Lecture: Quick sort and comparator

Agenda

- Sort 0 and 1.
- Quick sort
- Comparator problems.

Qu-1 Given an array A[n] of 0's and 1's in random order.

Sort the given array.

ip:

0	1	0	0	1	1	0	1	0

op:

0	0	0	0	0	1	1	1	1

Brute force:

Bubble sort — $O(n^2)$

Insertion " — $O(n^2)$

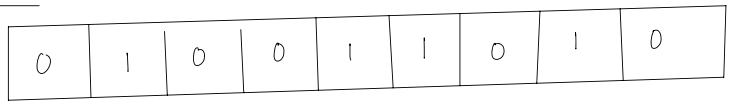
Arrays.sort(arr) — $O(n \log n)$

Count sort :- $O(n)$, $O(1)$

freq[2]

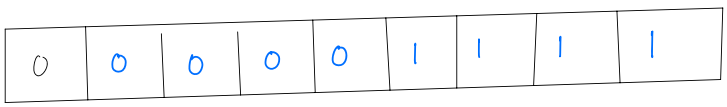
- Iterate array to build freq[]
- " freq[] for my answer.

Approach2



count0 = 5

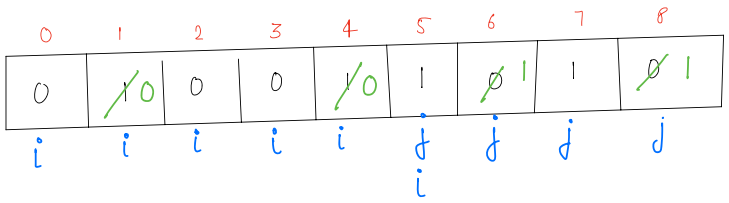
count1 = 4



Approach 3

TC: O(n)
SC: O(1)
} Expectation

Only 1 iteration



$i = 0$

$j = 1$

Algorithm

```
void sort01 ( int[] A) {
    i = 0;
    j = A.length-1;
    while( i < j) {
        if ( arr[i] == 0) {
            i++;
        } else {
            if ( arr[j] == 0) {
                swap (A, i, j);
            }
            j--;
        }
    }
}
```

TC: O(n)

SC: O(1)

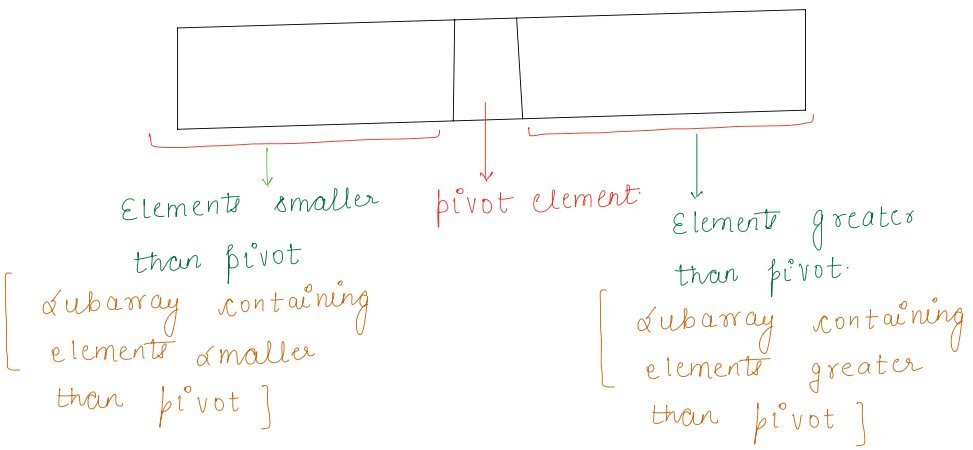
itr = 1 itr

Qu. Given A[], consider first element as pivot, rearrange the elements such that for all i :-

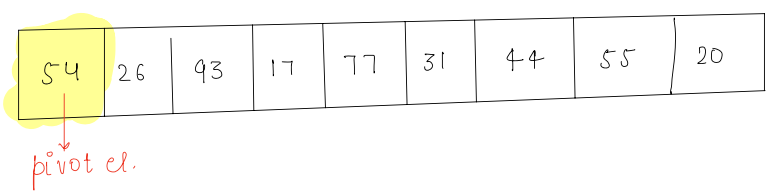
1) if A[i] < p, then it should be present on left side.

