Sunday, 12 September 2021 12:08 PM

Randomized Quick Sort

Quick sort is considered different from other sorting algos because it is randomized.

What is a randomized algo ??

In a normal algo

input [Unsorted array] → Algo → output [sorted array]

In a randomized algo,

input → Algo → output

Condition: ensure of regular intervals given n

random (n)

output optimum size "at"

→ Quick sort is randomized.

→ sort the array (a)

random (n [0 ... n-1])

we'll pick a random pivot element

x = a [random] → [0, ... n-1]

Split the array into 2 parts.

→ Linear time → we can partition the array.

→ Recursively for the left & the right parts until the size of the array is 0

Same array

We need to sort all elements from (L to n-1)

```
sort ( l, r ):

    if ( r-l ) ≤ 1 :
        return ;

a (array) → m = partition ( l, ... n-1 )

    for ( i = l ... )
        if ( a [i] < m )
            array [ a[i], a [m] ]
            m = i ;

    sort ( l, m ) ;
    sort ( m, r ) ;
```

Time Complexity :

Worst case complexity of normal quick sort :

picking max / min elt.

TC → O(n²)

Calculating time complexity of randomized algo, we take mean of operations.

good split

good split → $\frac{1}{2}$

bad split → $\frac{2}{3}$

$$T(n) = \sum_{k=1}^{n-1} (n + T(k) + T(n-k))$$

(1) → Partitioning cost pivot = elt.

$T(k)$ & $T(n-k)$ → a pivot

$$T(n) \leq \frac{1}{3}\left(n + T(\tfrac{n}{3}) + T(\tfrac{2n}{3})\right) + \frac{2}{3}(n + T(n))$$

a bad pivot

$$T(n) \leq n + \frac{1}{3}\left(T(\tfrac{n}{3}) + T(\tfrac{2n}{3})\right) + \frac{2}{3}T(n)$$

$$T(n) \leq n + \frac{1}{3}T(\tfrac{n}{3}) + \frac{1}{3}T(\tfrac{2n}{3}) + \frac{2}{3}T(n)$$

$$T(n) - \frac{2}{3}T(n) \leq n + \frac{1}{3}T(\tfrac{n}{3}) + T(\tfrac{2n}{3})$$

$$T(\tfrac{n}{3}) \leq n + T(\tfrac{n}{3}) + T(\tfrac{2n}{3})$$

$$T(n) \leq 3n + T(\tfrac{n}{3}) + T(\tfrac{2n}{3})$$

Proving by Induction,

$$\boxed{T(n) \leq c \cdot n \log n}$$

$$T(n) \leq 3n + c\tfrac{n}{3}\log\tfrac{n}{3} + c\tfrac{2n}{3}\log\tfrac{2n}{3}$$

$\log n - \log 3$; $\log n - \log\frac{3}{2}$

$$= n(3 + \tfrac{c}{3}\log n - \tfrac{c}{3}\log 3 - \tfrac{2c}{3}\log n + \tfrac{2c}{3}\log\tfrac{3}{2})$$

$$\leq 3 + \tfrac{2c}{3}\log\tfrac{3}{2}$$

$$= c \cdot n\log n + n\left(3 - \tfrac{c}{3}\log 3 - \tfrac{2c}{3}\log\tfrac{3}{2}\right)$$

$$\boxed{T(n) = c n \log n}$$

$\frac{1}{3}$ → good split ; $\frac{2}{3}$ → bad split

⇒ every 3 rd pick is a good pick.

1. Pick 3 random elts
2. Sort them
3. Take middle one as pivot

Inverse of sorting → Time sort (Insertion sort + Merge sort)

Cost → Intro sort (Quick sort + Heap sort + Insertion sort)

B) Given an array and a number k, k ≤ size of the array.

→ find the kth smallest element in the array.

Eg : arr [] → { 7, 10, 4, 3, 20, 15, 6 }
k = 3
ans → 6

Eg : arr [] → { 7, 9, 13, 22, 34, 6 }
k = 5
ans → 34

Approach (1) : sort the array and pick the kth smallest elt.

TC → nlogn

Approach (2) :

pick a random elt (x) and split the array in parts.

x = a [random (0 ... n-1)]

find (l, r, k)

```
    if (k-l == 1) return a[l];
    x = a[random ( l, ... r-1 )]
    m = l ;
    for ( i = l , ... r-1 )
        if (a[i] < x)
            array [ a[i], a[m] ]
            m++ ;
    if (k < m)
        return find (l, m, k);
    else
        return find ( m, r, k);
```

$$\left(n + \tfrac{2}{3}n + \tfrac{4}{9} \ldots\right) = 3n$$

qs

TC → O(n)

Count Sort

→ sort the elts by counting the frequencies of each elt in the array.

Eg 4 2 2 8 3 3 1

1. Find the max elt of the array
   max → 8
2. Make new array of size max + 1 & initialize with 0

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

3. Take the original array & place elements at the right place

1	2	2	3	3	4	8

Above set is not stable.

arr → 4 2 2 8 3 3 1

count → 0 1 2 2 1 0 0 0 1

output → 1 2 2 3 3 4 8

Time Complexity = O(n+k) → max

Space Complexity = O(max)