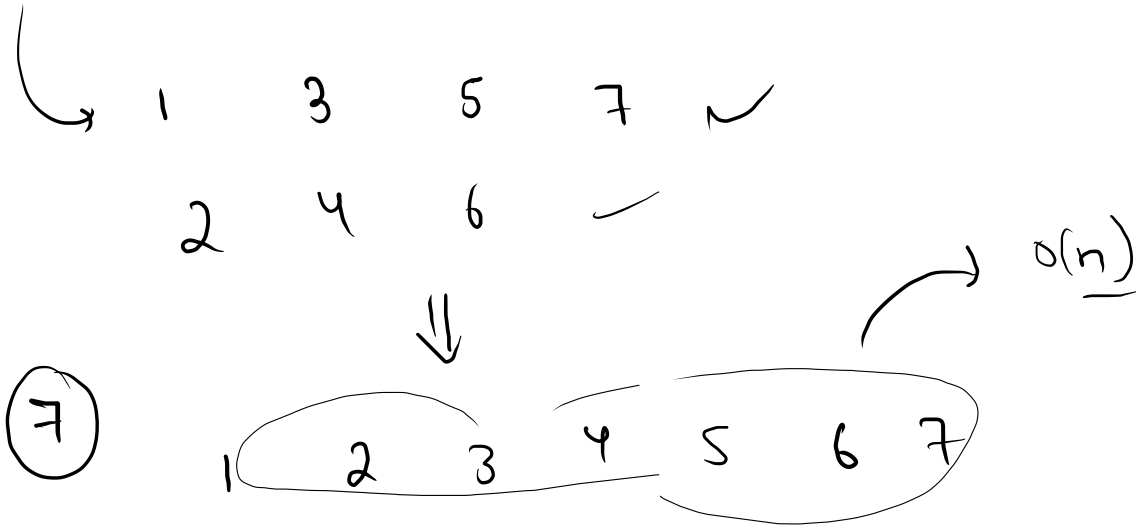
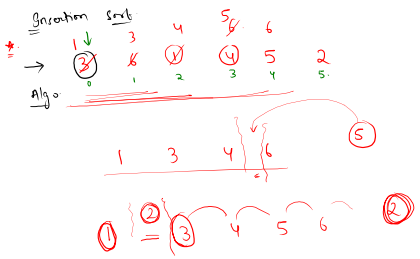
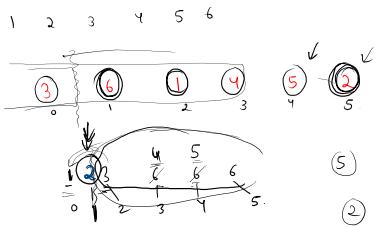
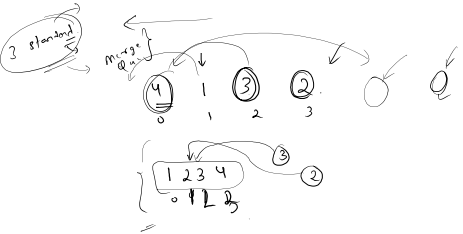
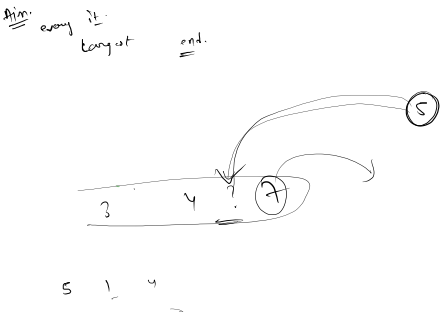
Merge 2 sorted Array.



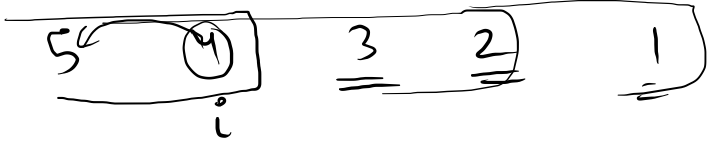

generation
Sort
Algo




3 standard
merge


Aim:
every it:
temp at
end.

eg.