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

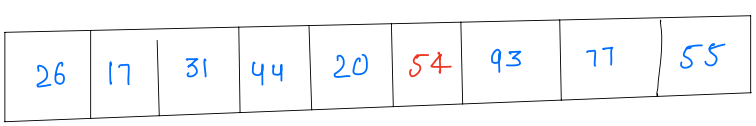
Note: All elements are distinct



if :-

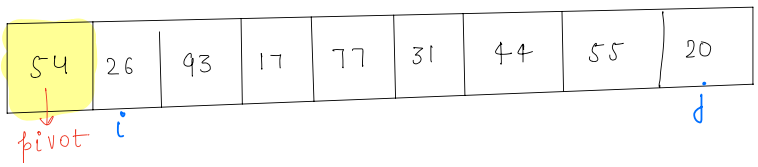


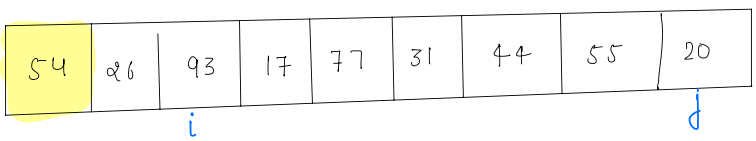
After partioning :-



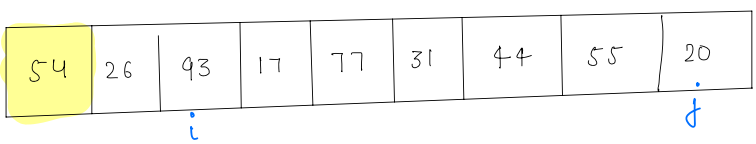
Approach

ip :-

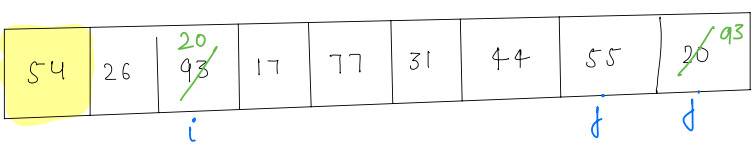




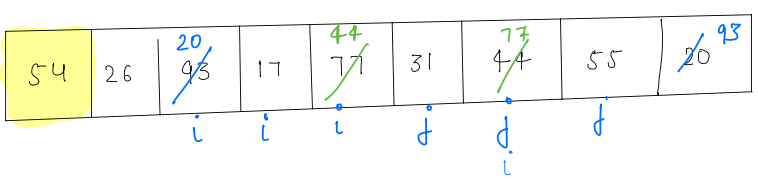
26 < 54. i++
93 > 54 stop



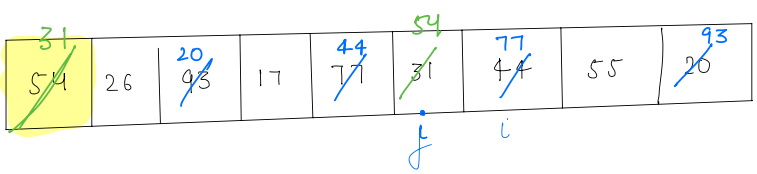
20 < 54. stop



swap (20, 93)



77 > 54 stop
55 > 54.
44 < 54.



Last step = swap (A, 0, j)

