Quick Sort & Comparator

Rajat Sharma
Sarthak
Naval Oli
Balaji S K
Subhranil Kundu
sharath r
Saurabh Ruikar
Vasanth
amit khandelwal
Rajendra
soumya hubballi
Vishal Mosa
Aditya
Garga Chakrabarty
Akansh Nirmal
Madhan Kumar M S
Pankaj
Sneha L
Murali krishna Talluri
Sushant
Divya P
Purusharth A
suyash gupta
Gowtham
Bhaveshkumar
Sanket Giri
Shani Jaiswal
Vimal
Gagan Kumar S
Sridhar Hissaria
Gokul Aditya S
Shradha Srivastava
Abhishek Sharma

Agenda

- Pivot Partition
- Quick Sort
- Comparator Problems

Current PSP

↓

66 %.

Q> Given an integer array, consider the first element as pivot and rearrange the elements such that for all i → p

if (A[i] < p) then it should be present on left side

if (A[i] > p) then it should be present on right side

All elements are distinct.

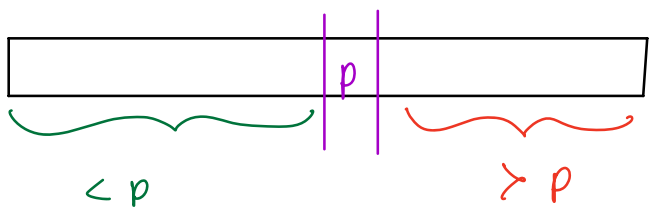
Return the correct position of pivot

A = 54 26 93 17 77 31 44 55 20

26	17	31	44	20	54	93	77	55
0	1	2	3	4	5	6	7	8

ans 5 ∵ 5 is the index of 54

first element is p



TC : O(N log N)

Bruteforce Sort the array.

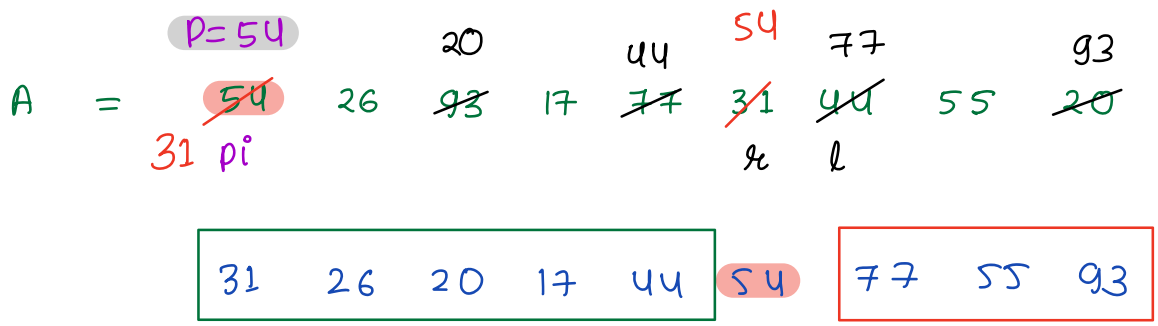
A = 54 26 93 17 77 31 44 55 20

= 17 20 26 31 44 54 55 77 93

Approach 2

Create an extra array and populate all the smaller elements first {first pass}
then pivot
then all the greater elements. {second pass}

TC : O(N)



if A[l] < p {l++}
if A[r] > p {r--}

r gives the correct position of pivot.

