Sorting : Arranging data in inc / dec order based on parameter

parameter

Ex1 : 4 7 11 14 18 → inc : numeric value

Ex2 : 18 14 11 7 1 → dec : numeric value

Ex3 : 1 2 11 9 6 10 : number of factors

	1	2	11	9	6	10
#factors	1	2	2	3	4	4

// Earnings

Heros Remuneration // SRK based remuneration inc

NTR	30
ramcharan	40
SRK	30
amer khan	60
Hrithik	35
Prabhas	40
Car driver	60

Heros	money
NTR	30
SRK	30
Hrithik	35
ramchar	40
Prabhas	40
amer khan	60
Car driver	60

Stable Sorting

// 2 data point who have same parameter, after sorting if relative order is maintained, Stable Sorting

↦ Sort data in inc order

Ex: ar[6] → 10 2 3 1 2 9

↓

ar[6] → 1 2 2 3 9 10

// Inplace: Sorting algorithm with O(1) Space.

Q) Given N array elements find k^{th} smallest element

Input

$k \leq \log N$

ar[8] : 2 8 4 -1 6 7 5 10 , k = 2 → 2 , k = 3 → 4

Ideal: ⇒ Sort & get k^{th} Element

$N \log N + O(1) \Rightarrow TC: O(N \log N)$

	0	1	2	3	4	5	6	7
ar[8] :	-1	2	4	8	6	7	5	10

get 1^{st} min ⇒ p = 0 ⇒ Iterate from [0, N-1] & get min ind : i = [0, N-1] ⇒ Swap (ar[ind], ar[0])

get 2^{nd} min ⇒ p = 1 ⇒ Iterate from [1, N-1] & get min ind : i = [1, N-1] ⇒ Swap (ar[ind], ar[1])

get 3^{rd} min ⇒ p = 2 ⇒ Iterate from [2, N-1] & get min ind : i = [2, N-1] ⇒ Swap (ar[ind], ar[2])

// Repeat above for k times

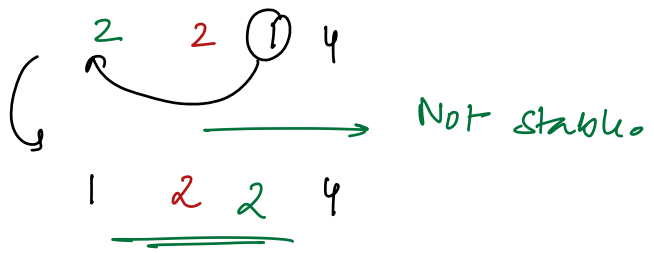
TC: $O(k^1 N)$ SC: $O(1)$

// Repeat above for N-1 times : Array Sorted ⇒ TC : $O(N^2)$ SC: $O(1)$

Selection Sort ⇒ Not stable

Eg: 2 2 1 4 —Stable→ 1 2 2 4

↓ as per above algorithm



// Pseudo code:

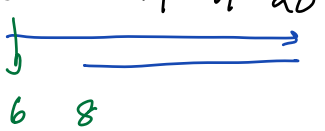
// Sort Entire array.

```
for(i=0; i<N-1; i++){
    minv = ar[i], ind = i;
    for(j=i; j<N; j++) {
        if(ar[j] < minv) {
            minv = ar[j]
            ind = j;
        }
    }
    Swap(ar[i], ar[ind])
}
```

// Selection Sort

TC: O(N²) SC: O(1)

Eg:
ar[5]: 6 8 14 19 20



// k^{th} Smallest, $k < \log N$

- If we sort : $N \log N$
- If we apply Selection sort : $k * N$

$k * N < N \log N$

// 28) Given an array, sort in asc order but we we can only swap adjacent elements. ⇒ (Bubble Sort)

