# Sorting

Content

→ Understanding Sorting

→ few problem on Sorting

→ 1 sorting algo

→ Comparator function

## Sorting ?

Arranging data in increasing / decreasing order based on some parameter.

eg 2 4 7 11 15 : sorted in Asc, parameter = array values

eg 15 9 6 2 0 : sorted in Desc, par = array values

eg 1 2 3 7 4 9 6 : sorted in Asc based on # factors.

#factors 1 2 2 2 3 3 4 par = # of factors of array values

Inbuilt library

↳ sort(), in every language sort() is present.

↳ How()? logic? ⇒ {In Advanced Batch}

TC: $O(n \log n)$ $n$ = no. of elements to sort

## Question 1 : Elements Removal

Given N elements, at every step remove an array element.

Cost of remove element : sum of all elements present in array

find min. cost to remove all elements.

Note: first calculate the cost, then remove the element.

eg $a[3] = \{2 \quad 1 \quad 4\}$

| | Cost |
|---|---|
| remove 2 : | $2+1+4 = 7$ |
| remove 1 : | $1+4 = 5$ |
| remove 4 : | $4$ |

total cost = $7+5+4 = 16$

| | cost |
|---|---|
| remove 1 : | 7 |
| remove 2 : | 6 |
| remove 4 : | 4 |

total cost = 17

| | cost |
|---|---|
| remove 4 : | 7 |
| remove 2 : | 3 |
| remove 1 : | 1 |

total cost = 11 ✓

eg $a[4] = \{3 \quad 6 \quad 2 \quad 4\}$

| | cost |
|---|---|
| remove 6 : | 15 |
| remove 4 : | 9 |
| remove 3 : | 5 |
| remove 2 : | 2 |

total cost = 31

Observation : deleting element by element in decreasing order gives the min. cost ?

$a[4] = \{\overset{0}{a}, \overset{1}{b}, \overset{2}{c}, \overset{3}{d}\}$

remove a : $a+b+c+d$
remove b : $b+c+d$
remove c : $c+d$
remove d : $d$

$\xrightarrow[\text{cost}]{\text{total}}$ $a + 2b + 3c + 4d$

$a \to$ max, $2b \to$ 2nd max, $3c \to$ 3rd max, $4d \to$ min

## Code

```
int minCost (a[]) {
    n = a.length
    Sort(a , DESC);  → sort a[] in desc order
                       ↳ TODO in your own language
    ans = 0
    for (i=0; i<n; ++i) {
        ans += a[i] × (i+1)
    }
    return ans
}
```

TC: ~~O(N)~~

TC: O(NlogN + N)

: O(NlogN)

SC: O(1)

**Dry run:**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| a[4] = | 3 | 6 | 2 | 4 |
| Sort(a) = | 6 | 4 | 3 | 2 |

ans = 6×1 + 4×2 + 3×3 + 2×4

= 6 + 8 + 9 + 8

= 31

Question 2 : Noble Integers { Elements are unique }

Given N elements, calculate no. of noble integers.

An element in a[] is said to be noble iff

{no. of elements < ele = ele itself}

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| eg | -1 | -5 | 3 | 5 | -10 | 4 |
| #lesn | 2 | 1 | 3 | 5 | 0 | 4 |

Ans = 3

| | | | | | |
|---|---|---|---|---|---|
| eg | -3 | 0 | 2 | 5 | |
| #lesn | 0 | 1 | 2 | 3 | |

Ans = 1

Brutefore :

for every element, iterate & get # ele < ele and compare with ele. itself.

```
int ans=0
for(i=0; i<n; ++i) {
    count=0   // # ele < a[i]
    for(j=0; j<n; ++j) {
        if(a[j] < a[i])
            ++count
    }
```

TC : O(N²)

SC : O(1)

```
if (a[i] == count)
    ++ans
}
```

**Idea:** Sort the array in asc. order

sorted(a) : $\underbrace{a[0]\ a[1]\ a[2] \ldots\ldots a[i-1]}$ $a[i]$ $a[i+1] \ldots$

↳ all these no. are less than $a[i]$

count $= i-1-0+1$
$= i$

$\Rightarrow$ if $(a[i] == i)$
then
$a[i]$ is noble

**Code**

```
int noble(a[]) {
    n = a.length
    Sort(a, ASC)  -> TODO
    ans = 0
    for (i=0; i<n; ++i) {
        if (a[i] == i)
            ++ans
    }
    return ans
}
```

TC: $O(N \log N)$

SC: $O(1)$

dry run:

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| | -1 | -5 | 3 | 5 | -10 | 4 |
| sort | -10 | -5 | -1 | 3 | 4 | 5 |

Ans = 3

# Question 3 :

Count Noble elements : { Elements can repeat }
↓
Duplicate values

We can solve using bruteforce :

TC: $O(N^2)$ SC: $O(1)$

eg

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| | 0 | 2 | 2 | 3 | 3 | 6 |
| #less | 0 | 1 | 1 | 3 | 3 | 5 |

Ans = 3

Above Indexing approach will not work

eg

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| | 10 | 1 | 1 | 1 | 4 | 4 | 4 | 7 | 10 |
| #less | 0 | 1 | 1 | 1 | 4 | 4 | 4 | 7 | 8 |

Observation 1 : index = #less is correct for only the first occurance of every element

Observation 2 : if one element is noble, all occurances are noble.

Idea: if element comes for the first time
↳ if (a[i] != a[i-1])

count of ele. less than a[i] = i

if element repeats → if (a[i] == a[i-1])

count of ele. less than a[i] will be same as prev. one.

Code

```
int noble (a[]) {
    n = a.length
    ans = 0
    sort (a, ASC); → TODO
    if (a[0] == 0) { ans = 1 }
    len = 0 // # ele < a[i]
    for (i = 1; i < n; ++i) {
        if (a[i] != a[i-1]) { // a[i] coming for first time
            len = i
        }
        if (a[i] == len)
            ++ans
    }
    return ans
```

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

3

$l \equiv len$

dry run

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 0 | 0 | 2 | 2 | 5 | 5 | 5 | 5 | 8 | 8 | 10 | 10 | 10 | 14 |
| l=0 | l=0 | l=2 | l=2 | l=4 | l=4 | l=4 | l=4 | l=8 | l=8 | l=10 | l=10 | l=10 | l=13 |
| ans=1 | 2 | 3 | 4 | | | | | 5 | 6 | 7 | 8 | 9 | |

Ans = 9

## Sorting Algo

sort in asc

arr = { 3 8 6 2 4 }  (i=0, j=4)

iter 1 :

6 8 2 4 8

{ 3 6 2 4 8 }  (i=0, j=3) : 8 is at correct position

iter 2 :

2 6 4 6

{ 3 2 4 6 8 }  (i=0, j=2) : 6, 8 is at correct position

iter 3 : 2 3

{ 2 3 4 6 8 } : completly sorted

## Code

```
Sort (a[]) {
    n = a.length
    for (j = n-1; j > 0; --j) {
        for (i = 0; i < j; ++i) {
            if (a[i] > a[i+1])
                swap(a[i], a[i+1])
        }
    }
```

⇒ BUBBLE SORT

TC: $O(N^2)$

SC: $O(1)$

3

>

3

iterations : $(n-1) + (n-2) + \ldots \ldots + 1$

$$= \frac{n(n-1)}{2} = O(N^2)$$

For any type of sorting like sort based on # factors, we use comparator function.

```
for (j = n-1; j > 0; --j) {
    for (i = 0; i < j; ++i) {
        if ( compare(a[i], a[i+1]))
            swap(a[i], a[i+1])
```

just implement this function, to get your desired sorting.

}

}

In Java / JS

```
Comparator customComparator = (Integer a, Integer b) -> {
    // If you want a to come before b : return -1
    // If you want a & b are same : return 0
    // If you want b to come before a : return 1
}

Sort(a, customComparator);
```

Say we have to sort on asc. order of # of factors

```
Comparator customComparator (Integer a, Integer b) -> {
    fa = factors(a)  -> O(√N)
    fb = factors(b)
    if (fa < fb)  return -1
    else
        return 1
}
```

TC to sort :

$O(N \log N \times \sqrt{N})$

## In Python

```
def customComparator(a,b):
    // If you want a to come before b : return -1
    // If you want a & b are same : return 0
    // If you want b to come before a : return 1

a.sort(key=cmp_to_key(customComparator))
```

## In C/C++

```
bool customComparator(int a, int b) {
    // If you want a before b : return true
    // Else                   : return false
}
sort(a, customComparator)
```