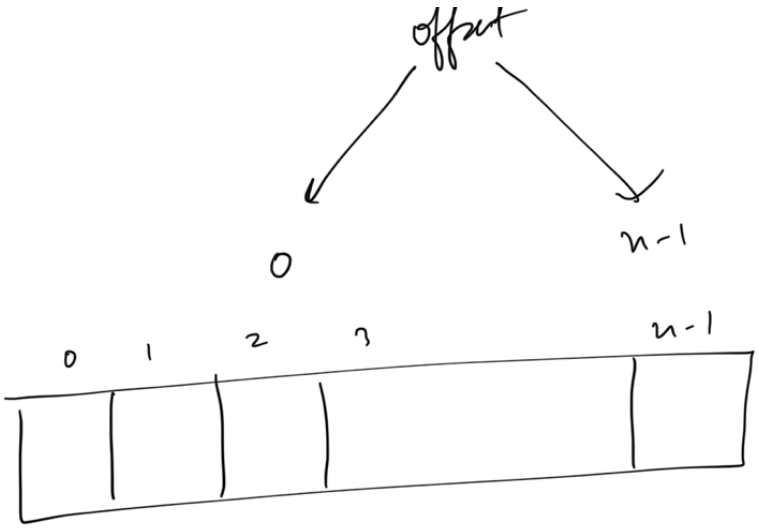
= 712 + (2)x(1

= Base Add + Singl
x
(i-i

3rd → 716

= 712 + 2x2

10th → 712 + 2x9



arr[0]

---

arr[5] = {-8, 10, 9, 4, 5}

1st and 5th

arr[0]

arr[4]

print (arr[0] + arr[4])

O(1)

O(1) + O(1)

= O(1)
T.C

---

(0) Write a piece of code for printing all elements of the array

printArray (int arr[N])

T.C = O(n)

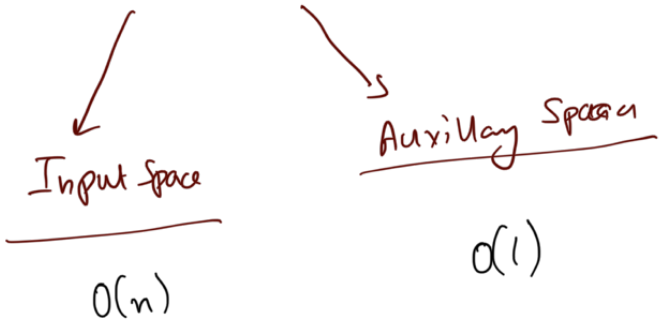
{

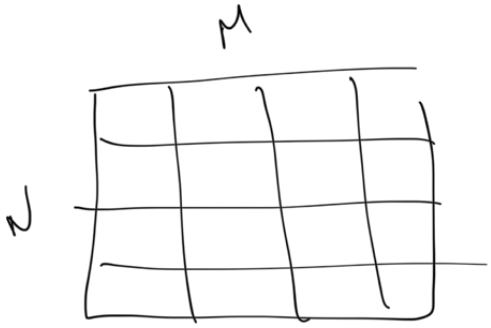
i ≤ n-1
||
for (i=0, i<n, i++)
print (arr[i])

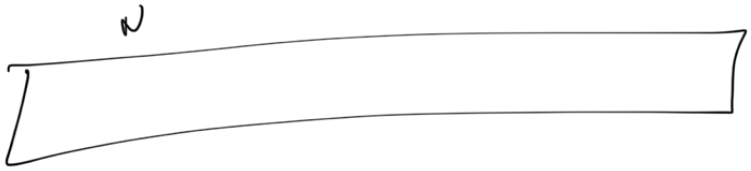
}

S.C

O(n) + O(1) = O(n)