	0	1	2	3	4	5	6	7
ar[8] :	3	-1	6	9	4	2	5	4
After 1 iterat :	-1	3	6	4	2	5	4	9
After 2 iterat :	-1	3	4	2	5	4	6	9
After 3 iterat :	-1	3	2	4	4	5	6	9

last j index

i=0 : j < N-1

i=1 : j < N-2

i=2 : j < N-3

i ; j < N-i-1

Repeat N-1 times array is sorted

```
for(i=0; i<N-1; i++){
    c=0
    for(j=0; j<N-i-1; j++){        → j = N-1, j+1 = N } Array out of bounds
        if(ar[j] > ar[j+1]){
            // swap ar[j] & ar[j+1] ; c++
        }
    }
    if(c==0) { No swaps, sorted, break }
}
```

N-2 N-1

// N Elements

i = 0 : j : [0, N-2] : N-1
i = 1 : j : [0, N-3] : N-2
i = 2 : j : [0, N-4] : N-3
⋮
1

Sum of N-1 natural numbers

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

$\Rightarrow$ TC = $O(N^2)$ SC = $O(1)$

Ex: ar[5] : 3 4 6 10 14

After : 0 swaps → break;

Inplace ✓ Stable

// 2 2

↳ never swap, relative order won't change, Stable

38) Given 2 sorted array A[N], B[M], merge & create & return a sorted array.

a[3] : { -1, 4, 8 }

b[2] : { 2, 9 }

c[5] : { -1, 2, 4, 8, 9 }

Sol 1: c[N+M]

copy A[] → c[] : N

copy B[] → c[] : M

sort c[] : (N+M) log(N+M)

TC : N+M+(N+M) log(N+M)

//

a[7] : { (-5) -1 3 7 (10) 12 15 }

b[5] : { (-4) 0 2 8 9 }

c[12] : { (-5), -4, -1, 0, 2, 3, 7, 8, 9, 10, 12, 15 }

```
merge ( A[], N, B[], M) {
    C[N+M];
    P1 = 0, P2 = 0, P3 = 0
    while( P1 < N && P2 < M) {
        if( A[P1] < B[P2] ) {
            C[P3] = A[P1] P1++; P3++
        }
        else {
            C[P3] = B[P2], P2++, P3++;
        }
    }
    while ( P1 < N) { C[P3] = A[P1], P1++, P3++ }
    while (P2 < M) { C[P3] = B[P2], P3++, P2++ }
}
```

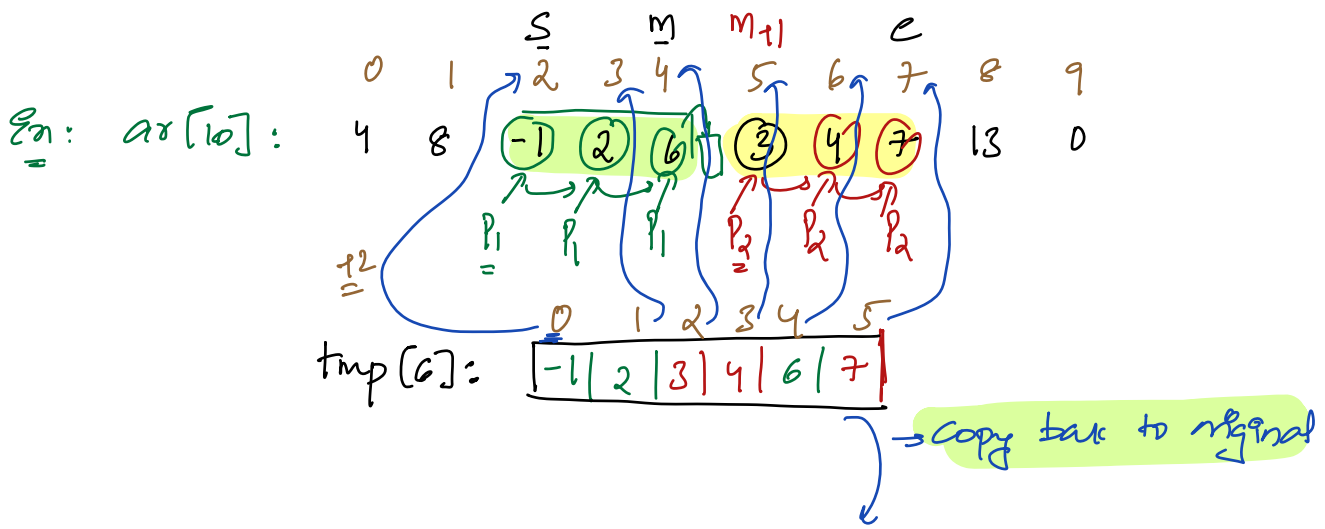
TC: $O(N+M)$ SC: $O(1)$

10:45PM break

48) Given N array elements & 3 indices s m e

given // Subarray → [s, m] is sorted
// Subarray → [m+1, e] is sorted

Q Sort Subarray from [s e]