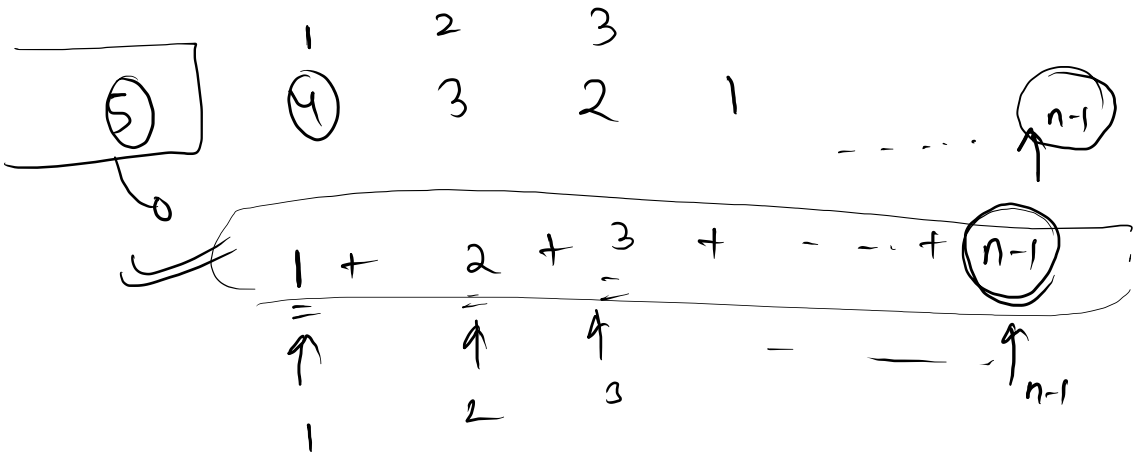
$$1 + 2 + 3 + 4 + \cdots + n-1$$

$$1 + 2 + 3 + 4 + \cdots + n-1 + 0$$

$$\frac{n(n+1)}{2}$$

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

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



Total

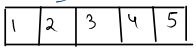
$$1 + 2 + 3 + 4 + \cdots + (n-1)$$

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

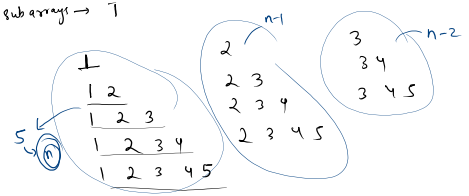
$$= \frac{n^2 - n}{2}$$

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

Print All subarray ?

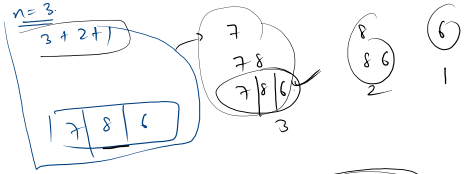


subarrays → 1



n = 5.

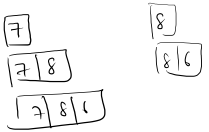
5 + 4 + 3 + 2 + 1



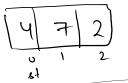
Total ⇒ $\frac{n(n+1)}{2}$

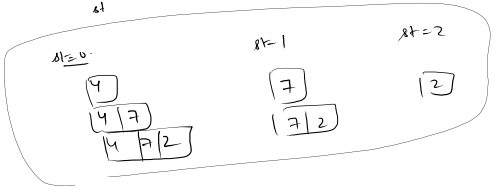
n = 4 → 4 + 3 + 2 + 1

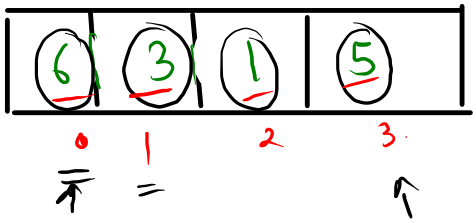
$\frac{n(n+1)}{2}$



Eg:







st → [0, n-1]