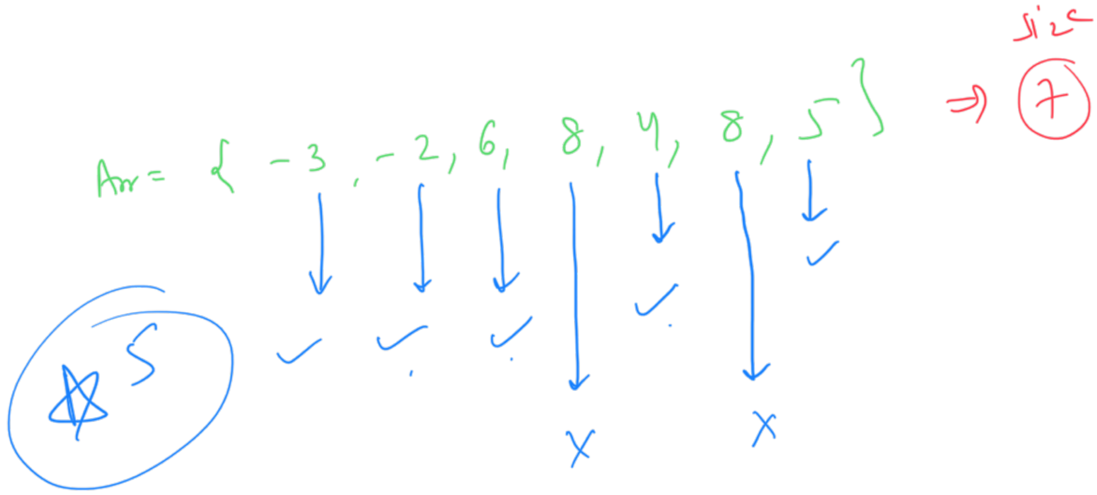






Q Given an array of size N. Count the

number of elements that have atleast
1 element greater than itself.



Obs 1 : Every element in the collection WILL HAVE some element greater than it but for the greatest element(s)

Arr[N]
unique elements

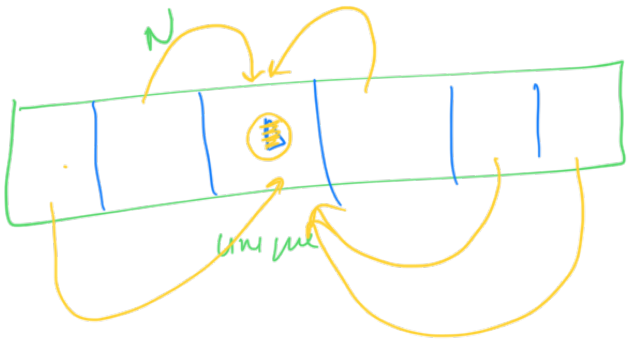
N-1

12, 5, 7, 10, 15

(A) about a + uniqueness

N-1

unique



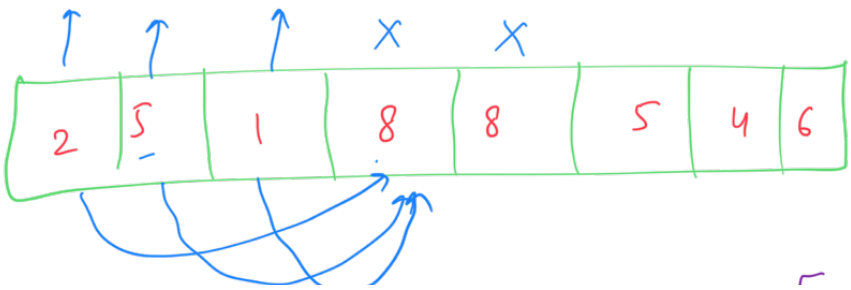
N-1

N-1

O(1)

