20 May 2024 21:34

Sorting

→ Ascending order → small to big

→ Descending order → Big to small

O(n)

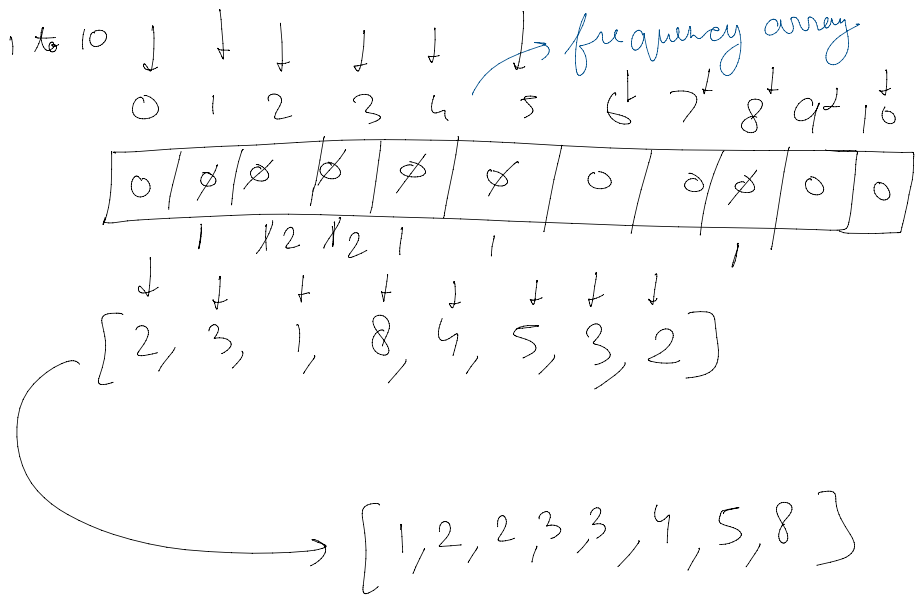
1) Count sort

O(n log n)

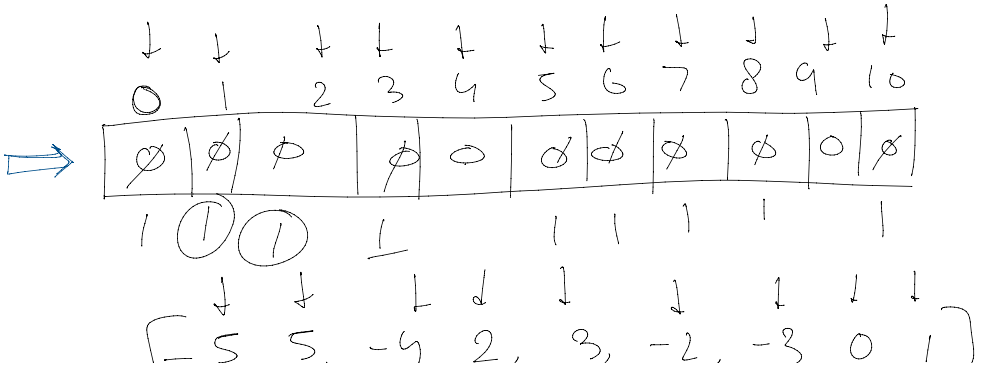
1) Merge → O(n log n), O(n)
2) Quick → O(n²), O(1)

O(n²)

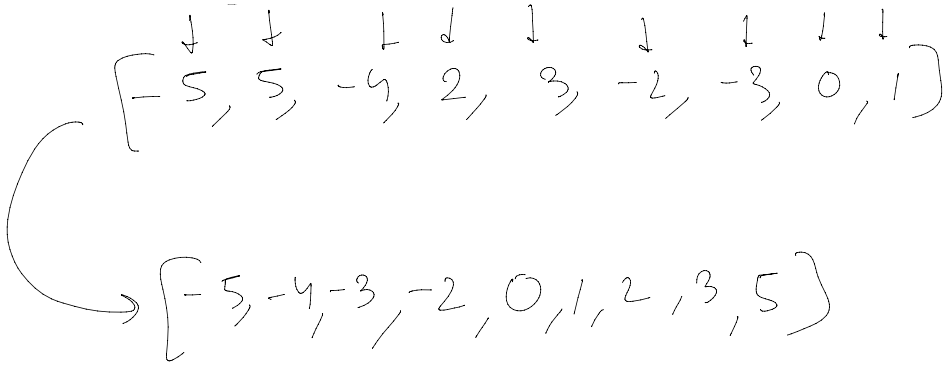
1) Bubble sort
2) Insertion sort
3) Selection sort



