

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
<u>Quicksort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
<u>Mergesort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Timsort</u>	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Heapsort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(1)$
<u>Bubble Sort</u>	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Insertion Sort</u>	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Selection Sort</u>	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Tree Sort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(n)$
<u>Shell Sort</u>	$\Omega(n \log(n))$	$\Theta(n(\log(n))^2)$	$O(n(\log(n))^2)$	$O(1)$
<u>Bucket Sort</u>	$\Omega(n+k)$	$\Theta(n+k)$	$O(n^2)$	$O(n)$
<u>Radix Sort</u>	$\Omega(nk)$	$\Theta(nk)$	$O(nk)$	$O(n+k)$
<u>Counting Sort</u>	$\Omega(n+k)$	$\Theta(n+k)$	$O(n+k)$	$O(k)$
<u>Cubesort</u>	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$

```

1 **Question**
2 https://leetcode.com/problems/rank-teams-by-votes/

```

C++

TC: $O(n \log n)$

SC: $O(1)$

```

string rankTeams(vector<string>& votes) {
    int m = votes.at(0).size();
    //  "ABC", "ACB", "ABC", "ACB", "ACB"
    //  m=3
    //  [  [2,0,0],[0,1,1],[0,1,1],[0,0,0],[0,0,0],...[0,0,0]  ]
    //      0->A    1->B    2->C    3->D.....                25->Z
    vector<vector<int>> mp(26, vector<int>(m,0)); //  O(26*26) =
O(1)
    for(string& voter : votes){ // O(N * M) n = no. of votes, m
= number of teams
        for(int i = 0 ; i < voter.length() ; i++){
            mp[voter[i]-'A'][i]++;
        }
    }

    vector<pair<vector<int>, char>> table(26); //to store the ran
k chart against each char
    for(int i = 0 ; i < 26 ; i++){ // O(26) -> O(1)
        table[i] = {mp[i],(i+'A')};
    }

    sort(table.begin(),table.end()).rll(pair<vector<int>,char>& a.

```

JAVA

```

class Solution {
    public String rankTeams(String[] votes) {
        HashMap<Character, int[]> map = new HashMap<>();

        int l = votes[0].length();
        for(String vote : votes){
            for(int i = 0; i < l; i++){
                char c = vote.charAt(i);
                int[] arr = map.get(c);
                if(arr == null){
                    arr = new int[l];
                    map.put(c, arr);
                }
                arr[i]++;
            }
        }

        // Sort teams based on their ranks
        // ...
    }
}

```

Python

```

class Solution:
    def rankTeams(self, votes: List[str]) -> str:

        ranks = [[([0]*len(votes[0])) + [team] for team in votes[0]]

        idx_map = {}
        for i, team in enumerate(votes[0]):
            idx_map[team] = i

        for vote in votes:
            for idx, team in enumerate(vote):
                i = idx_map[team]
                ranks[i][idx] += 1

class Rank:
    def __init__(self, rank):
        self.rank = rank[:-1]
        self.name = rank[-1]

    def __lt__(self, other):
        print(self.rank, other.rank)
        for i,j in zip(self.rank, other.rank):
            if j != i:
                return i < j

        return self.name > other.name

ranks = [Rank(rank) for rank in ranks]

out = ""
for r in sorted(ranks, reverse=True):
    out += r.name

return out

```

In []:

1

Question<https://leetcode.com/problems/h-index/> (<https://leetcode.com/problems/h-index/>)**Brute Force Solution**TC: $O(n^2)$ SC: $O(1)$ **class Solution:****def** hIndex(self, citations: List[int]) -> int:**#** // [7,7,100,500,7]**#** // 0 1 2 3 4 5**#** // [4,4,4,4,4,7]**#** // 0 1 2 3 4 5?x**#** // [3,0,6,1,5]

res = 0

for h **in** range(0, len(citations)+1):

count = 0

for citation **in** citations:**if** citation >= h:

count +=1

if count < h:**break**

res=h

return res**Python**TC: $O(N \log N)$ SC: $O(1)$

```
int hIndex(vector<int>& citations) {
    int ans=0;
    sort(citations.begin(), citations.end());
```

PythonTC: $O(N \log N)$ SC: $O(1)$

```
def hIndex(self, citations: List[int]) -> int:
    citations.sort(reverse = True)
    res = 0
    n = len(citations)
    for i in range(n):
        if citations[i] >= i+1:
            res += 1
    return res
```