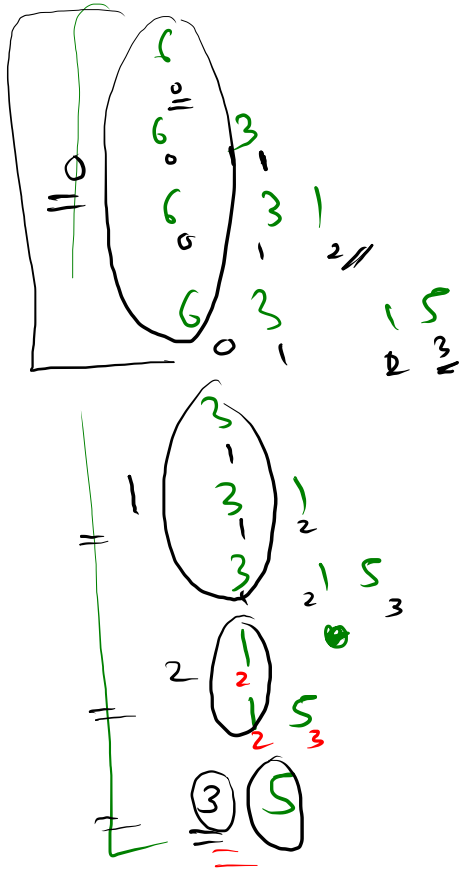
Obs

st = 0 → end → 0, 1, 2, 3

st = 1 → 1, 2, 3

st = 2 → 2, 3

st = 3 → 3

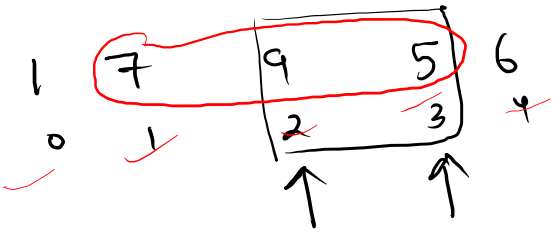


Kunal: → dry run.

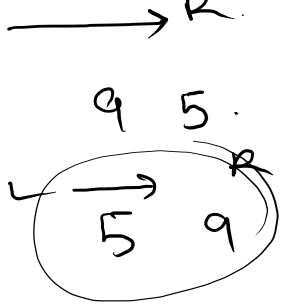
Yesterday.

1. Insertion sort ✓
2. All Subarray.

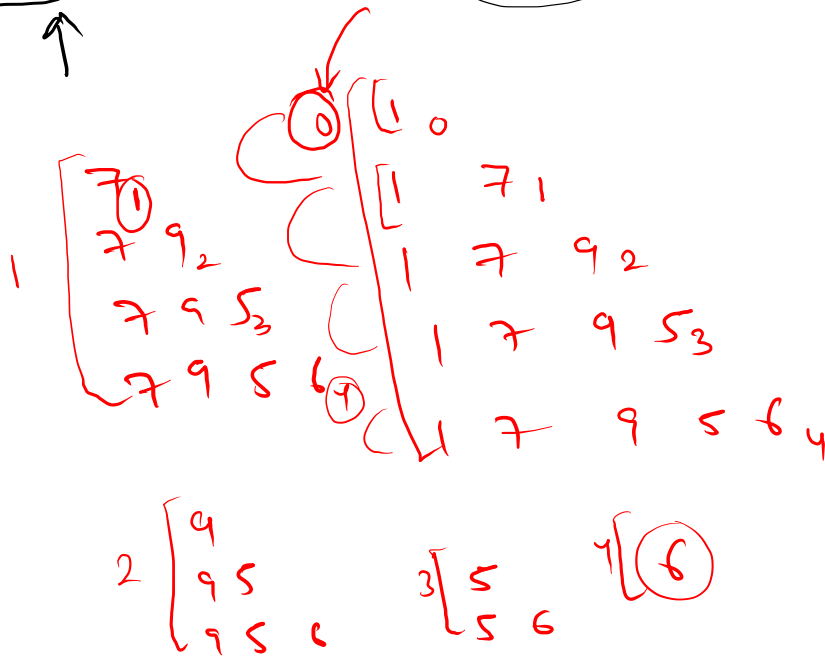
Sub array.



[1 , 3]

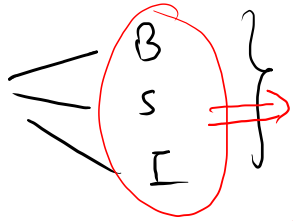


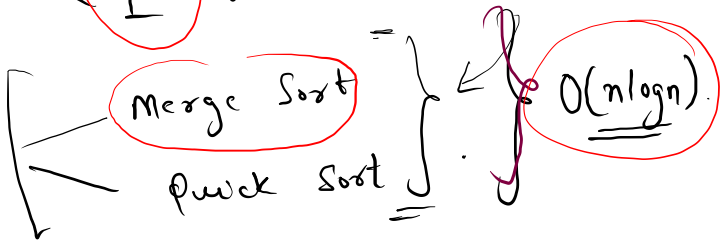

```
for(int st = 0 ; st < n ; st++){
    for(int ed = st; ed < n; ed++){
        for(int k = st; k <= ed; k++){
            System.out.print(arr[k] + " ");
        }
        System.out.println();
    }
}
```