-5 to 5 → -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5



arr[val+5]++



INT_MIN to INT_MAX

Inbuilt: T: $O(n \log n)$ + S: $O(1)$

Anagram

abcd cbda ✓

abcd ccbda ✗

aabcd abbcd ✗

Lower case alphabets → 26

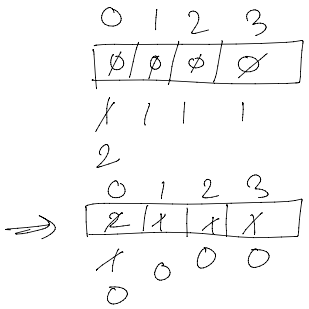
abcd

assume there are just 4 alphabets.

a = a b a d c b = c d a a b

0 1 2 3

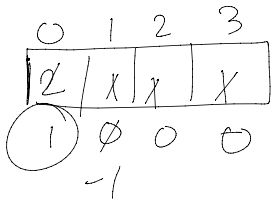
$\rightarrow O(n)$

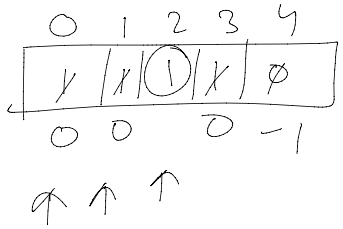


T: O(n)

S: O(1)

a => a b c a d b => a b b c d





a b c d a b d e x

1) Creat a frequency array of size 26 initialized 0s.

2) Iterate the string a(i):

freq[a[i] - 'a']++

3) Iterate the string b(i):

freq(b[i] - 'a')--

4) Iterate the freq array (i):

4) Iterate the freq array : (1).

a) If freq[i] ≠ 0, return false

5) return true.

Chocolate distribution problem

[3, 4, 1, 9, 56, 7, 9, 12] M = 5

56 - 1 ⇒ 55

12 - 1 ⇒ 11

I

1) Find all possible distributions.

2) Find min(max_choc - min_choc).

(10, 100, 105, 60, 106, 107, 108) ③

p1

(10, 60, 100, 105, 106, 107, 108)
0, 1, 2, 3, 4, 5, 6

p2

90

45

r

②

6

2/(2)

1) Sort the input array.

2) $P_1 = 0$, $P_2 = k-1$ T: O(n log n)

3) ans = INT_MAX S: O(1)

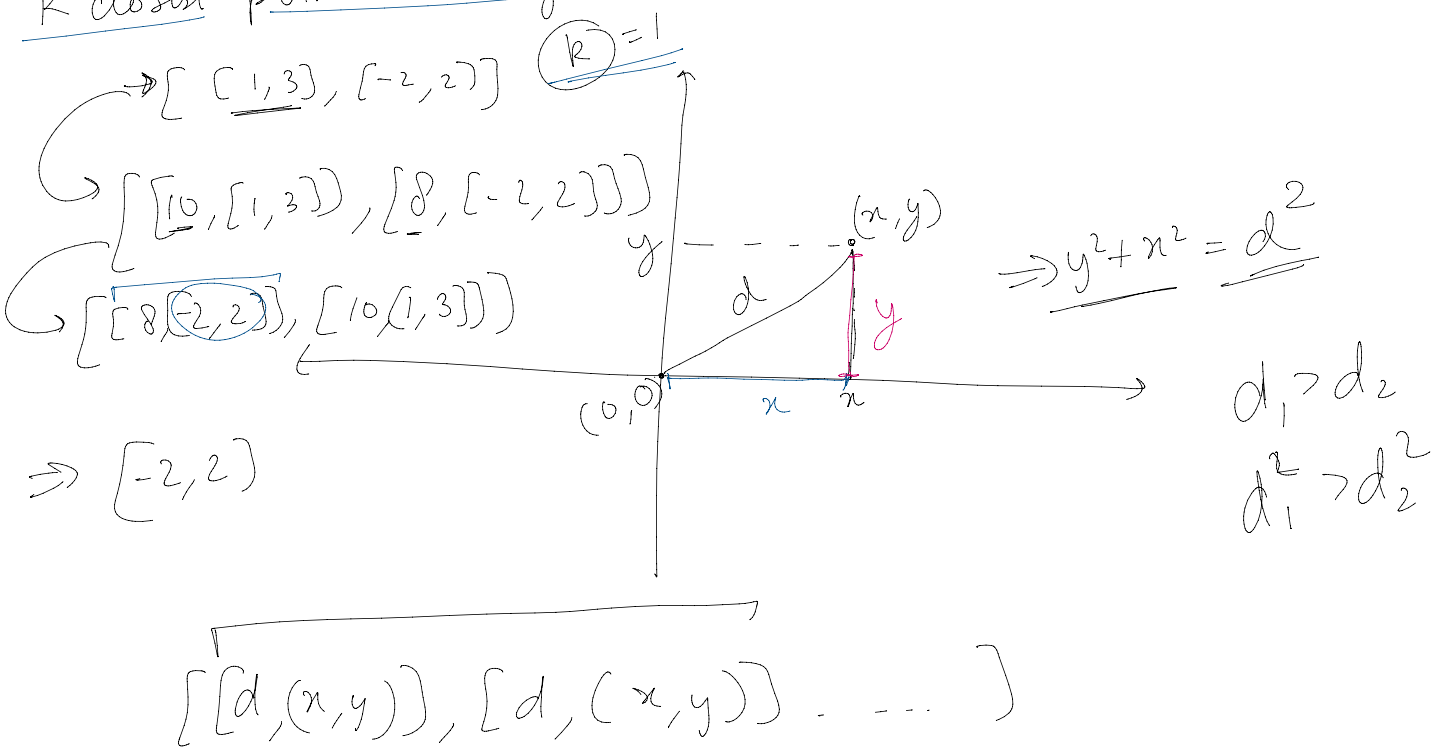
4) Iterate the array till $P_2 < n$:

1) ans = min(ans, (arr[P_2] − arr[P_1]))

2) P_1++ ; P_2++

5) return ans.

K closest points to origin



1) Creat a vector of pair.

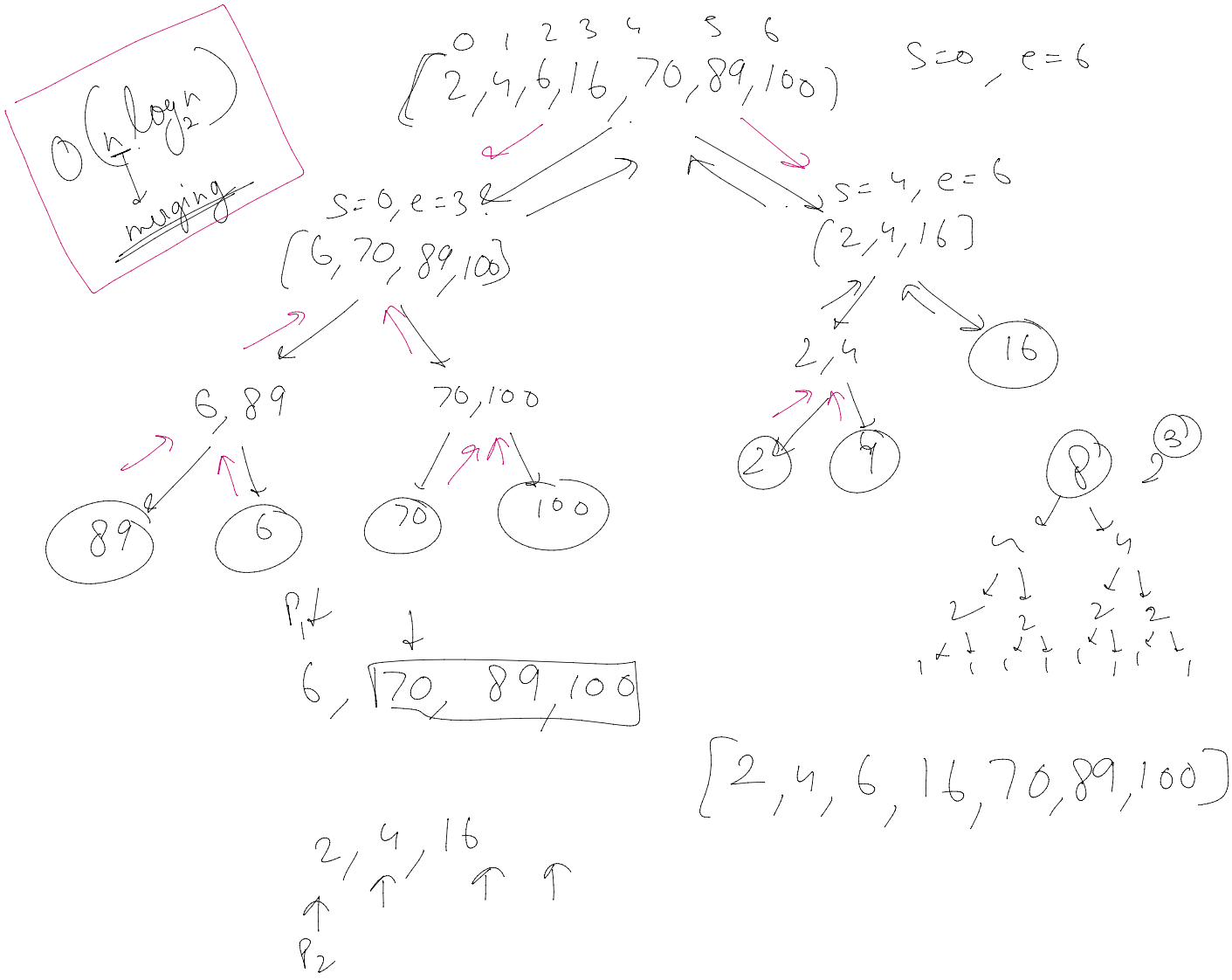
2) Store the information in this format : $[[x_1^2+y_1^2, [x_1, y_1]], [x_2^2+y_2^2, [x_2, y_2]] \ldots]$