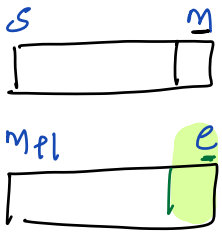


s m e : 4 8 -1 2 3 4 6 7 13 0

2 4 7

```
// merge ( A[], s, m, e) {
    T[e-s+1]
    P1 = s, P2 = m+1, P3 = 0
    while ( P1<=m && P2<=e)
    |   if( A[P1] < A[P2]) { T[P3] = A[P1]; P3++, P1++}
    |   else { T[P3] = A[P2]; P3++; P2++}
    }
    while ( P1<=m) { T[P3] = A[P1]; P3++, P1++}
    while (P2<=e)  { T[P3] = A[P2]; P3++; P2++}

    copy T[] -> A[]
    i = 0; i<=(e-s); i++) {
    |   A[s+i] = T[i]
    }
}
```



TC : $O(e-s+1) \Rightarrow O(N)$

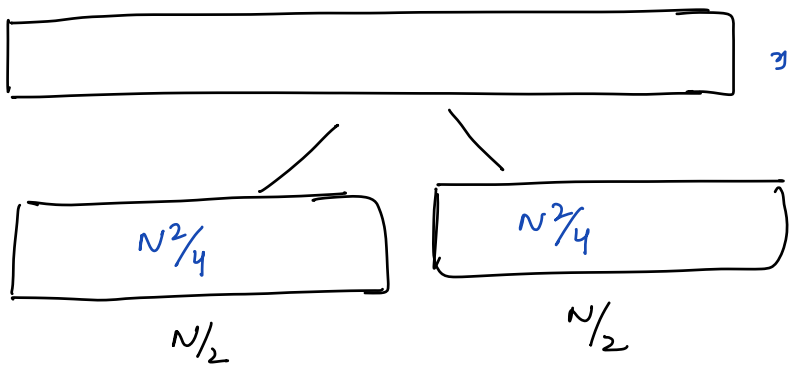
SC : $O(e-s+1) \Rightarrow O(N)$

TODO: => Home work

Wether we can do it in O(N) time without extra