---

no uniqueness constraint



8 ← (1) Find the biggest elem →

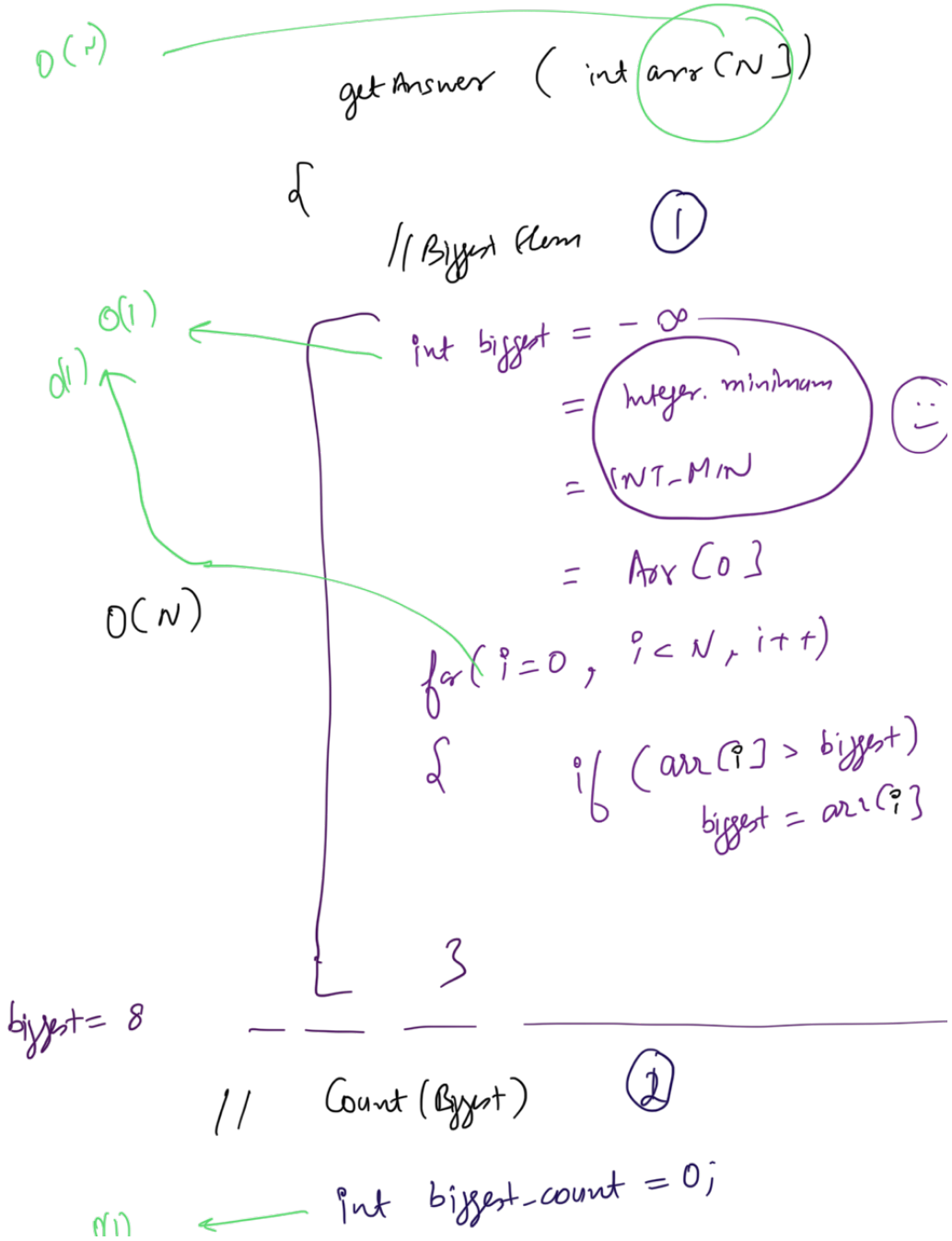
Count(8) = 2 ← (2) Count (E)

← (3) Return N − count(E)

N − 2

8 − 2

= 6



O(N) for (i=0; i<N; i++)

{

O(1) if (arr[i] == biggest)

b_c = biggest_count + 1

}

b_c = 2

O(1) Return N - biggest_count ;

}

T.C = O(N) + O(N) + O(1)

= O(N)

T.S.C

S.C = O(N)

Input O(N)

Auxiliary O(1)

8:59a

9:01a

Q

Arr [N]