Pseudocode

```
int partition ( A[N] ) {
    p = A[0]
    l = 1
    r = N-1

    while ( l <= r ) {
        if ( A[l] < p ) { l++ }
        else if ( A[r] > p ) { r-- }
        else {
            swap (A, l, r)
        }
    }
    swap( A, 0, r)
    return r
}
```

TC : O(N)

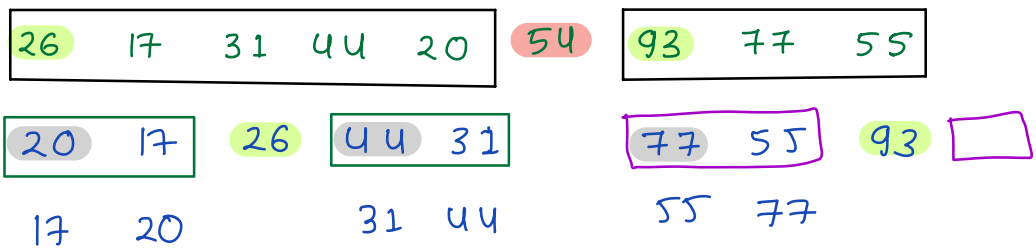
SC : O(1)

HW

Change the above code to handle duplicates ?

Quick sort {divide & conquer}

A = 54 26 93 17 77 31 44 55 20




```
void quicksort ( A[N] , l , r ) {
    if ( l < r ) {
        pi = partition ( A , l , r )   // first element as pivot
        quicksort ( A , l , pi - 1 )
        quicksort ( A , pi+1 , r )
    }
}
```

```
int partition ( A[N] , start , end ) {
    p = A[start]
    l = start + 1
    r = end

    while ( l <= r ) {                    // Handle duplicates
        if ( A[l] <= p ) { l++ }
        else if ( A[r] > p ) { r-- }
        else {
```

swap(A, l, r)

swap(A, start, r)
return r

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

Time complexity of Quick Sort

1 2 3 4

2 3 4

3 4

4

the sub problem reduces by only one element.

3 1 2 4

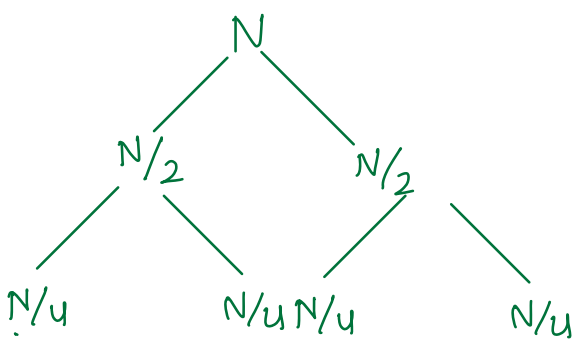
1 2 3 4

1 2 4

3 splits the array in ~ half

Best case scenario for quicksort

pivot always splits the array in half



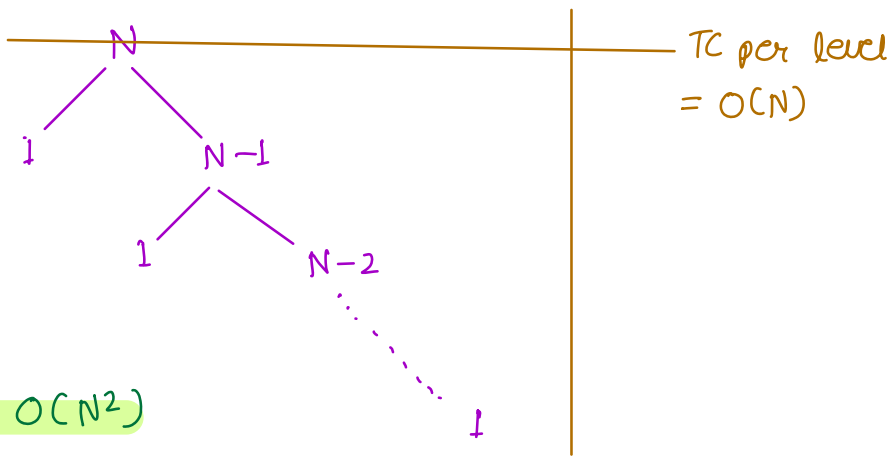
SC : O(log N)

due to recursion stack.

