Today's Content:

- → Inversion Count
- → Wave Array
- → Smaller Subarray to Sort
- → Contest: { Saturday 12AM - Monday }
- → Contest discussion monday

Q) Given 2 arrays A[N] & B[M]

(Indices)

Count no of pairs (i, j) such that A[i] > B[j]

Ex1:

	0	1	2
A :	7	3	5

	0	1	2
B :	2	0	6

⟨7, 2⟩ | ⟨3, 2⟩ | ⟨5, 2⟩
⟨7, 0⟩ | ⟨3, 0⟩ | ⟨5, 0⟩
⟨7, 6⟩ |

ans = 7 pairs

Ex2:

	0	1	2
A :	3	1	6

	0	1	2
B :	2	4	9

⟨3, 2⟩ | ⟨6, 2⟩
⟨6, 4⟩

ans = 3 pairs

Brute force idea :

Check all pairs & get count

```
c = 0
for(i=0; i<N; i=i+1){
    for(j=0; j<M; j=j+1){
        if(A[i] > B[j]){
            c = c+1
        }
    }
}
return c;
```

TC: O(N*M)

SC: O(1)

Idea 2

Eg 3: A[4] = { 7 8 2 4 } B[4] = { 3 5 1 10 }

↓

Sorting

A[4] = { 2 4 7 8 } (P1 P1 P1 P1)

B[4] = { 1 3 5 10 } (P3 P2 P2 P2)

1	2	3	4	5	7	8	10

c = 4 + 0 + 3 + 0 + 2 + 0 + 0 + 0 = 9

Eg 4: N = 4

A[4] = { 8 5 3 10 } B[4] = { 5 4 9 2 }

↓

Sort

A[4] = { 3 5 8 10 } (0 1 2 3) (P1 P1 P1 P1)

B[4] = { 2 4 5 9 } (0 1 2 3) (P2 P2 P2 P2)

· No: of Elements in left [P1 N-1] = N - P1

2	3	4	5	5	8	9	10

ans = 10

N - P1 + 0 | N - P1 + 3 | N - P1 + 2 | N - P1 + 1

N - P1 + 4

Pseudocode :

```
InversionCount(int A[], int N, int B[], int M){

    Sort(A) → NlogN        SC: O(N) + logN
    Sort(B) → MlogM        SC: O(M) + logM

    Total Iterations → NlogN + MlogM + N + M
    SC: O(N+M)

    c = 0, P1 = 0, P2 = 0

    while( P1 < N && P2 < M ){ → N+M

        if(A[P1] <= B[P2]){ // A[P1] should come first
            P1 = P1 + 1
        }
        else{ // B[P2] should come first // A[P1] > B[P2]
            c = c + N - P1
            P2 = P2 + 1
        }
    }

    // If one of them, they are no pairs

    return c;
}
```

28) Google / Microsoft / De Shaw / Inversion Count

Given A[N], find no: of pairs i, j such that i < j && A[i] > A[j]

Eg1: ar[5] = { 6 2 9 3 5 }

Indices: 0 1 2 3 4

+3 +0 +2 +0 +0 } Total = 5

Eg2: ar[10] = { 10 3 8 15 6 12 2 18 7 1 }

0 1 2 3 4 5 6 7 8 9

6 2 4 5 2 3 1 2 1 0 = 26

BF: Check all pairs

```
c=0;
i=0; i<N; i++){
    j=i+1; j<N; j=j++){
        if(A[i]>A[j]){
            c=c+1
        }
    }
}
```

TC: O(N^2)

SC: O(1)