50) Given N Array elements we have to sort

↳ 100

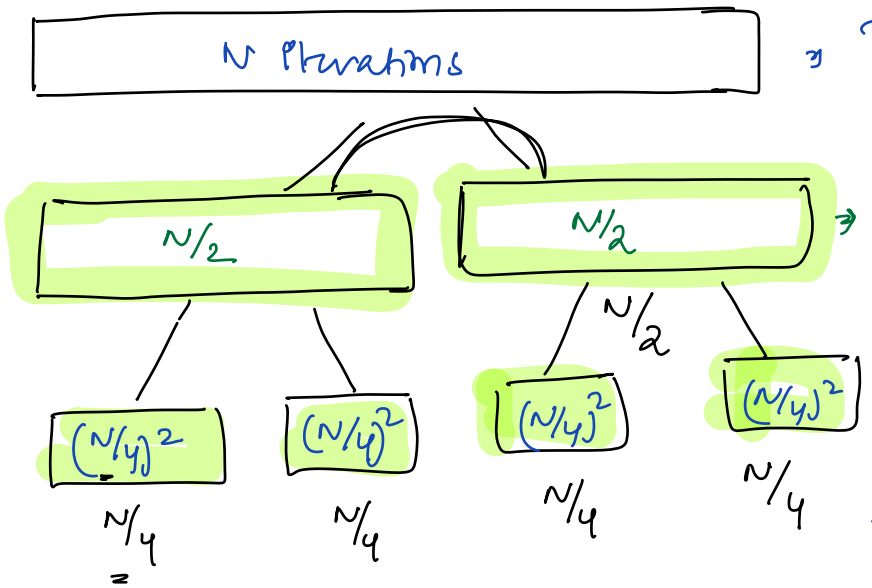


$\Rightarrow N^2 \Rightarrow 10^4$ iterations

$\Rightarrow \frac{2N^2}{4} + N \Rightarrow \frac{10^4}{2} + 100$

$5 * 10^3 + 100$

$\Rightarrow$ 5100



$(N/4)^2 * 4 + 2N$

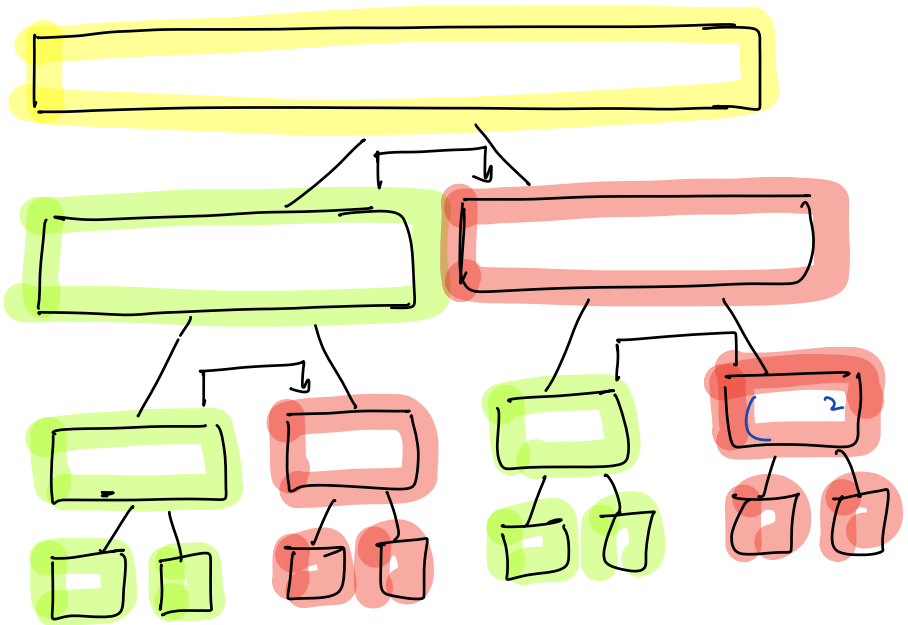
$\frac{N^2}{4} + 2N$

$N = 100$

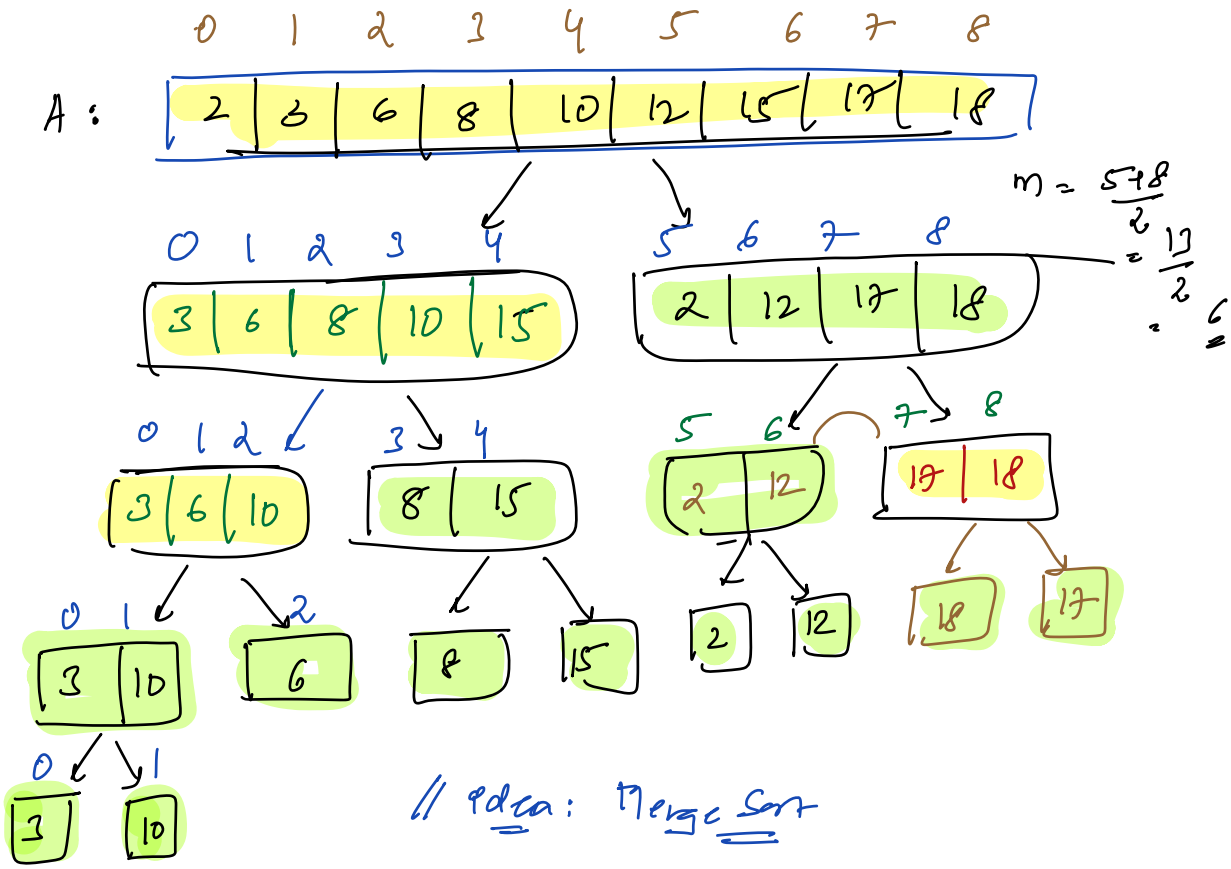
$10^4 + (2)(100)$

$2500 + 200$

$\Rightarrow 2700$



//

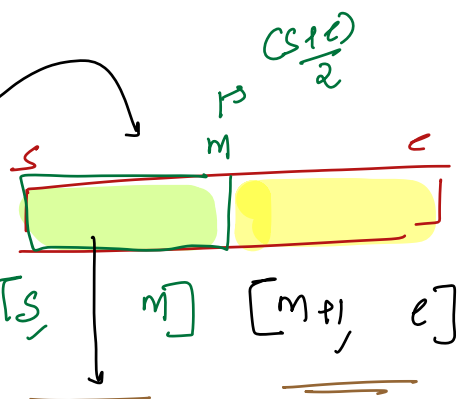


// Idea: Merge Sort

Ask: Sort subarray from [s, e]

```
// mergeSort( A[], s, e ) {
    if(s == e) return;
    m = (s + e)/2
    // mergeSort(A, s, m)
    // mergeSort(A, m+1, e)
    // merge(A, s, m, e)
}
```

main logic:



$T(N) = 2T(N/2) + N$

$T(N) = N \log N$ //

SC : $O(N) + \log(N)$ //

HW: Stable

```
// void mergeSort(A[], s, e) {
    if(s==e) return;
    m = (s+e)/2
    mergeSort(A, s, m)
    mergeSort(A, m+1, e)
    merge(A, s, m, e)
}

// merge(A[], s, m, e) {
    T[e-s+1]
    P1 = s, P2 = m+1, P3 = 0
    while(P1<=m && P2<=e)
        if(A[P1] < A[P2]) { T[P3] = A[P1]; P3++, P1++ }
        else { T[P3] = A[P2]; P3++; P2++ }
    }
    while(P1<=m) { T[P3] = A[P1]; P3++, P1++ }
    while(P2<=e) { T[P3] = A[P2]; P3++; P2++ }

    for(i=0; i<=(e-s); i++) {
        A[s+i] = T[i]
    }
}
```