Algorithm

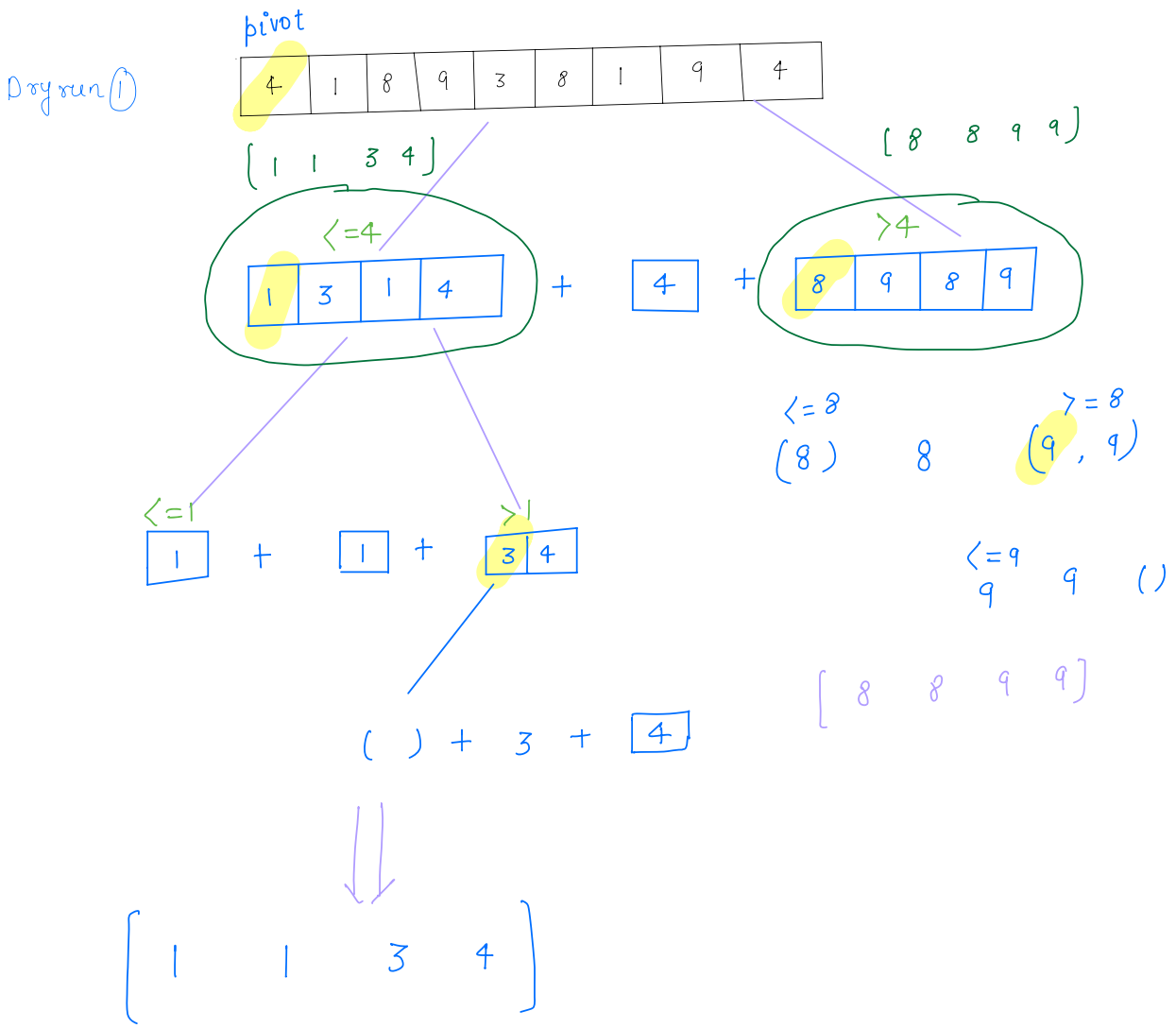
```
                                        0          A.length-1.
void partition(A[], start, end) {

    pivotValue = A[start];
    i = start+1;
    j = end;
    while(i<=j) {
        if(A[i] <= pivotvalue) {
            i+=1;
        } else if(A[j] > pivotvalue) {
            j--;
        } else {
            swap(A, i, j);
            j--;
            i++;
        }
    }
    swap(A, start, j);
}
```

TC: O(n)

SC: O(1)