	0	1	2	3	4	5	6	7	8	9
ar[10] = {	10	3	8	15	6	12	2	18	7	1 }

	0	1	2	3	4
A[ ] =	10	3	8	15	6

sort

= 3 6 8 10 15

{ total pairs, Sum }

	5	6	7	8	9
B[ ] =	12	2	18	7	1

total pairs, Sort

1 2 7 12 18

P_1 P_2

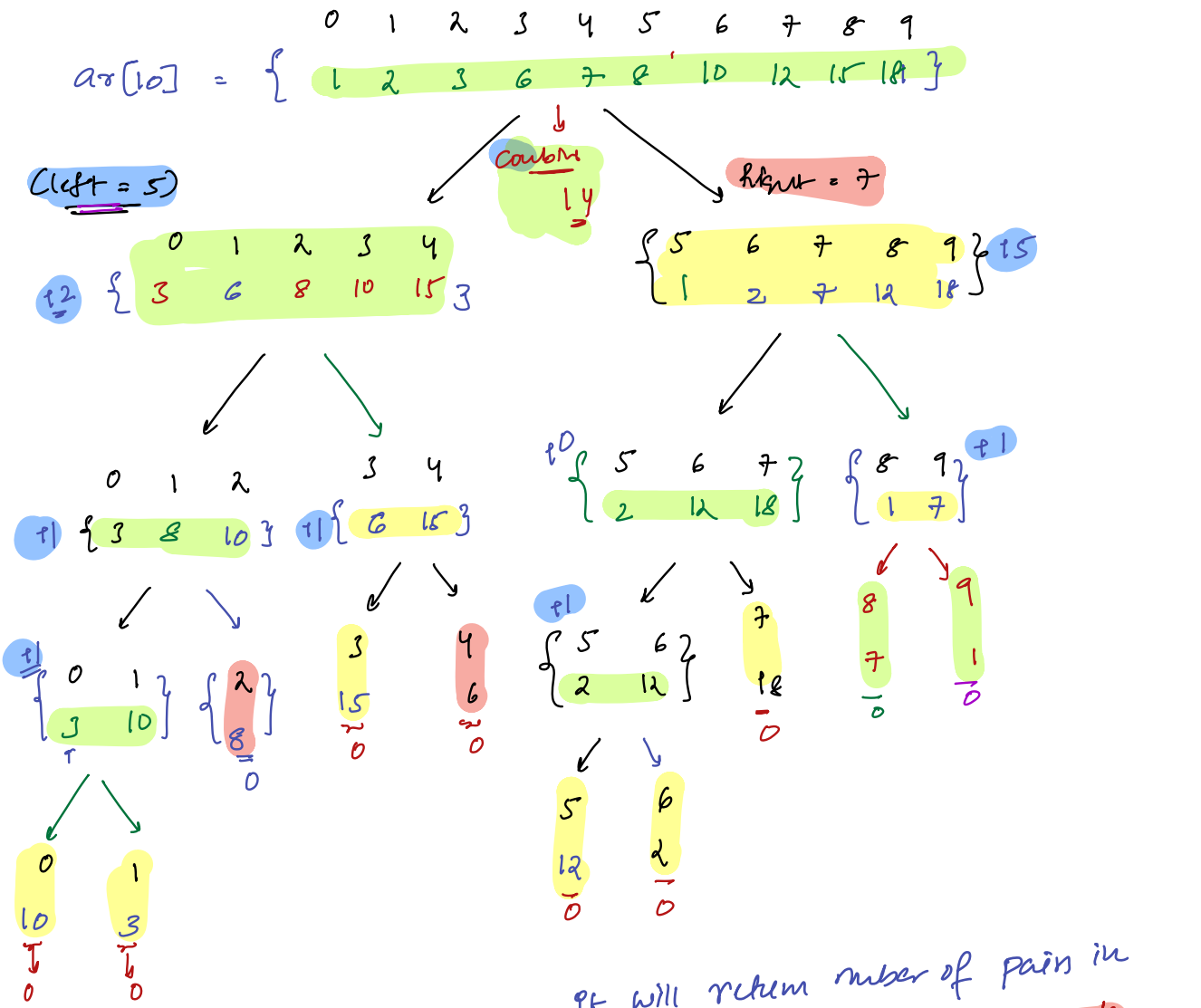
1 | 2 | 3 | 6 | 7 | 8 | 10 | 12 | 15 | 18 = 14

+5 +5 +3 +1

Total Pairs =

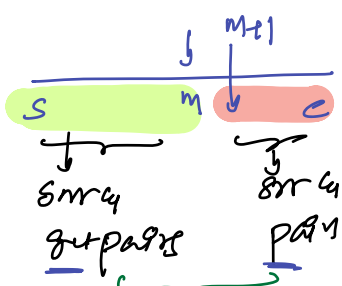
{ Total pairs A } + { Total pairs B } + { Total pairs Betwe A & B }

we need to sort A, & B so that we need can get pairs between them



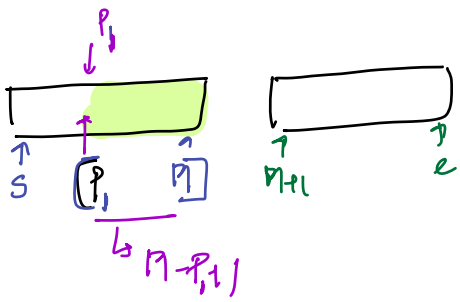
It will return number of pairs in subarray [s e] & also sort subarray [s e]

```
Inver( int ar[], int s  e){
    if(s==e) { return 0 }
    int m = (s+e)/2
    int L = Inver(ar, s, m)
    int R = Inver(ar, m+1, e)
    int C = mergeInv(ar, s, m, e)
    return (L+R+C)
}
```