2nd only place where 2 element are getting compared

This is where we use custom comparator

$\Rightarrow T(N) = 2T(N/2) + N$

$T(N/2) = 2T(N/4) + N/2$

$= 2 * [2T(N/4) + N/2] + N$

$= 4T(N/4) + 2N \Rightarrow 2^2 T(N/2^2) + 2N$

$T(N/4) = 2T(N/8) + N/4$

$= 4[2T(N/8) + N/4] + 2N$

$= 8T(N/8) + 3N \Rightarrow 2^3 T(N/2^3) + 3N$

$= 16T(N/16) + 4N \Rightarrow 2^4 T(N/2^4) + 4N$

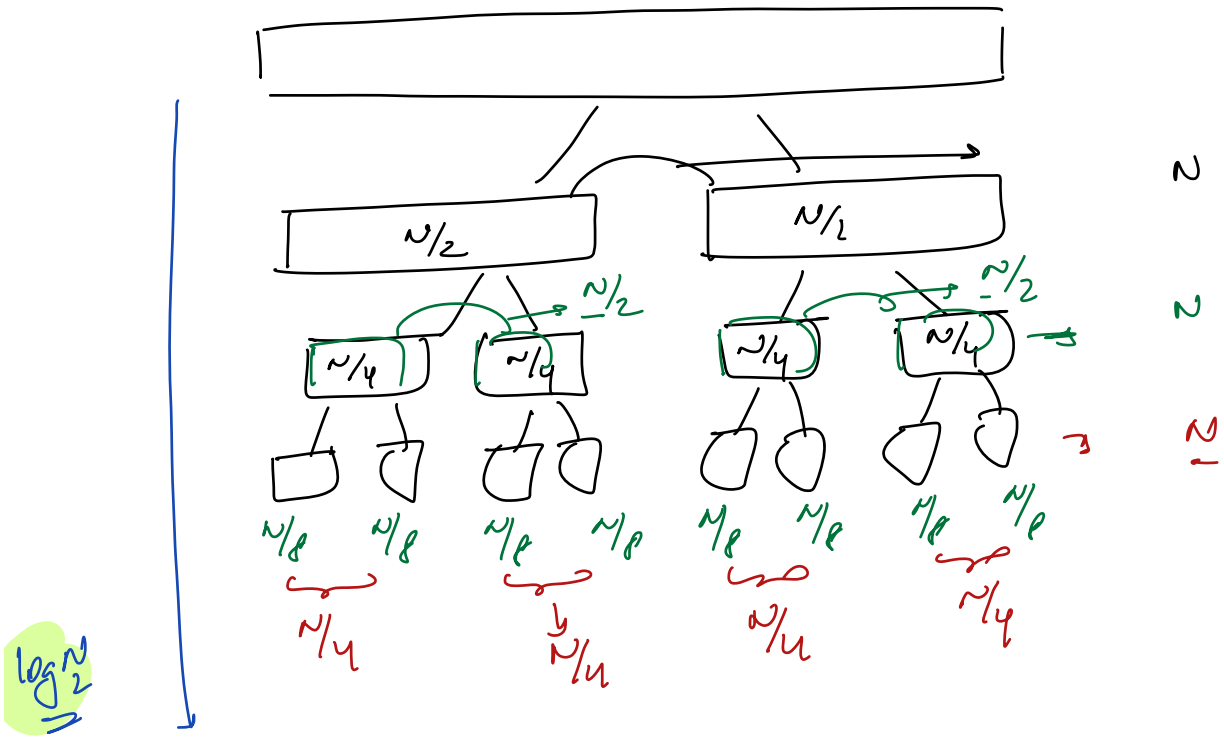
$\Rightarrow 2^k T(N/2^k) + kN \Rightarrow k = \log_2 N$

$2^{\log_2 N} T(1) + \log_2 N * N$

$N + N \log N$

$TC \Rightarrow O(N \log N)$ $\quad$ recursive space

$SC \Rightarrow O(k) \Rightarrow O(\log N)$



→ $N \log \frac{N}{2}$

Idea: At every level we have we will have only N iterations for merge

Total levels are → (log N)

TC: O(N log N)