Quick sort

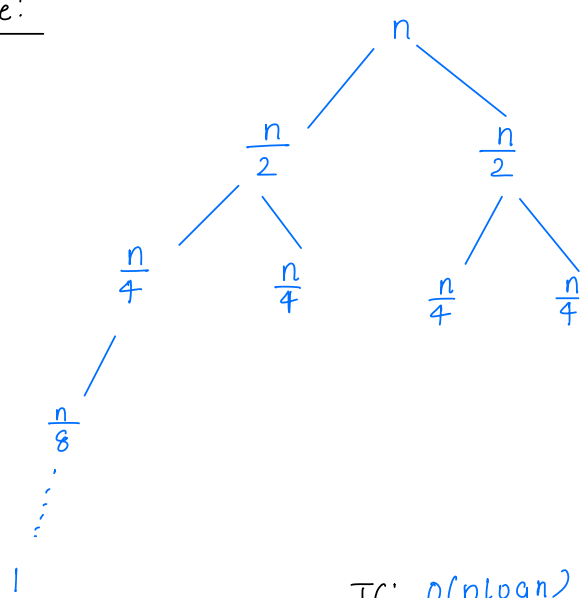

Dry run (1)
pivot
4 1 8 9 3 8 1 9 4
[1 1 3 4]
[8 8 9 9]
<=4
1 3 1 4
+
4
+
>4
8 9 8 9
<=1
1
+
1
+
>1
3 4
<=8
(8)
8
>=8
(9 , 9)
<=9
9
9
( )
( ) + 3 + 4
[8 8 9 9]
[1 1 3 4]

Algorithm

```
void quicksort(int[] A, int s, int e) {
    if( s>=e) {
        return;
    }
    int pivotIdx = partition( A, s, e);
    quicksort(A, start, pivotIdx-1);
    quicksort( A, pivotIdx+1, end);
}
```

T.C and S.C analysis

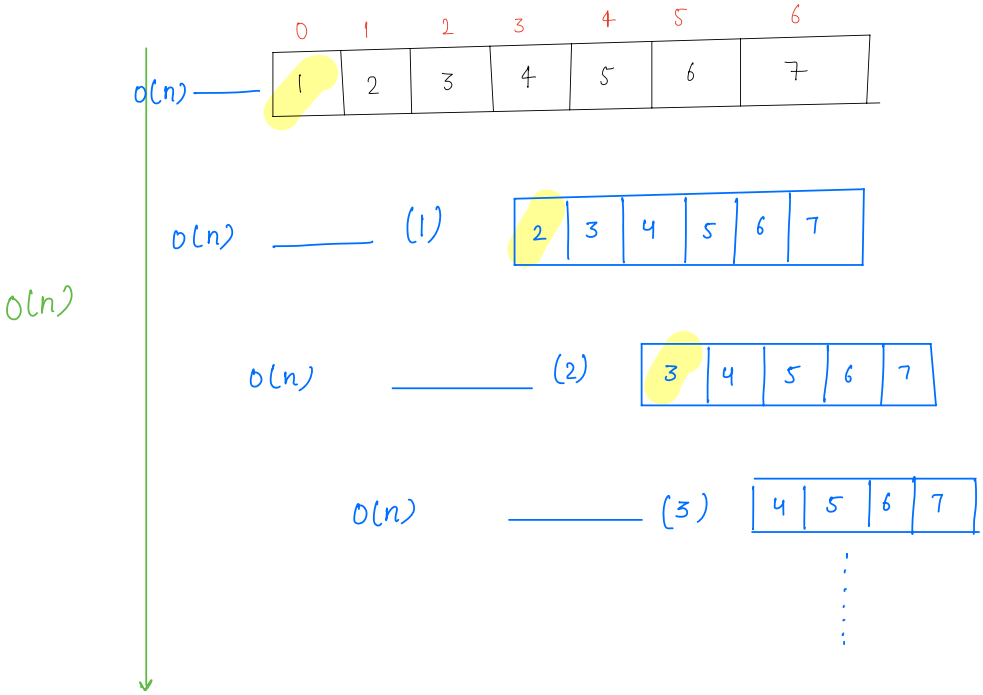
Best case:



TC: O(nlogn)

SC: O(logn)

Worst case time complexity



TC: $O(n^2)$ SC: $O(n)$

↓

sorted array.

Randomised Quick sort

→ Randomised quick sort helps us to get away with worst case T.C.

Why picking random el. helps?

→ Randomised Q.s helps us to get away with worst case T.C.

→ The odds | prob of choosing a random el as min is very very very --- low.

Example:

n el, prob that a random el is min ⇒ 1/n

prob. that again next time, random el is min ⇒ 1/n-1

Then → $\frac{1}{n-2}$

Then → $\frac{1}{n-3}$

⋮ ⋮

$\frac{1}{n} * \frac{1}{n-1} * \frac{1}{n-2} * \frac{1}{n-3} \cdots\cdots$

$\boxed{\frac{1}{n!}}$ → very very very ------ ∞ small.