number K

Return True if two indexes i, j

s.t a[i] + a[j] == k

i != j

I/P 3, -2, 1, 4, 3, 6, 8

k = 10

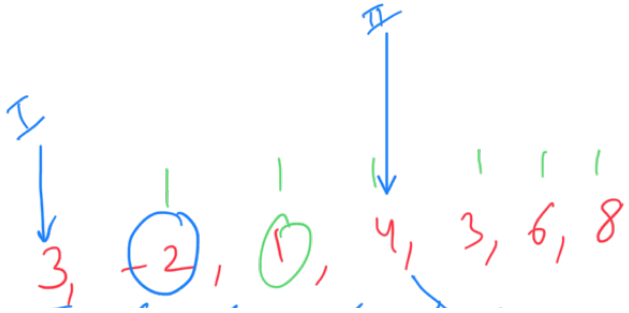
k = 15 ☆ False

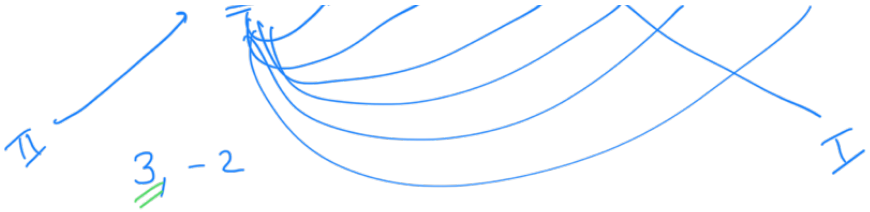
4 + 6 = 10

k = 5

4 + 1 or 1 + 4 ☆ True

☆ True





3, -2
3, 1
3, 4
3, 3
3, 6
3, 8

-2, 3
-2, 1
-2, 4
-2, 3
-2, 6
-2, 8

```
for (i=0, i<n, i++)
{
    int first = arr[i];
    for (j=0, j<n, j++)
    {
        int second = arr[j];
        if (i != j) {
            if (arr[i] + arr[j] == k)
                ☆☆☆ Return TRUE
        }
    }
}
```

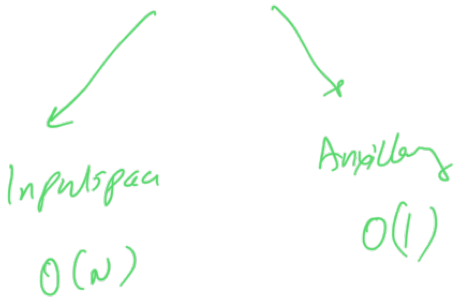
O(N) X O(N)