Arrays.sort

Algo sorting

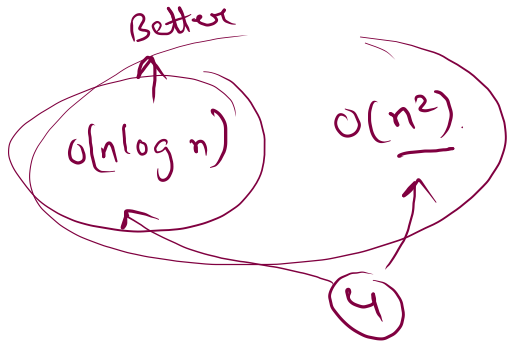




Java uses modified merge sort

⇓

O(nlogn)



CC2_01 Maximum Product of 3 Numbers

Take an integer **array** of size N as input and print the **maximum** product of three numbers.

Note: You should not use **inbuilt sort** function. } Array.sort

Sample Input 0

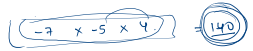
```
5
-7 3 -5 2 4
```

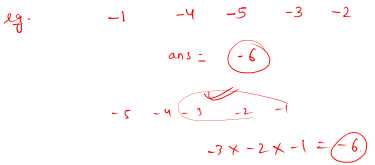
Sample Output 0

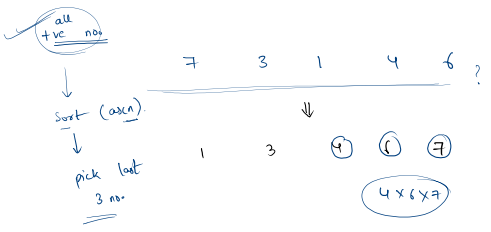
```
140
```

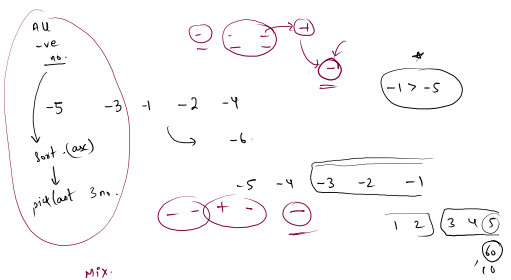
Constraints

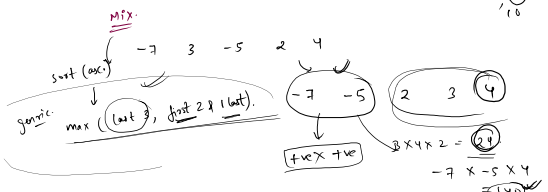
- 3<=N<=1000
- -1000<=arr[i]<=1000











eg.

5
-7 3 -5 2 4

max ⁿ of 3 no: in array

-7 3 -5 → -7×3×2 = -42
-7 3 2
-7 3 4 → 35×3 = 105
3 -7 -5
3 -7 2
1 3 -7 4

(7)

1 2 3 -4 -5 8

-5 -4 1 2 3 8 → 48

-5 × -4 = 20 × 8 = 160

-5 0 1 2 3 8

8 × -5 × 0 = 0

2 × 3 × 8 = 48

+ve +ve → res ↑
-ve -ve → res ↑
+ve -ve → res ↓

max

x y z a b c

max (x,y,c , a,b,c)

eg.

7 -4 3 1 2

-40 -10 2 3 7

r1 = 2.3.7 = 42
r2 = 400.7 = 280
r2 = -4.1.7 = -28

max(r1, r2) → r1

-4 1 -2 3 7

-4 -2 1 3 7

r1 = -4.-2.7 = 56
r2 = 1.3.7 = 21

-4 -2 -1 3 7

56

n = 5

Sort an array in wave form 1

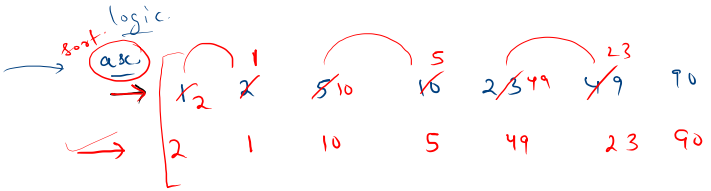
Given an unsorted array of integers, sort the array into a wave like array. An array 'arr[0..n-1]' is sorted in wave form if arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= .....

