Today's content

- Quick Sort
- Count Sort
- Radix Sort (#hint)

Q) Given N array elements, re-arrange the elements such that

- arr[0] should go to its sorted position
- All elements <= arr[0] go to left side of arr[0]
- All elements > arr[0] go to right side of arr[0]

arr[11]: { 10 3 8 15 6 12 2 18 7 15 4 }
0 1 2 3 4 5 6 7 8 9 10


idea-1:

Sort the array.

arr[11]: { 2 3 4 6 7 8 10 12 15 15 18 }
0 1 2 3 4 5 6 7 8 9 10

{T.C → O(N log N).}

idea-2:

arr[11]: { 10 3 8 15 6 12 2 18 7 15 4 }
0 1 2 3 4 5 6 7 8 9 10

temp → [ 3 8 6 2 7 4 10 15 18 12 15 ]
0 1 2 3 4 5 6 7 8 9 10

arr[5]: { 10 18 3 10 7 }
0 1 2 3 4

temp[5]: { 3 10 7 10 18 }
0 1 2 3 4

p1 p2

pseudo-code:

```
void re-arrange ( arr, N) {
    temp [N];
    p1 = 0, p2 = N-1;
    for ( i = 1; i < N; i++) {
        if (arr[i] ≤ arr[0]){
            temp [p1] = arr[i]
            p1++
        }
        else {
            temp [p2] = arr[i]
            p2--
        }
    }
    temp [p1] = arr[0]
    //Copy all elements from temp[] to arr[].
}
```

{ T.C → O(N)
S.C → O(N) }

optimize S.C.

idea-3 → re-arrange without using any extra space.

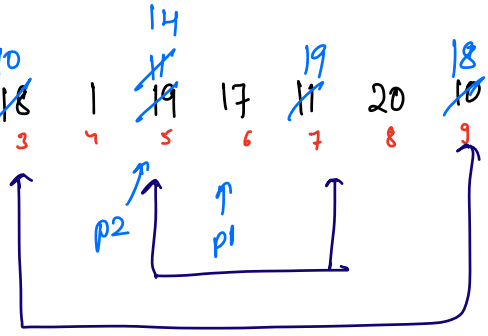
① arr[11]: { 10 3 8 15 6 12 2 18 7 15 4 }
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9) (10)


if p1 > p2 → break.

{ 2 3 8 4 6 7 10 18 12 15 15 }

→ swap arr[0] with arr[p2]

② arr[12]: { 14 6 8 18 1 19 17 11 20 10 33 29 }
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11)




if (p1 > p2) break.

swap arr[0] with arr[p2]

arr[12]: { 11 6 8 10 1 14 17 19 20 18 33 29 }

③ arr[4]: { 8 4 1 7 }
(0) (1) (2) (3)


if (p1 > p2)
break;
swap (arr[0], arr[p2])

arr[4]: { 7 4 1 8 }

```
void rearrange (int[] arr, int N){
    p1 = 1 , p2 = N-1
    while ( p1 <= p2 ){
        if (arr[p1] <= arr[0]){
            p1++
        }
        else if ( arr[p2] > arr[0]){
            p2--
        }
        else{
            swap arr[p1] with arr[p2]
            p1++ , p2--
        }
    }
    // swap arr[0] with arr[p2]
}
```

T.C → O(N)
S.C → O(1)

Q) Given N array elements and subarray from [s, e]
Re-arrange subarray [s,e] such that arr[s] should come to correct position in subarray.

s = 2, e = 7.