Break: 8:23 – 8:33

Qu: Given A[n], sort the array in ascending order on the basis of count of factors. If count of factors are equal, then sort the elements on basis of their magnitude.

ip :-

0	1	2	3	4
9	3	10	6	4
3	2	4	4	3

factors

9: 1, 3, 9
3: 1, 3
10: 1, 2, 5, 10
6: 1, 2, 3, 6
4: 1, 2, 4

op:

3	4	9	6	10

Comparator

Asc

```
int compare(a, b) {
    if(a < b) {
        return -1;
    } else if(a > b) {
        return 1;
    }
    return 0;
}
```

Dsc.

```
int compare(a, b) {
    if(a < b) {
        return 1;
    } else if(a > b) {
        return -1;
    }
    return 0;
}
```

Code:

```
class factorComparator implements Comparator<Integer> {
    @override
    public int compare(int a, int b) {
        int fa = countfactors(a);
        int fb = countfactors(b);
        if (fa < fb) {
            return -1;
        } else if (fa > fb) {
            return 1;
        } else {
            if(a<b){
                return-1;
            } else if(a>b){
                return1;
            }
            return 0;
        }
    }
}

void sortfactors(List<Integer> arr) {
    Collections.sort(arr, new factorComparator());
}
```

Qu Given points[], where points[i] = [xi, yi] represents a point on the x-y plane and integer B, return B closest points to origin (0, 0).

The distance b|w 2 points on x-y plane is the euclidian distance $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

You may return answer in any order.

Example 1:

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

Distance b|w (1,3) and origin (0,0) $\Rightarrow \sqrt{(1-0)^2 + (3-0)^2} = \sqrt{10}$

Distance b|w (-2,2) and origin (0,0) $\Rightarrow \sqrt{(-2-0)^2 + (2-0)^2} = \sqrt{8}$

Ans = (-2,2).

Example 2: points = { (3, 3) (5, -1) (-2, 4) } B = 2.

Distance b|w (3,3) and origin $\Rightarrow \sqrt{(3-0)^2 + (3-0)^2} = \sqrt{18}$

Distance b|w (5,-1) and origin $\Rightarrow \sqrt{(5-0)^2 + (-1-0)^2} = \sqrt{26}$

Distance b|w (-2,4) and origin $\Rightarrow \sqrt{20}$

Ans = (3,3) and (-2,4)

Approach

points = { (3, 3) (5, -1) (-2, 4) } B = 2

↓ sort on euclidian distance

{ (3,3) (-2,4) (5, -1) }

Any

```
int compare( Points a, Points b ) {
    double d1 = Math.sqrt( a.x * a.x + a.y * a.y);
    double d2 = Math.sqrt( b.x * b.x + b.y * b.y);
    if( d1 < d2){
        return -1;
    } else if ( d1 > d2) {
        return 1;
    }
    return 0;
}
```

Qu. Given a list of non-negative integer nums, arrange them such that they form largest number and return it.

Since the result may be very large, so you need to return a string instead of an integer.

10	2

$\Rightarrow$ "210"

3	30	34	5	9

$\Rightarrow$ 3430953 ✗

9534330 ✓

Should we sort the numbers in descending order and append them?

3	30	34	5	9

↓ desc

34, 30, 9, 5, 4 $\Rightarrow$ 3430954 (wrong)

---

34 < 30 $\Rightarrow$ false

9 > 34 $\Rightarrow$ false

"Ayush" > "Ayuxh" $\Rightarrow$ false

"9" > "34" $\Rightarrow$ true.

"34" > "30" $\Rightarrow$ true.

↑↑ ↑↑

3 < 30 $\Rightarrow$ false

"3" > "30" $\Rightarrow$

3	30	34	5	9

9534 3 30

9534 30 3

Approach

```
int compare(int a, int b) {
    if(a<b) {
        return -1;
    } else if( a>b) {
        return 1;
    }
    return 0;
}
```

a = "3" (string)
b = "30" (string)

a+b → 330
b+a → 303

a b

string a = "he"
string b = "llo"
print (a+b) ⟹ hello

```
String ab = String.valueOf(a) + String.valueOf(b);
String ba = String.valueOf(b) + String.valueOf(a);
int cmp = ab.compareTo(ba);
if (cmp < 0) {
    // ab < ba
    return 1;
}
else if (cmp > 0) {
    // ab > ba
    return -1;
}
return 0;
}
```

"Ay".compareTo"By"
↳ negative value

"Ay".compareTo"Ax"
↳ +ve value

"Ay".compareTo"Ay"
0

Thankyou :)