Exactly same as merge sort N log N

Worst case TC for quick sort

pivot is the smallest element.



overall TC = $O(N^2)$

SC = O(N)

Randomized Quicksort

54 26 93 17 77 31 44 55 20

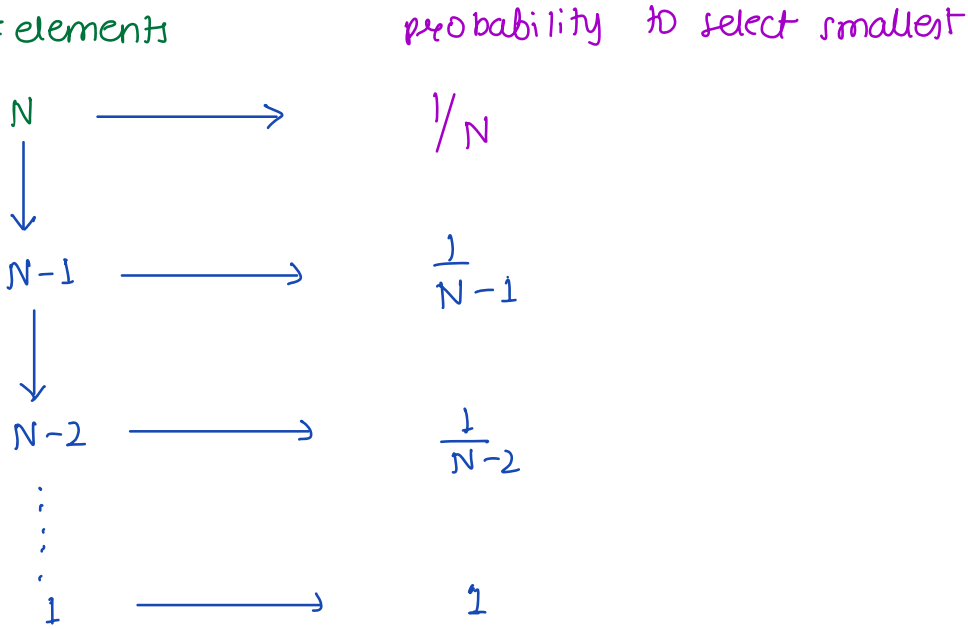
Instead of choosing A[0] as pivot

we pick any element at random to be the pivot

54 26 93 17 77 31 44 55 20

Probability of the worst case scenario

pivot is the smallest element.



probability of picking smallest element at random always.

$$\frac{1}{N * (N-1) * (N-2) \ldots 1} = \frac{1}{N!} \approx \text{very small}$$

For the above reason

TC for randomised quicksort = $O(N \log N)$

SC = $O(\log N)$

Size of the largest array that can be sorted ? $10^{\wedge}6$

value of $\log_2(10^{\wedge}6) \approx 20$

For this reason SC for sorting is considered $O(1)$

Comparator

Break 10 : 38

A comparator is a function that compares two values and returns a result indicating whether values are

less than

equal to

greater than

For Languages — Java, Python, JS, C#, Ruby

f ⟶ first argument

s ⟶ second argument

return

-ve ⟶ f comes before s

+ve ⟶ f comes after s

0 ⟶ f == s

For Languages — C++

f ⟶ first argument

s ⟶ second argument

return

true ⟶ f comes before s

false ⟶ otherwise.

sorting based on factors

Given A[N], sort the data in ascending order of count of factors. If the factor count is same sort them on the basis of their magnitude. {asc}

A	=	9	3	10	6	4
#factors	=	3	2	4	4	3

Order ⟶

3	4	9	6	10
2	3	3	4	4

⟶ Count of factors in ascending order.

Assuming countFactors function is already present.

TC : $O(C\sqrt{N})$

return
- true ⟶ f comes before s
- false ⟶ otherwise.