Eg → arr[11]: { 10 3 [ 8 15 6 12 2 18 ] 7 15 4 }
(indices: 0 1 2 3 4 5 6 7 8 9 10)

p2 ↑ 15 (index 6), p1 ↑ 2 (index 3)

swap arr[s], arr[p2].

arr[11]: { 10 3 [ 6 2 8 12 15 18 ] 7 15 4 }

pseudo-code.

```
int rearrange (int[] arr, int s, int e) {
    p1 = s+1, p2 = e
    while ( p1 <= p2 ) {
        if (arr[p1] <= arr[s]) {
            p1++
        }
        else if ( arr[p2] > arr[s]) {
            p2--
        }
        else {
            swap arr[p1] with arr[p2]
            p1++ , p2--
        }
    }
    // swap arr[s] with arr[p2]
    //should return correct position of arr[s]
    return p2
}
```

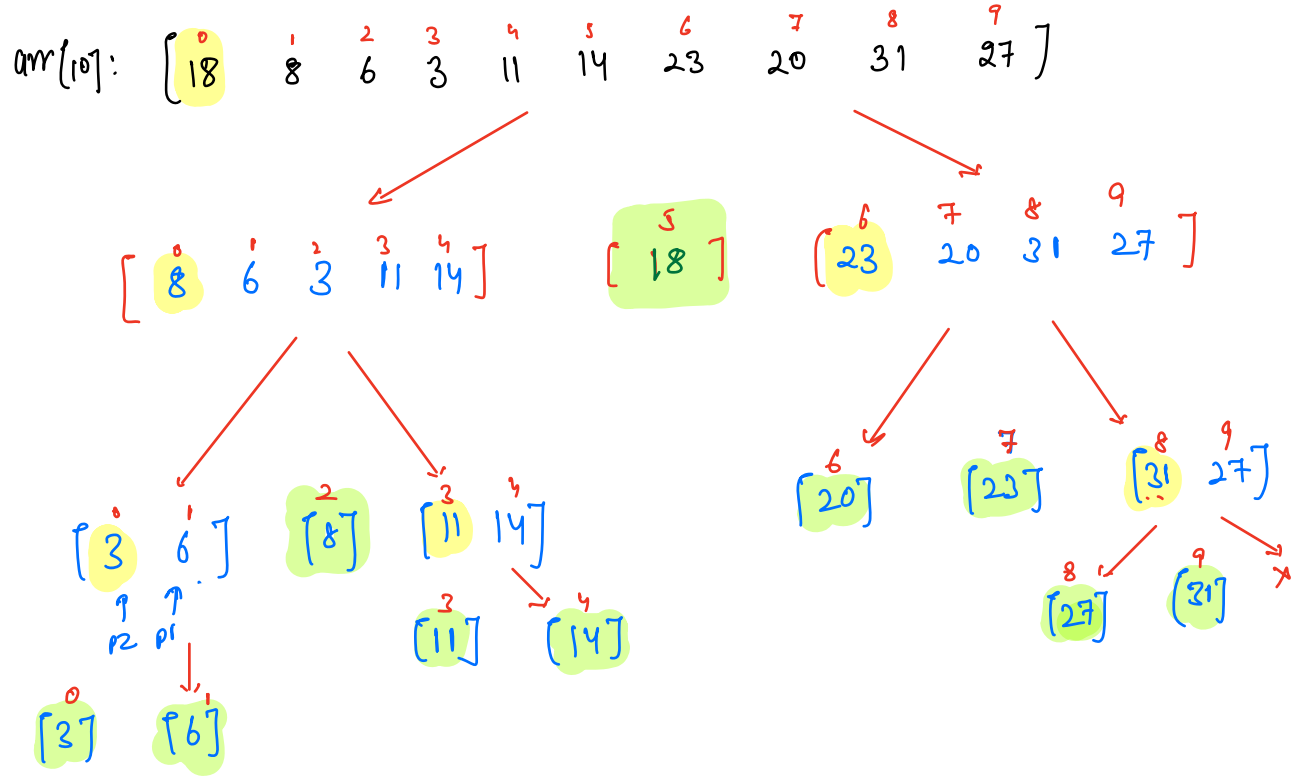
T.C → O(N)
S.C → O(1)

//Aim : Sort the sub-array from s to e.

```
void QuickSort ( arr[], s , e) {
    if( s >= e ) { return }
    p = rearrange (arr, s, e);
    QuickSort ( arr, s, p-1)
    QuickSort ( arr, p+1, e)
}
```

// T(0) = 1.