1) get combine pairs
2) Merge as well

```
int mergeInv(int A[], int s, int m, int e){
    int tmp[e-s+1]; int c=0
    int p1=s, p2=m+1, p3=0
    while(p1<=m && p2<=e){
        if(A[p1]<=A[p2]){
            tmp[p3]=A[p1]; p1++; p3++
        }
        else{
            tmp[p3]=A[p2]; p2++; p3++; c=c+(m-p1+1)
        }
    }
    while(p1<=m){ tmp[p3]=A[p1]; p1++; p3++ }
    while(p2<=e){ tmp[p3]=A[p2]; p2++; p3++ }
    for(i=0; i<=e-s; i++){
        A[i+s]=tmp[i]
    }
    return c;
}
```



only extra pair

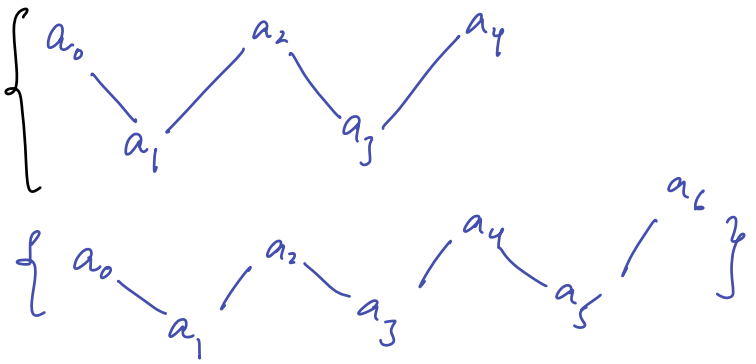
10:45 break

lowers

TC: O(NlogN)

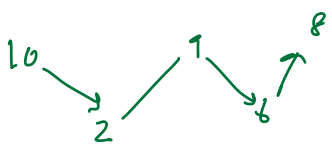
SC: O(N)

38) given N distinct array elements re-arrange array in a wave form



Ex: 6 8 2 9 10

ans: 6 10 9
2 8

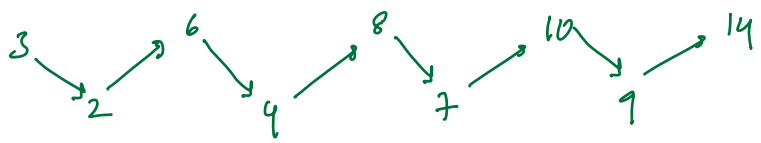


ans: 6 9 10
2 8

More than 1 wave is possible
Q) out of all wave arrays we can return any wave array.

Ex: 8 2 4 10 9 3 14 6 7
6

Sort: 2 3 4 6 7 8 9 10 14
5



Idea: Sort array

Swap adjacent pairs

```
i=0; i<N; i=i+2
    // take care Edge Cases
    swap(ar[i], ar[i+1])
```

TC: O(NlogN + N)

TC: O(NlogN)

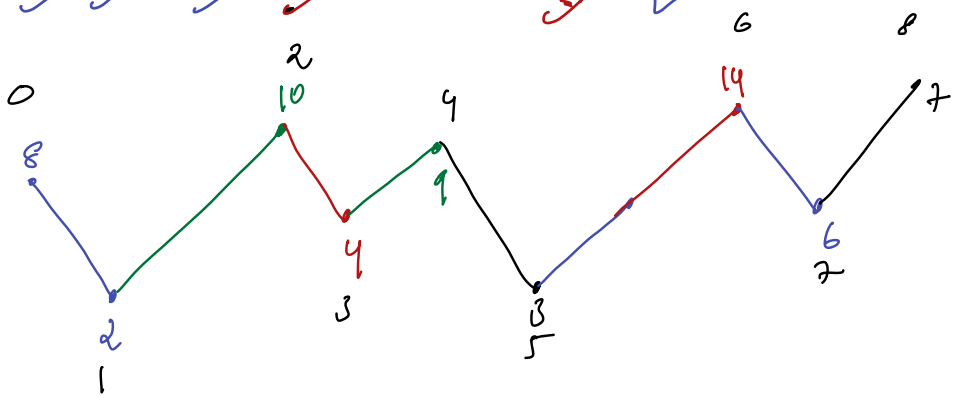
SC: O(N)

Lexicographically Best Way

If this is question we need to sort

Ex:

0	1	2	3	4	5	6	7	8
8	2	4	10	9	3	6	14	7
		10	4			14	6	



Idea:2 TC: O(N) SC: O(1)

```
i=0; i<(N-1); i=(i+1)        // Edge Case
```

→ check for every index

```
    if i%2==0 : in that from i, i+1 it should decrease
        if(ar[i] < ar[i+1]){ swap }
    if i%2==1 : in that from i, i+1 it should increase
        if(ar[i] > ar[i+1]){ swap }
```

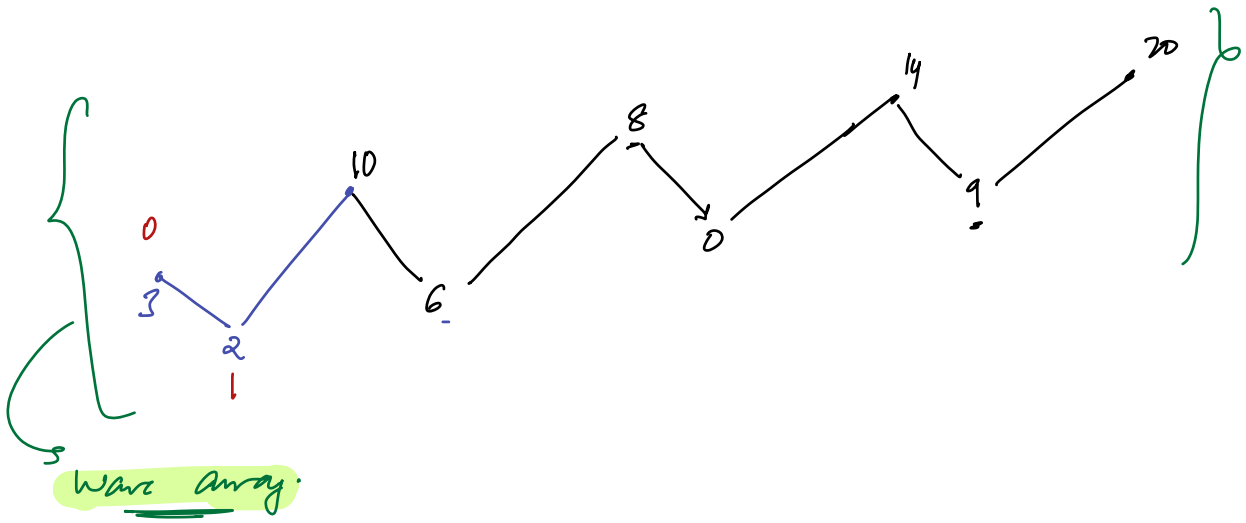
Eg:

3 2 ~~8~~ ~~10~~ 6 0 ~~9~~ ~~14~~ 20

10 ~~8~~ 8 14 9

2

6



// Starting on line:

0	1	2	3	4	5	6	7
1	3	4	6	8	10	14	20

Initially

[0 6] is sorted new elem at index 7

comp:

- 6 7 : ar[6] > ar[7] : swap
- 5 6 : ar[5] > ar[6] : swap
- 4 5 : ar[4] > ar[5] : swap
- 3 4 : ar[3] > ar[4] : swap
- 2 3 : ar[2] > ar[3] : swap
- 1 2 : ar[1] > ar[2] : (increas

[0 7] is sorted

// Eg:

0	1	2	3	4	5
1	3	7	11	14	20

[0 4]

i

4 | ar[i] > ar[i+1]

4 | ar[4] > ar[5] , swap, i = i--

3 | ar[3] > ar[4], swap, i = i--

2 | ar[2] > ar[3], swap, i = i--

1 | ar[1] > ar[2] swap, i = i--

0 | ar[0] > ar[1] swap, i = i--

-1 : Break (stop)

// ar[N]:

i = N-1 // i+1 out bour bounds

for(int i = 0 ; i < N-1 ; i = i+1) {

[0 i] sorted ← [i+1] → [0 i+1] as sorted

j = i;

while (j >= 0 && ar[j] > ar[j+1]) {

swap(ar[j], ar[j+1])

j--

}

}

Insertion sort

TC : O(N^2)

SC : O(1)

// given array [0 0] is sorted

p=0: // ar[5]:

0	1	2	3	4
3	2	-4	-1	0

p=1 ar[5]:

0	1	2	3	4
2	3	-4	-1	0

p=2 ar[5]:

0	1	2	3	4
-4	2	3	-1	0

p=3 ar[5]:

0	1	2	3	4
-4	-1	2	3	0

p=4 ar[5]:

0	1	2	3	4
-4	-1	0	2	3

array sorted

∴ no need to print anything

Note:

If entire array is sorted Insertion Sort: $O(N)$