C++

```
bool compare (val1, val2) {
    cnt1 = countFactors(val1)
    cnt2 = countFactors(val2)

    if (cnt1 == cnt2) {
        if (val1 < val2) { return true}
        else return false
    }

    return cnt1 < cnt2
}
```

Python

```
A.sort ( key = lambda x : (countFactors(x), x))
```

Java

```
collections.sort ( A , (f , s) ⟶ {
        cnt1 = countFactors (f);
        cnt2 = countFactors (s);
        if (cnt1 == cnt2) return f - s ;
        return cnt1 - cnt2 ;
});
```

NOTE : f - s ⟶ ascending order on magnitude

cnt1 - cnt2 ⟶ ascending order on #factors

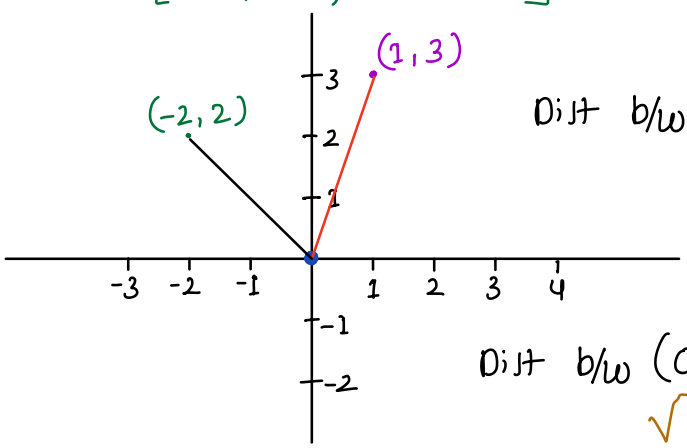
B closest points to origin **

Given array of points where points[i] = $[x_i, y_i]$ and an integer B.

Return B closest points to the origin {0,0}

Distance b/w (x_1, y_1) & (x_2, y_2) = $\sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}$

points = [[1,3], [-2,2]] B = 1



Dist b/w (0,0) & (1,3)

$\sqrt{1^2+3^2} = \sqrt{10}$

Dist b/w (0,0) & (-2,2)

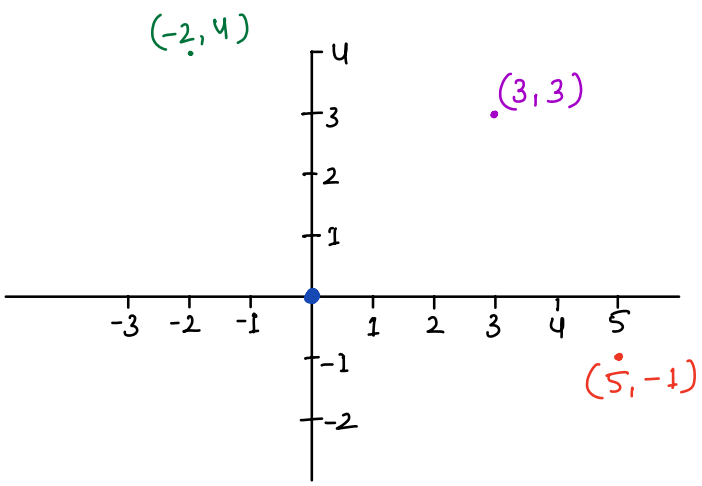
$\sqrt{2^2+2^2} = \sqrt{8}$

sorted points = [-2,2], [1,3]

ans = [[-2,2]]

points = [[3,3], [5,-1], [-2,4]] B = 2

Dist from (0,0) $\sqrt{18}$ $\sqrt{26}$ $\sqrt{20}$



sorted points $[3,3]$, $[-2,4]$ $[5,-1]$

distance from 0,0 $\sqrt{18}$ $\sqrt{20}$ $\sqrt{26}$

ans = $[[3,3], [-2,4]]$

Size of ans == B

Approach Use comparator

Assume we have two points p_1 & p_2

Dist b/w p_1 & origin = $\sqrt{x_1^2 + y_1^2}$

Dist b/w p_2 & origin = $\sqrt{x_2^2 + y_2^2}$

NOTE : you can ignore sqrt in the distance formula.

Revision Framework

Pre lecture

→ Read before the class {content to be taught}

→ Next day after the class → Revise all notes
Revise the assignment.

Bookmark the question which is tough for you
for a revision later

→ Every Sunday revise all class notes & assignment
for the week.