→ {Doubt session}.




arr[10] → { 3 6 8 11 14 18 20 23 27 31 }

Time Complexity.

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

$$\{ T(N) = 2T\left(\frac{N}{2}\right) + N. \}$$

T.C → $O(N \log N)$ , S.C → $O(\log_2 N)$

in worst case.

$T(N) = N + T(N-1)$

$T(N-1) = (N-1) + T(N-2)$

$T(N) = N + (N-1) + T(N-2)$

$T(N-2) = (N-2) + T(N-3)$

$T(N) = N + (N-1) + (N-2) + T(N-3)$

$T(N-3) = (N-3) + T(N-4)$

$T(N) = N + (N-1) + (N-2) + (N-3) + T(N-4)$

Generalization

$T(N) = (N) + (N-1) + (N-2) + \cdots + T(N-K)$

// $N - K = 0 \Rightarrow K = N$

{ T.C → $O(N^2)$, S.C → $O(N)$ }

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

## Worst case condition

→ when reference element is minimum or the maximum element.

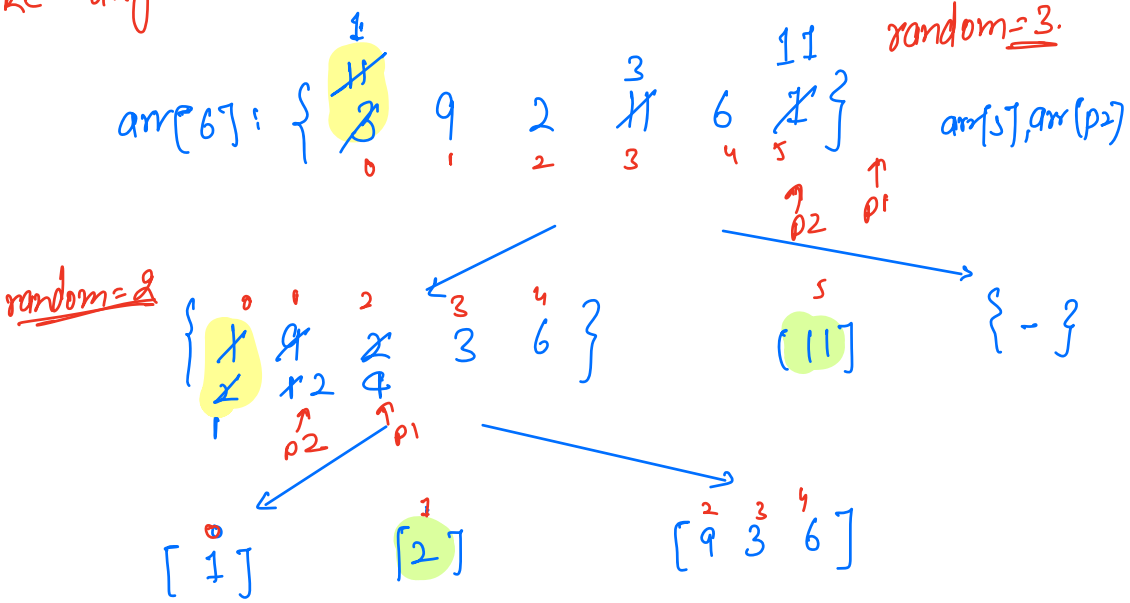
→ Eg:→ Sorted array in inc/dec order.

arr[5]: { 1 2 3 9 16 }

arr[5]: { 16 7 4 2 0 }

## Randomized Quick Sort

→ take any random element as reference.




Why? → If you use random order, then probability of picking min/max element as reference element is very low.

Q) Given an array elements where all the elements are in the range [1-4]. Sort the array.

Eg: arr[10]: { 3 1 4 4 2 4 2 3 1 2 }
0 1 2 3 4 5 6 7 8 9

① → sort using Merge Sort. T.C - O(N log N)

② → HashMap to store frequency of every element.

| |
|---|
| 3 → 2 |
| 1 → 2 |
| 2 → 3 |
| 4 → 3 |

```
K = 0
for ( i = 1 ; i <= 4 ; i++){
        int count = hm[i];
        for( j = 1 ; j <= count ; j++){
                print(i);
                { arr[K] = i ; }
                  K++
        }
}
```

arr[] is now sorted.