3) Sort this structure

4) Take the first k values.

Merge sort T: $O(n \log n)$ S: $O(n)$

[89, 6, 70, 100, 2, 4, 16]



How to merge two sorted arrays. → $O(n)$ → $O(n)$

How to merge two sorted arrays:-

> We are given two sorted arrays: arr1, arr2.
n m

1) create an array of size (n+m). k=0. (arr3)

2) $P_1=0$, P_2 0. P_1 runs on arr1 and P_2 runs on arr2.

3) while ($P_1 < n$ && $P_2 < m$):

a) If (arr1[P_1] < arr2[P_2]):

i) arr3[k++] = arr1[P_1++]

b) Else:

i) arr3[k++] = arr2[P_2++]

4) while ($P_1 < n$):

arr3[k++] = arr1[P_1++]

5) while ($P_2 < m$):

arr3[k++] = arr2[P_2++]

6) return arr3

mergeSort(arr, s, e) {

```
if (s>e) {
    return;
}
```

→ m = s + (e-s)/2;

→ mergeSort(arr, s, m);

→ mergeSort(arr, m+1, e);

→ merge(arr, s, e);

}

Count inversion

```
  0  1  2  3  4
[2, 4, 1, 3, 5]
```

a[i] > a[j] and i < j

4 > 1, 1 < 2 → (4,1)

(4,3)

(2,1)

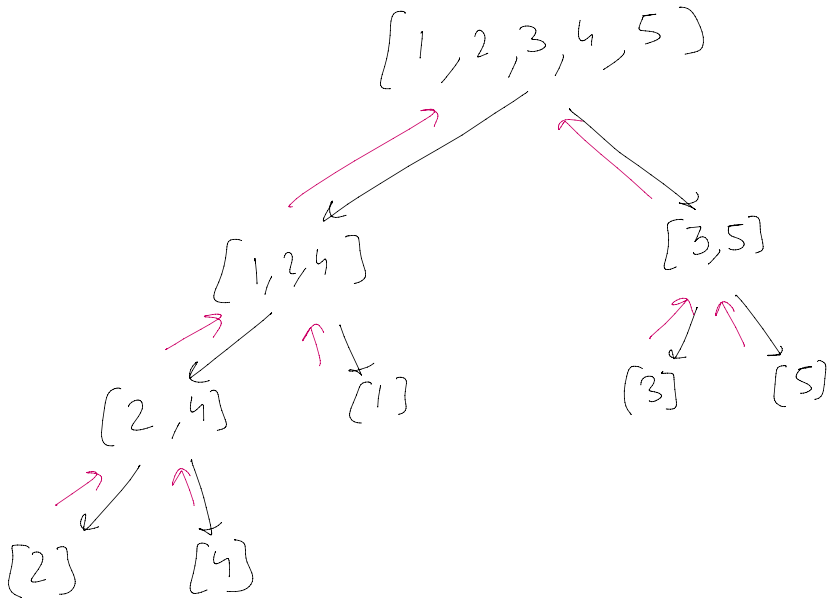
③

I

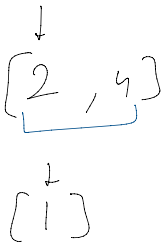
```
arr=0;
for(int i=0; i<n; i++) {
    for(int j=i+1; j<n; j++) {
        if(arr[i] > arr[j]) {
            arr++;
        }
    }
}
```

3

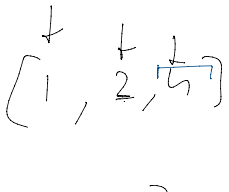
return vars;



(1,2,4)



3



1,2,3,4,5