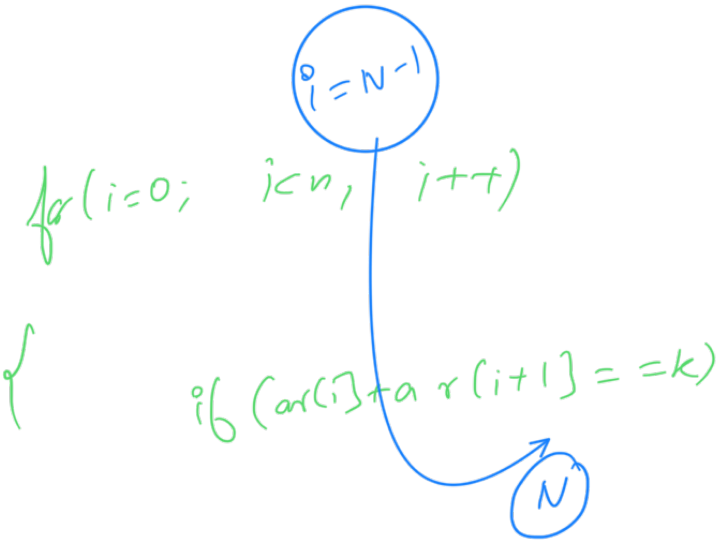
}

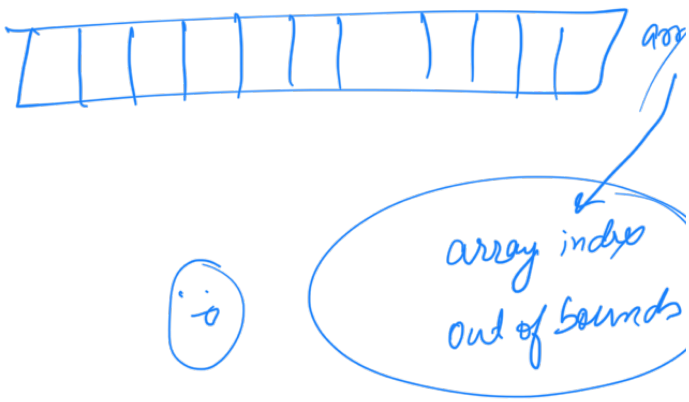
return FALSE ☆ ☆ ☆ ☆

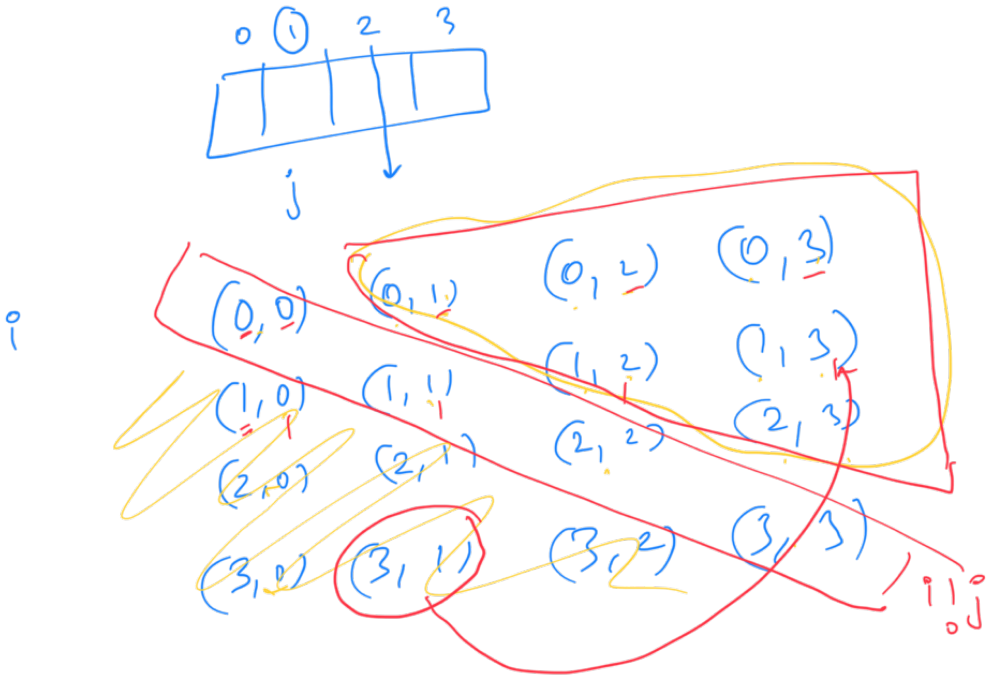
T.C = $O(N^2)$

S.C









O(N) ← for(i=0 ; i<N ; i++)

{

int first = arr(i)

for(j=i+1 ; j<N ; j++

{

if(arr(i) + arr(j)=

return True;

}

}

return false;

T.C = O(N²)

S.C =

O(N) O(1)