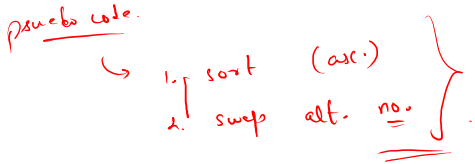
Sample Input 0

```
7
10 90 49 2 1 5 23
```

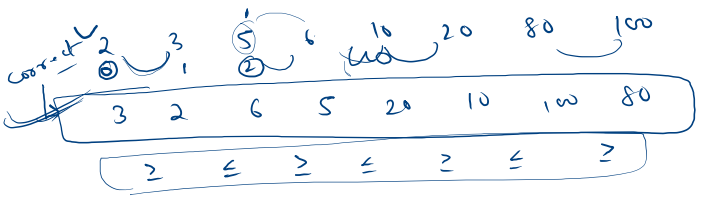
Sample Output 0

```
2 1 10 5 49 23 90
```





Input: arr[] = {10, 5, 6, 3, 2, 20, 100, 80}
Output: arr[] = {10, 5, 6, 2, 20, 3, 100, 80}

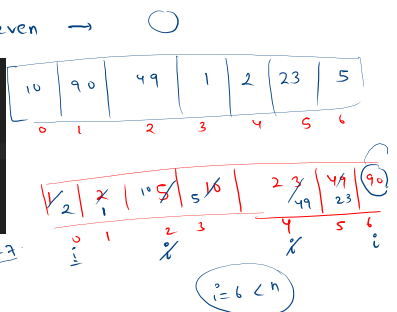


logic

idex even →

```
Arrays.sort(arr);

for(int i = 0 ; i < n-1 ; i += 2){
   int temp = arr[i];
   arr[i] = arr[i+1];
   arr[i+1] = temp;
}
for(int i = 0; i<n; i++){
    System.out.print(arr[i] + " ");
}
```



Max Subarray 2

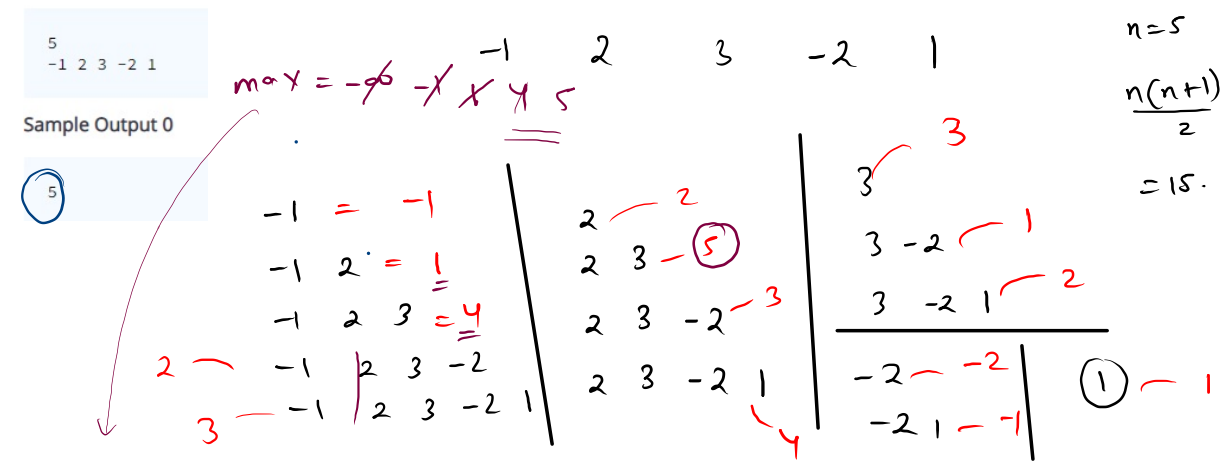
Given an array **arr[]** of N integers. Find the **contiguous sub-array**(containing at least one number) which has the **maximum sum** and print its sum.

Sample Input 0

```
5
-1 2 3 -2 1
```

Sample Output 0

```
5
```



T. print all ~~et~~ sub-array.

Kadane's: