Sorting: arrangement of data in some order on the basis of some parameter

2 3 9 12 17 19 — sorted in asc order (magnitude)

19 6 5 2 -1 -19 — sorted in desc order (magnitude)

	1	13	9	6	12
count of factors	1	2	3	4	6

sorted in asc order (count of factors)

(Advanced)
X how to sort?

Why sorting? searching becomes easier

library / inbuilt function → sort()

↓

TC : $O(n \log_2 n)$

- $O(n^2)$ / $O(n^3)$

  ↓

  $O(n \log n)$ / $n^2 \log n$ → Always try to think about sorting

- $O(n)$ X Sorting

  ↓

  $O(\log n)$

1. Given an array of N integers, we've to delete all elements of the array. Before deleting an element, pay cost = sum of elements in the array (at that point). Find min cost.

Distinct elements

Ex [2 1 4]

delete 1	2+1+4 = 7
delete 2	2+4 = 6
delete 4	4 = 4
	Total cost = 17

delete 4	2+1+4 = 7
delete 2	2+1 = 3
delete 1	1 = 1
	11 ↗ min cost

Ex [4 6 1]

delete 6	4+6+1 = 11
delete 4	4+1 = 5
delete 1	1 = 1
	17

Ex [3 5 1 -3]

delete 5	3+5+1+(-3) = 6
delete 3	3+1+(-3) = 1
delete 1	1+(-3) = -2
delete -3	-3 = -3
	2

[a b c d]

delete a → a+b+c+d
delete b → b+c+d
delete c → c+d
delete d → d

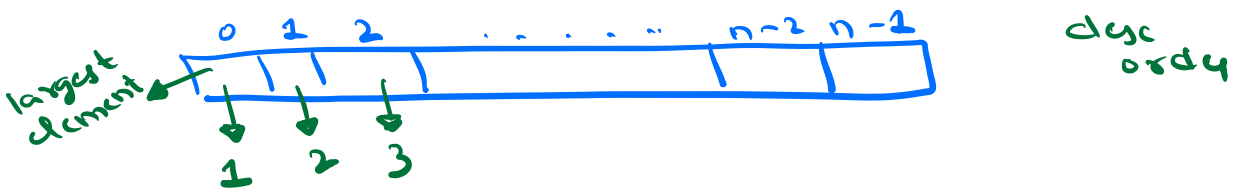
a+2b+3c+4d → min. cost

largest element

$a > b > c > d$

Start removing from largest end

- sort(arr) → TC: O(n log n)



i → (i+1) times

1) sort data in desc ordn → n log n
2) cost = 0

```
for (i = 0 ; i < n ; i++) {
    cost += a[i] * (i+1) ;
}
```

n

TC: O(n log₂ n)
SC: O(n)

sort() → SC O(n)
↓
depend on sorting algo

2. In an array of N element, find count of noble integers.

Distinct nos

A[i] is noble if
count of elements < A[i] = A[i]

	0	1	2	3	4	5
Ex	1	-5	3	5	-10	4
count	2	1	3	5	0	4

ans = 3

	0	1	2	3
Ex	-3	0	2	5
	0	1	2	3

ans = 1

- A -ve element can't be noble.

count = a[i]

0 or +ve ≠ -ve

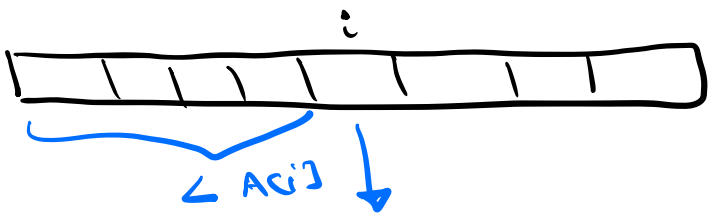
Idea 1: For every element, count smaller elements

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

TC: $O(n^2)$

SC: $O(1)$

Idea 2 sort data(asc)



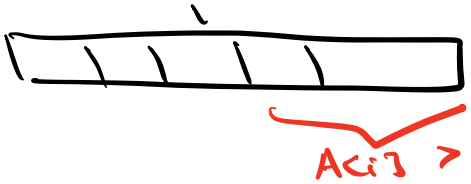
[0 i-1]
i-1-0+1

smaller elements → i

```
// sort data
ans = 0
for (i=0 ; i<n ; i++) {
    if (a[i] == i)
        ans++
}
```

count of smaller elements

desc



N-i-1
[i+1 N-1]
(N-1) - (i+1) + 1
= N-1-i-X+X

What if there are duplicates?

	0	1	2	3	4	
Ex	-10	1	1	3	100	ans = 3
count	0	1	1	3	4	

	0	1	2	3	4	5	6	7	8	
Ex	-10	1	1	2	4	4	4	8	10	ans = 5
count	0	1	1	3	4	4	4	7	8	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Ex	-3	0	2	2	5	5	5	5	8	8	10	10	10	14
count	0	1	2	2	4	4	4	4	8	8	10	10	10	13

ans = 7

```
if cur_elem != prev_ele
    count = index
```

1. sort the data → nlogn
2. ans = 0
   cnt = 0
   if (ar[0] == 0) → cnt of smaller elements
       ans++

TC : O(nlogn)

```
for (i = 1 ; i < n ; i++) {          → n
    if (a[i] != a[i-1])
        cnt = i;
    if (a[i] == cnt)
        ans++
}
```

	0	1	2	3	4	5	6	7	8
	-3	0	2	2	5	5	7	8	8
cnt = 0	1	2	2	4	4	6	7	7	
ans = 0	0	1	2	2	2	2	2	2	

	7	8	9
5	8	8	
4	8	8	

3. Sort data in asc order by count of factors
if count is equal, sort in asc order based on magnitude

	9	3	10	6	4	⇒	3	4	9	6	10
cnt	3	2	4	4	3		2	3	3	4	4

sort( ) → by default, asc | dsc order by magnitude

sort(_ , _ , _)
- _ : start index
- _ : end index
- _ : comparator fn (greater <int>) → rules of your sorting

* Custom sorting

x, y

x < y (asc)

↓

x should come first

x, y → 2 nos. are given

if (count_f(x) < count_f(y))
x should come first

count_f(x) > count_f(y)
y should come first

count_f(x) == count_f(y)
x <= y
x should come 1st
x > y
y should come 1st

x	y
5	5
cnt 2	2

int / bool

comp(int x, int y) → data type of element that you sort

if first argument (x) should come first in sorted data
↓
return true
else
return false

<

```
int cntx = count_factors(x);
int cnty = count_factors(y);

if (cntx < cnty)
    return true;
else if (cnty < cntx)
    return false;
else <
    if (x <= y)
        return true
    else
        return false;
```

$\sqrt{m}$ → max dc

for every comparison
↓
TC → comparator fn

>

Total time = no. of comparisons x time in comparator

- sort in desc order of magnitude

```
bool comp(int x, int y) {
    if (x < y)
        return false;
    else
        return true;
}
```

x y
2 < 5
5 comes first
return false

x y
5 > 2
return true

- array of strings
  sort acc. to length of strings in asc order

```
bool comp(string x, string y) {
    if (x.length < y.length)
        return true
    else
        return false
}
```

x y
cat camel
3 5
x to come 1st
return true

x y
monkey dog
6 3
return false

dog cat
lexicographical
↓
char vs char

Doubts

5 ← [2 5 10 20 30]

n → size of array

count_f(30)

for(i=1 ; i <= $\sqrt{30}$; i++)

count_f(m) → [1 $\sqrt{m}$]

O($\sqrt{m}$)

→ max element