☺

i=0

i=1

i=N-2

inner loop

N-1

N-2

2

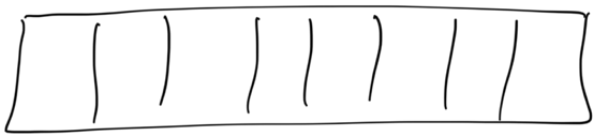
$\frac{N(N-1)}{2}$

$= N^2$

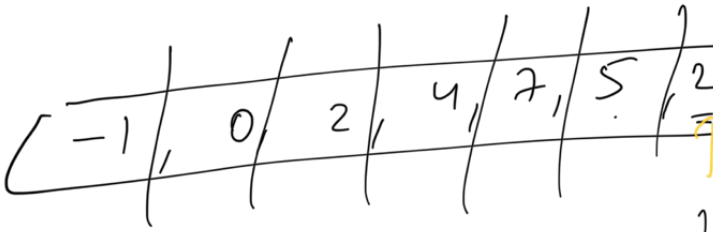
$= O(N^2)$ ☺

Nested Loops

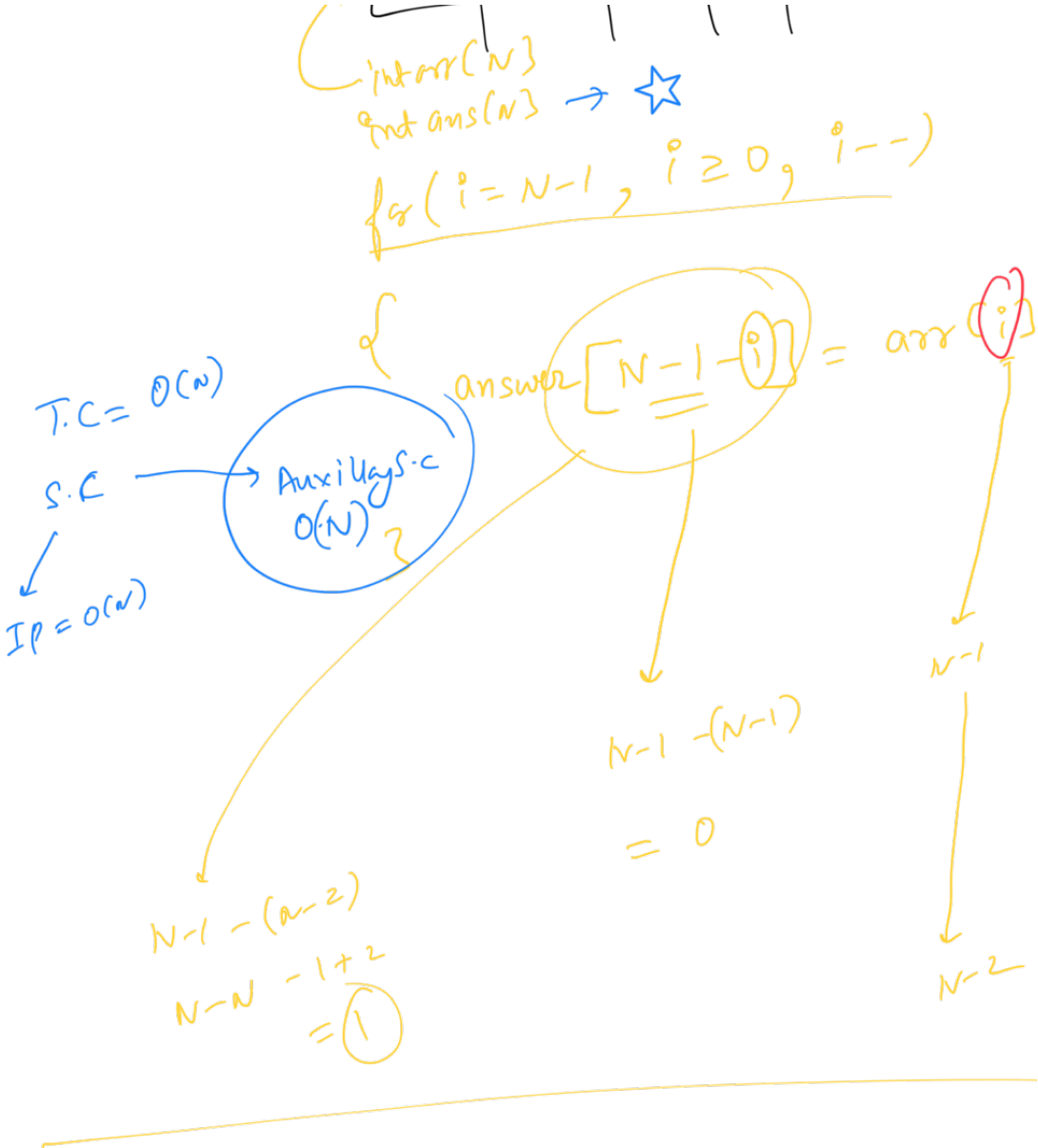
@ Microsoft Amazon