{ T.C → O(N)
S.C → O(1) }

// Given N array elements, where every element lies in the range [a, b]. Sort the array.

pseudo-code:

```
HashMap < int, int > hm;
// iterate & insert all elements in hashmap. → N.
k = 0
for ( i = a ; i <= b ; i++){
        int count = hm[i];
        for( j = 1 ; j <= count ; j++){
                print(i);
                { arr[k] = i ; }
                  k++
        }
}
// arr[] is now sorted
```

total outer loop iterations = b - a + 1 (R)

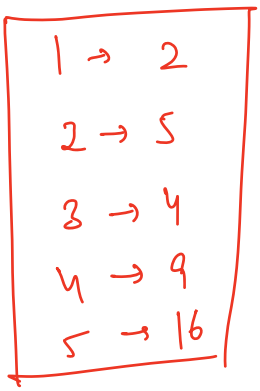
total inner loop iterations = N.

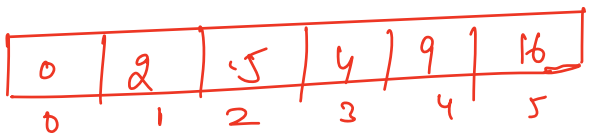
total iterations = N + R + N
= 2N + R

{ T.C → O(N+R)
S.C → O(R) }

| i | j | iterations |
|---|---|---|
| a | [1, freq of a] | freq(a) |
| a+1 | [1, freq of a+1] | freq(a+1) |
| ⋮ | | ⋮ |
| b | [1, freq of b] | freq(b) |

→ N.




(# try to use array instead of hm.)

{ T.C → O(N+R)
S.C → O(R) }

R ≤ N

T.C → O(N+N)
{ T.C → O(N)
S.C → O(N) }

R > N log N

T.C → (N + N log N)
T.C → O(N log N)
S.C → O(N log N)

Count sort?.

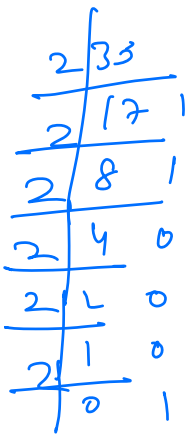
[→ Generally used when a small range is given & data is very huge.]

---

{H.W} → count sort stable ?

$A = 35$ , $B = 6$.

$(100011)_2$ $(0110)_2$




dvd = div * quo + remainder

$q \rightarrow (1 << 1)$ div * quo = $(B << 1)$

$= (1100)_2 \leq A$.

$q \rightarrow (1 << 2)$ div * quo = $(B << 2)$

$\rightarrow 4$ $= (1100)_2 \leq A$.

$q \rightarrow (1 << 3)$ div * quo = $(B << 3)$

$\rightarrow 8$. $= (111000)_2 > A$

$q \rightarrow 100$

$q \rightarrow 5$ 101

$\rightarrow 6$ 110

$\rightarrow 7$ 111

$A = 35$ , $B = 6$.

$(100011)_2$ $(0110)_2$

$q \rightarrow 100$

$q \rightarrow 1\ 0\ 0\ 0$

[dvd = div * quo + remainder]

$B << 0 \Rightarrow B \times 1$

$B << 1 \Rightarrow B * 2^1$ $[q = 2]$ $\leq A$ ✓

$B << 2 \Rightarrow B * 2^2$ $[q = 4]$ $\leq A$ ✓

$B << 3 \Rightarrow B * 2^3$ $[q = 8]$ $\leq A$ ✗

$A = [A - (B << 2) = 35 - 24 = 11]$

$q \rightarrow 101$

$B << 0 \Rightarrow B * 1$ $[q = 1]$ $\leq 11$ ✓

$B << 1 \Rightarrow B * 2^1$ $[q = 2]$ $\leq 11$ ✗