Consistency is the true foundation of trust. Either keep your promises or do not make them.

— Roy T. Bennett

## Sorting

Arrangement of data in a particular order on the basis of some parameter.

2, 3, 9, 12, 17, 19 ⟶ sorted in asc. order based on value.

19, 6, 5, −1, −2, −9 ⟶ sorted in desc. order based on value.

1, 13, 9, 6, 12 ⟶ sorted in inc. order of the count of factors.

count of factors → 1 2 3 4 6

Why sorting? searching becomes easier.

~~How to sort?~~

inbuilt function to sort :-

sort(...) ⟶ O(n log n) T.C. ~~hiding~~

# $n^2$ ↓ n log n → Always try to think about sorting.

Q1) Array of n integers. You have to delete all the elements of the array. For each deletion, you have to pay a cost.

> cost to delete an element = Sum of all elements of the array at that moment

Find the min. cost possible.

Distinct Elements

A = {~~2~~ ~~1~~ ~~4~~}

delete 1 → 2+1+4 = 7
delete 4 → 2+4 = 6
delete 2 → 2 = 2
15

delete 4 → 2+1+4 = 7
delete 2 → 2+1 = 3
delete 1 → 1 = 1
11

Ans → 11.

A : [~~4~~ ~~6~~ ~~1~~]

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

{ a b c d }

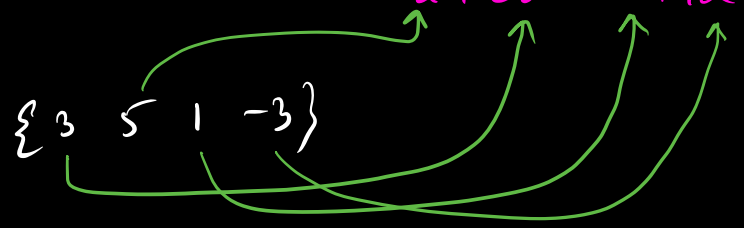
Remove a → a+b+c+d

Remove b → b+c+d

Remove c → c+d

Remove d → d

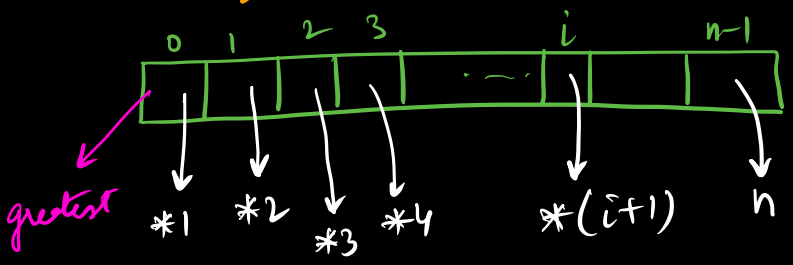
a + 2b + 3c + 4d




5 + 2*3 + 3*1 + 4(-3)

= 5 + 6 + 3 + (-12) = 2.

sort(arr) in desc. order.




sort the array in desc order → O(n log n)

```
ans = 0
for(i=0; i<n; i++){
    ans += arr[i] * (i+1);
}
return ans;
```

→ O(n)

O(n log n)

Q2) Find the count of noble integers in an array.

↳ ar[i] is noble if

count of elements smaller than ar[i] = ar[i]

---

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| | 1 | -5 | 3 | 5 | -10 | 4 |
| nr. of elements smaller than a[i] | 2 | 1 | 3 | 5 | 0 | 4 |

ans = 3.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| | -3 | 0 | 2 | 5 |
| | 0 | 1 | 2 | 3 |

ans = 1.

```
ans = 0
for (i → 0 to n-1) {
    cnt = 0
    for (j → 0 to n-1) {
        if (ar[i] > ar[j])
            cnt++;
    }
    if (ar[i] == cnt)
        ans++;
}
```

TC → $O(n^2)$.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| ar → | -3 | 0 | 2 | 5 |
| index → | 0 | 1 | 2 | 3 |
| count → | 0 | 1 | 2 | 3 |

if sorted in asc. order,

count = index

for A[i] i

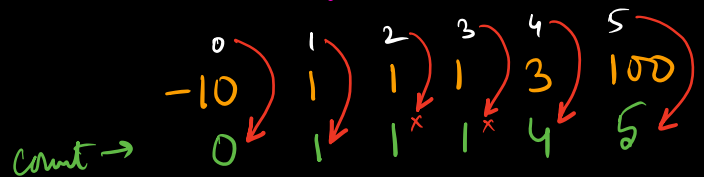
if (ar[i] == count)

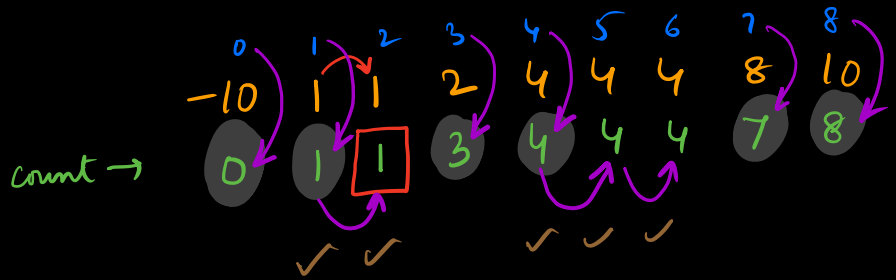
sort the data

```
int ans=0;
for(i=0; i<n; i++){
    if(ar[i] == i)
        ans++;
}
```

O(n log n) T.C.

What if there are duplicates?



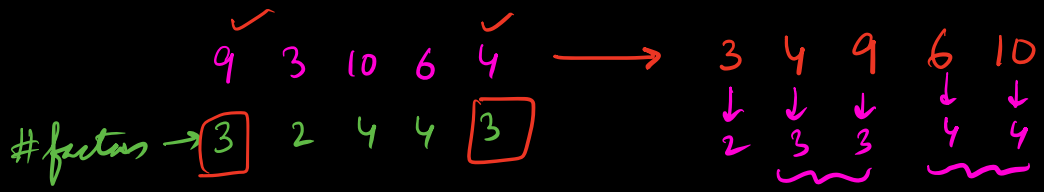



```
sort(arr)  // asc. order        O(n log n) TC
int ans = 0;
if(arr[0] == 0)
    ans++;
int count = 0;
for(i=1; i<n; i++){
    if(arr[i] == arr[i-1]) {
        // do nothing
    }
    else {
        count = i;
    }
    if(count == arr[i])
        ans++;
}
```

O(n) TC.

[Break till 10:50 PM]

# sort the data in the asc. order of the count of factors.
If no. of factors are equal, then sort them on the basis of increasing order of their magnitudes.




sort( ——— , cmp )
arr.
↳ comparator function.

cmp (x, y) ⟶ x ≤ y ⇒ put x before y
x > y ⇒ put x after y.

x, y

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 first.
x > y :
y should come first.

```
bool cmp(int x, int y){
    int cx = count_factors(x);
    int cy = count_factors(y);
    if(cx < cy)
        return true;
    if(cx > cy)
        return false;
    if(x < y)
        return true;
    return false;
}
```

if first argument (x) should come first, return true, else return false.

$O(\sqrt{A_{max}})$ T.C.

# Write a comparator fn. that you'll use to sort an array of strings in the decreasing order of lengths.

```
bool cmp(string x, string y){
    if(x.length() >= y.length())
        return true;
    return false;
}
```