I/P



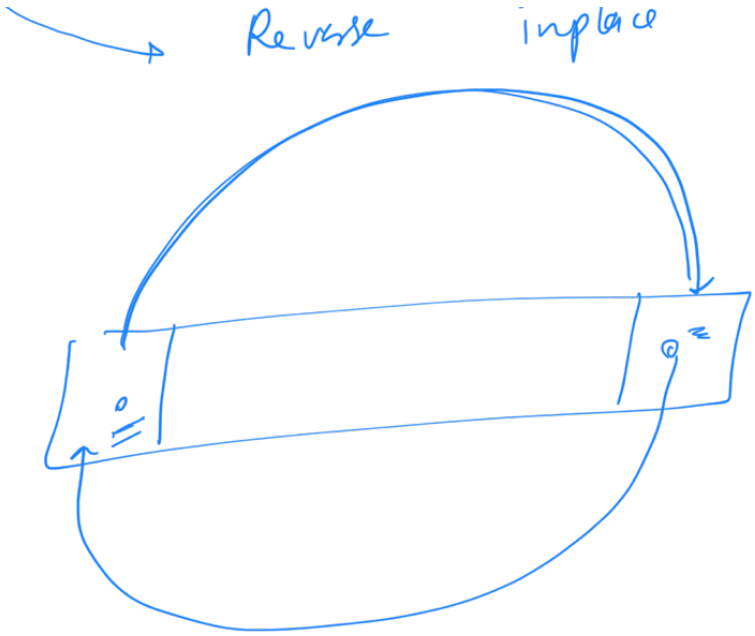
O/P

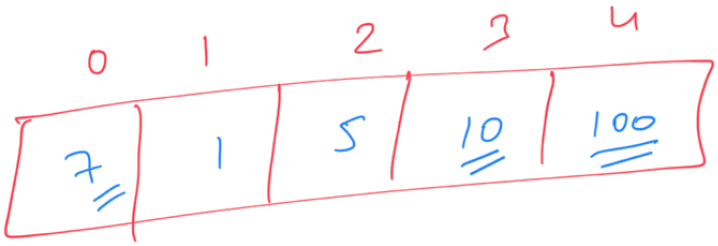


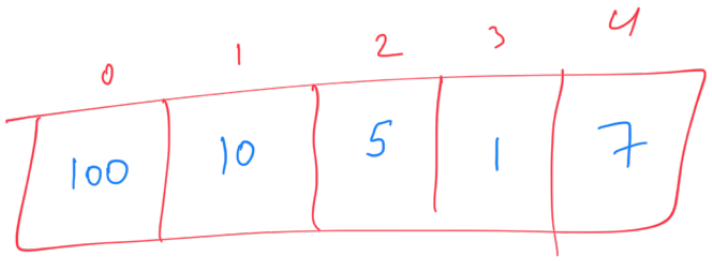
Reversed Array

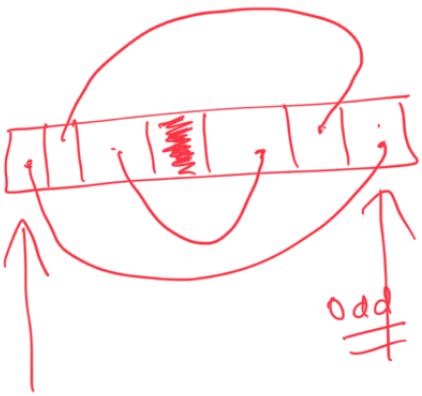
Reverse the array without any O(n) auxiliary space

