Sorting 4

Friday, 5 March 2021 8:04 PM

1Q) Given an array re-arrange the array in such a way that all the less than ar[0] are to the left of it & all the elements greater than ar[0] are to the right of it ar[0]

Ex: 8 2 9 10 13 5 7 ⇒ 2 5 7 8 9 10 13

Ex2: 3 9 -1 0 -3 11 6 1 → -1 0 -3 1 3 9 11 6

0	1	2	3	4	5	6	7
-1	0	-3	1	3	6	11	9

ar[0]

→ Sort the array

TC: NlogN SC: O(N)

Ex2: 10 3 7 1 9 11 5 21

tmp[]

0	1	2	3	4	5	6	7
3	7	1	9	5	10	21	11

p1 p1 p1 p1 p1 p2

[Copy to original array]

```
void reorder(int ar[], int n){
    int tmp[n], p1 = 0, p2 = n-1
    for(int i=1; i<n; i++){
        if(ar[i] > ar[0]) { tmp[p2--] = ar[i] }   // we place ar[i] in back
        else { tmp[p1++] = ar[i] }
    }
    tmp[p1] = ar[0] // Both p1 & p2 are same
    // copy tmp[] → ar[]
    // At which index we are inserting ar[0] // p1
}
```

TC: O(N) SC: O(N)

→ reorder (w/o extra space)

Ex: ar[] =

0	1	2	3	4	5	6	7	8	9
10	3	8	7	6	2	12	15	13	11

Ex2:

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

3 1 2 1 9 6 3 5 6 7

1 1 2 3 9 6 3 5 6 7

<= >=

All data <= x | All data >= x

↦ [99% of Quick Sort]

```
partition(int ar[], int lo, int hi){
    int p1 = lo+1, p2 = hi; int ele = ar[lo]
    while(p1 < p2){
        if(ar[p1] < ele){ p1++; continue; }
        // p1 is at wrong position
        if(ar[p2] >= ele){ p2--; continue; }
        // p2 is at wrong position
        swap(ar[p1], ar[p2])
    }
    if(ar[p1] <= ele){
        swap(ar[p1], ar[lo]) // p1
        return p1
    }
    else{
        swap(ar[p1-1], ar[lo]) // p1-1
        return p1-1
    }
}
```

→ Note: Partition element can be anything, pick a random and from [lo-hi] swap(ar[lo], ar[rnd]) continue

Ex: 8 3 2 7 10

7 3 2 8 10

Ex2: 10 3 8 1 6 7 2 18 12 15

Exp: 10 3 2 8 7

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

a_0 a_1 a_2 a_3 a_4 a_5 ... a_{n-3} a_{n-2} a_{n-1}

[Correct position]

$[a_1\ a_3\ a_4\ a_{n-3}\ a_{n-1}]$ a_0 $[a_2\ a_5\ a_6\ a_{n-2}]$

0 k-1 k k+1 N-1

```
void quickSort(int ar[], int lo, int hi){
    ① if(lo >= hi) return
    // Use ar[lo] to partition entire array
    ④ {int k = partition(ar, lo, hi)}
    // ar[lo] is placed in its actual position k.
    // Left part ⇒ (lo, k-1)
    ② quickSort(ar, lo, k-1) ✓ ] ⇒ 9N/10 ⇒ T(9N/10)
    // Right part = (k+1, hi)
    ③ quickSort(ar, k+1, hi) ✓ ] N/10 ⇒ T(N/10)
}
```

Recurrence Relation:

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

In Best Case ⇒ O(NlogN)

In worst Case ⇒ O(N²)

In worst Case:

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

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

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

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

$= N + N-1 + N-2 + \dots 1 + T(0)$

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

$= O(N^2)$

In Avg Case

partition 90% 10%

$T(N) = N + T(9N/10) + T(N/10)$

⇒ If we solve this using recurse-tree method

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

Merge Sort:

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

1) Substitution
2) Recursive Tree

Masters Thermom [Simplified] ⇒ order of polynomial

$T(N) = aT(N/b) + O(N^c)$

$t = \log_b^a$, $b \ge 1$, $a \ge 1$

: $c < t$: $T(n) = \Theta(n^t)$

$c = t$: $T(n) = \Theta(n^t \log n)$

$c > t$: $T(n) = \Theta(n^c)$

→ tight bound

$T(N) = 2T(N/2) + N$ → [Because of merge sort]

$a = 2, b = 2, c = 1$

$t = \log_b^a = 1$

$t == c$: $\Theta(n^t \log n)$

: $\Theta(n\log n)$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

⇒ int ar[20] = {1 2 3 3 4 4 5 5 1 2 2 3 4 5 2 1 1 3 3 4}

⇒ $1 \le ar[i] \le M$

```
int cnt[M+1] = {0}
for(int i=0; i<n; i++){
    cnt[ar[i]]++;
}
```

→ Iterations ⇒ O(N) → count Array

cnt ⇒ [0, 4, 4, 5, 4, 3]

```
int k=0
for(int i=1; i<=M; i++){
    for(int j=1; j<=cnt[i]; j++){
        ar[k++] = i;
    }
}
```

Iterations

i	j
1	cnt[1]
2	cnt[2]
3	cnt[3]
M	cnt[M]

M * N

Iterations: 5N

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

// 1 - 4
2 - 4
3 - 5
4 - 4
5 - 3

$10^9 \le ar[i] \le 10^9 + 1000$

cnt[1000] = {0}

value	index (low offset)
10^9	0
10^9+1	1
10^9+2	2
10^9+1000	1000

→ I can construct cnt[]

$a \le ar[i] \le b$

// R = b-a+1

cnt[R] = {0}

value	index
a	0
a+1	1
a+x	x
b	b-a

⇒ cnt[]

```
a ≤ ar[i] ≤ b;
cnt[b-a+1] = {0};
for(int i=0; i<n; i++){
    cnt[ar[i]-a]++;
}
// Using cnt array for given array
int k=0
for(int i=a; i<=b; i++){
    for(int j=1; j<=cnt[i-a]; j++){
        ar[k++] = i;
    }
}
```

→ b-a+1 + O(N)

⇒ R = b-a+1 ⇒ [Max arr[] - Min arr[] + 1]

TC: O(N+R) SC: O(R)

If R ≈ N

⇒ If $R \approx 10^6$ ✓, If $R > 10^6$ *

If R < NlogN: $T(N+R) < N\log N$

If R ≥ NlogN: $T(N+R) > N\log N$ *