PythonTC: $O(N \log N)$ SC: $O(1)$

```
class Solution:
    def hIndex(self, citations: List[int]) -> int:
        citations.sort(reverse = True)

        # [3,0,6,1,5]
        # [6,5,3,1,0]
        # 0 1 2 3 4
        # 1 2 3 4 5
        # []
        res = 0
        n = len(citations)
        for i in range(n): # i=0...4
            if citations[i] < i+1:
                break
            res = i + 1

        return res
```

JavaTC: $O(N)$ SC: $O(N)$

```

public int hIndex(int[] citations) {

    int len = citations.length;

    int[] freq = new int[len+1];
    for(int citation: citations) {
        if(citation > len) {
            freq[len]++;
        } else {
            freq[citation]++;
        }
    }

    int totalCitations = 0;
    for(int citation = len; citation >= 0; citation--) {
        totalCitations += freq[citation];
        if(totalCitations >= citation) {
            return citation;
        }
    }
}

```

In []:

1

```

1  **Radix sort**
2
3  [10, 102, 20, 24, 4, 104]
4
5  Step-1
6  0 -> 10,20
7  1
8  2 -> 102
9  3
10 4 -> 24, 4, 104
11 5
12 6
13 7
14 8
15 9
16 output: 10,20,102,24,04,104
17
18 0 -> 102,04,104
19 1 -> 10
20 2 -> 20, 24
21 3
22 4 ->
23 5
24 6
25 7
26 8
27 9
28 102, 04,104, 10, 20, 24
29
30 0 -> 004,010,020,024

```

```
31 1 -> 102,104
32 2 ->
33 3
34 4 ->
35 5
36 6
37 7
38 8
39 9
40 4,10,20,24,102,104
41
42
43 \[10, 102, 24, 20, 4, 104\]
44
45 0 -> 010,024,020,004
46 1 -> 102,104
47 2 ->
48 3
49 4 ->
50 5
51 6
52 7
53 8
54 9
55
56 10,24,20,4,102,104
```

In []:

1

Radix sort stability

[10, 102, 24A, 4, 24B, 104]

Step-1: ones

0 -> 10

1 ->

2 -> 102

3

4 -> 24A, 4, 24B, 104

5

6

7

8

9

10, 102, 24A, 4, 24B, 104

Step-2: tens

0 -> 102,04, 104

1 -> 10

2 -> 24A,24B

3

4 ->

5

6

7

8

9

102,04,104,10,24A, 24B

Step-2: hundreds

0 -> 04,10,24A,24B

1 -> 102,104

2 ->

3

4 ->

5

6

7

8

9

4, 10, 24A, 24B, 102, 104

In []:

1

Bucket sort

```
[10, 104, 24, 20, 4, 102, 10000]
```

```
Range -> (4, 104)
```

```
Bucket size->10
```

```
Buckets = vector< vector<int> > ( ceil (max(arr)/ bucketSize))
```

```
data size = N
```

```
number of buckets = B
```

```
data in each bucket = N/B -> B * (N/B log N/B)
```

```
0 0-9 -> 4
```

```
1 10-19 -> 10
```

```
2 20-29 -> 24, 20
```

```
3 30-39
```

```
4 40-49
```

```
5 50-59
```

```
6 60-69
```

```
7 70-79
```

```
8 80-89
```

```
9 90-99
```

```
10 100-109 -> 104,102
```

```
Sort individual buckets using some sorting alog.
```

```
0 0-9 -> 4
```

```
1 10-19 -> 10
```

```
2 20-29 -> 20, 24
```

```
3 30-39
```

```
4 40-49
```

```
5 50-59
```

```
6 60-69
```

```
7 70-79
```

```
8 80-89
```

```
9 90-99
```

```
10 100-109 -> 102,104
```

```
Join all bucket
```

```
4, 10,20,24,102,104
```

In []:

1

In []:

1

Stability of merge sort depends on how you pick equal elements

```
[10, 2A, 20, 2B] [2A, 2B, 10, 20] (pick from 1st
if equal) [2B, 2A, 10, 20] (unstable if you pick from second)
[10, 2A] -> [2A, 10] [20, 2B] -> [2B, 20]
```

In []:

1

quick sort

Properties: inplace, not-stable

- TC
 - AVG = BEST = $O(N \log N)$
 - Worst = $O(N^2)$
- SC
 - AVG = $O(\log N)$
 - WORST = $O(N)$

```
void qsort(array, start, end) {  
    if end-start <=1  
        return  
  
    p = partition(array, start, end)  
  
    qsort(array, start, p)
```

In []:

1