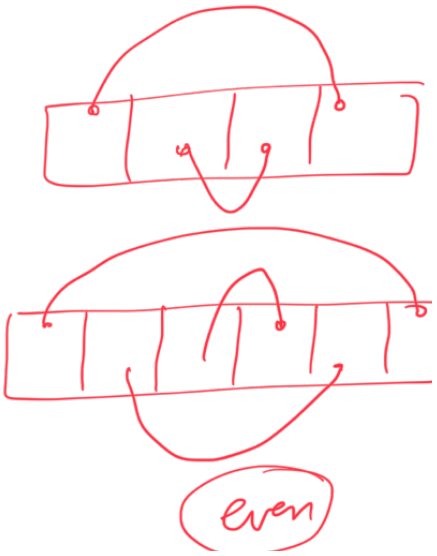
:-D

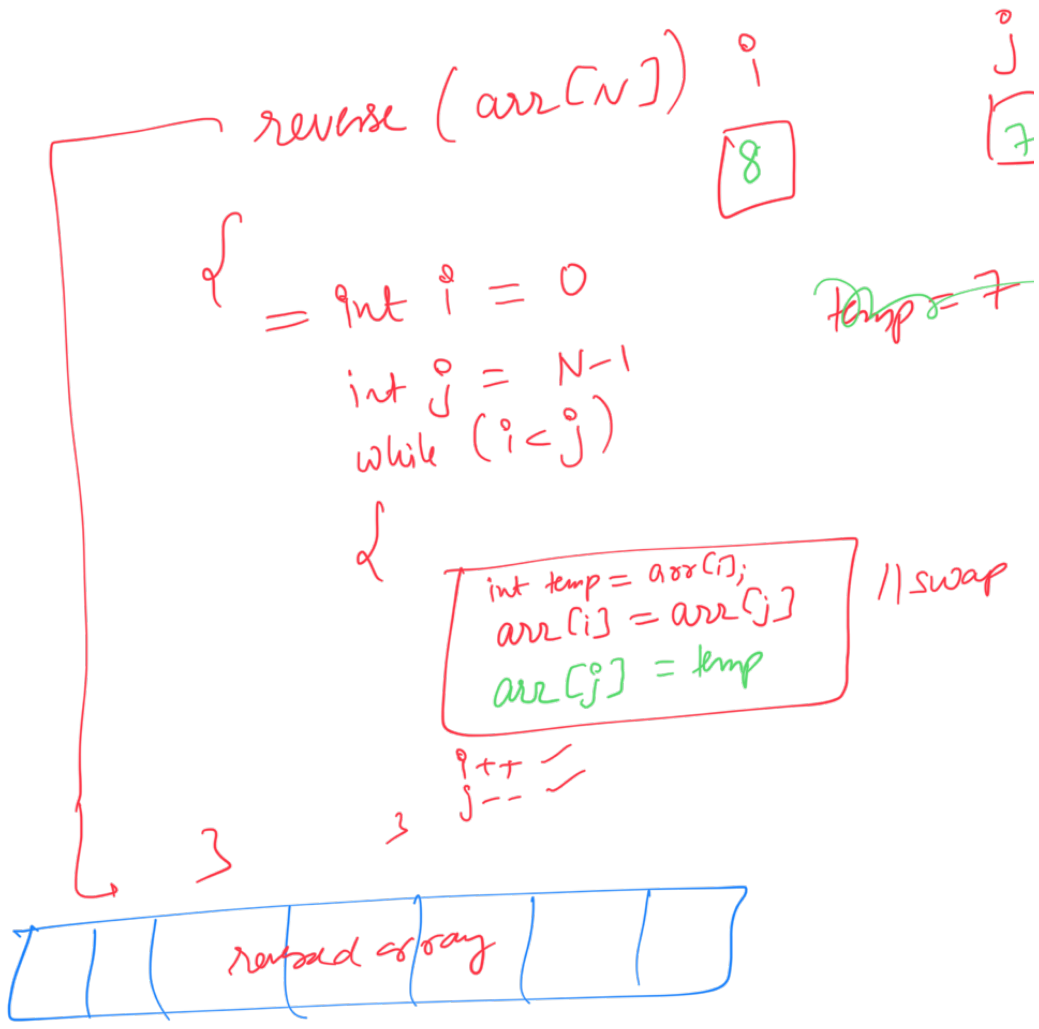

Reverse
inplace

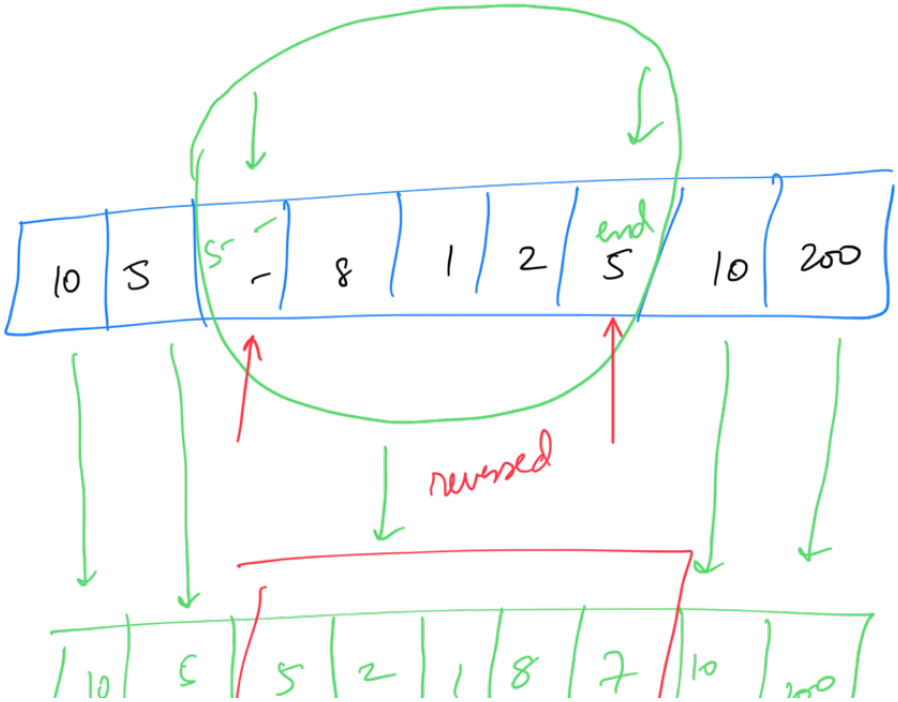

0
1
2
3
4
7
1
5
10
100


0
1
2
3
4
100
10
5
1
7


odd


even


reverse (arr[N])
i
8
j
7
{
int i = 0
int j = N-1
while (i<j)
{
int temp = arr[i];
arr[i] = arr[j]
arr[j] = temp
//swap
i++
j--
}
}
reversed array


10
5
5
8
1
2
5
end
10
200
reversed
10
5
5
2
1
8
7
10
200