Sorting :

- → Understand Sorting
- → Few Problems on Sorting
- → 1 Sorting Algo
- → Comperator

Sorting : Arranging data in increasing / decreasing order based on a parameter

Ex1: 2 4 7 11 15 : Inc, par = value

Ex2: 15 9 6 2 0 : Desc, par = value

Ex3: 1 2 3 7 4 9 6 : Sorted in Inc based on #factr

#factr: 1 2 2 2 3 3 4 factors

// Inbuilt library :

- sort( ), in every language sort() func present
- How? logic ? => { In Adr Content }

TC: N log N : N represents, # Elements to sort

18] Elements Removal

Given N elements, at every step remove an array element

Cost to remove element = Sum of array elements present in array

Find Min Cost to Remove all Elements?

Note: Add Cost of Removal first & then Remove it

Ex1: ar[3] = {2 1 4}

	Cost
remove 2 :	7
remove 1 :	5
remove 4 :	4
Total Co :	16

ar[3] = {2 1 4}

	Cost
remove 4 :	7
remove 2 :	3
remove 1 :	1
Total Co :	11

Ex2: ar[4] = {3 6 2 4}

	Cost
remove 6 :	15
remove 4 :	9
remove 3 :	5
remove 2 :	2
Total Cost :	31

Ex3: ar[3] = {6 -3 4}

	Cost
remove 6 :	7
remove 4 :	1
remove -3 :	-3
Total Cost :	5

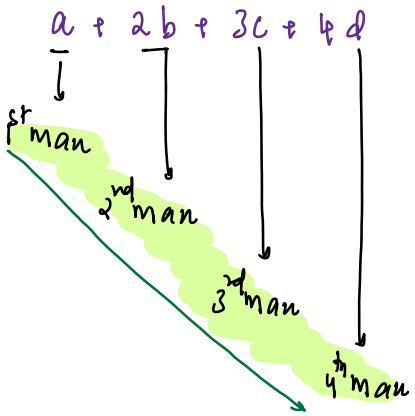
	0	1	2	3
ar[4] = {	6	4	3	2 }
	* 1	* 2	* 3	* 4

// Obs: deleting ele by ele in decreasing to get min cost?

Ex: ar[4] : { a b c d }

	Cost
remove a :	a + b + c + d
remove b :	b + c + d
remove c :	c + d
remove d :	d

→ Total Cost:




```
int MinCost(int arr[]) {
    int n = ar.length;
    Sort(ar, desc) // sort arr[] in desc order
                   └→ // Syntax, In your language of choice
    int c = 0;
    i = 0; i<n; i++) {
        // How many time ar[i] will in TS = i+1 times
        c = c + ar[i] * (i+1)
    }
    return c;
}
```

TC: (N log N + N) ⇒ O(N log N)

2Q) Noble Integers { Data is Distinct }

Given N array element, Calculate no: of Noble Integers

An element ele in ar[] is said to be Noble if

↳ { No of elements < ele = ele itself }

↳ count = ele

Obs1: Negative Can never be noble

Ex1: { -1 -5 3 5 -10 4 } // ⇒ ans: 3

#less : 2 1 3 5 0 4

Ex2: { -3 0 2 5 } // ⇒ ans = 1

(0 1 2 3)

less : 0 1 2 3

Ex3: { -10 -5 1 3 4 5 10 } // ⇒ ans = 3

(0 1 2 3 4 5 6)

#less : 0 1 2 3 4 5 6

// Idea1:

for every element in arr[]
iterate on arr[] & get
number of elements < than
& compare with ele itself

```
int ans=0
i=0; i<N; i++){
    int c=0 // ar[i] need elements <
    j=0; j<N; j++){
        if(ar[i] > ar[j]){
            c=c+1
        }
    }
    if(ar[i]==c){ ans=ans+1 }
}
```

TC: O(N²) SC: O(1)

Idea2:

Sort the array inc

#elements < ar[i] = i

```
int Noble(int ar[]){
    int n= ar.length
    Sort(ar[], inc) // Sort in inc
    int c=0;
    for(i=0; i<n; i++){
        if(ar[i]==i){
            c=c+1
        }
    }
    return c;
}
```

TC: (NlogN + N) = O(NlogN)

// 308) Noble Integers : { Data can repeat } → Idea 1 ✓ , Idea 2 *

Ex1:

	0	1	2	3	4	5
	0	2	2	3	3	6
#less :	0	1	1	3	3	5

ans = 3

Ex2:

	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

ans = 7

Ex3:

	-10	1	1	2	4	4	4	8	10
#less	0	1	1	3	4	4	4	7	8

ans = 5

ans = 7

Ex4:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	-3	0	2	2	5	5	5	5	8	8	10	10	10	14
#less:	0	1	2	2	4	4	4	4	8	8	10	10	10	13

Obs1: If element coming first time, if (ar[i] != ar[i-1])

Count of elements less than that will be, c = i

Obs2: If element repeats, if (ar[i] == ar[i-1])

Count of elements less than that will remain same

```
int NobleDistinct(int ar[]){
    int n = ar.length;
    int c = 0;
    Sort(ar, inc) // TODO
    int less = 0 // # no of elements less than ar[i]
    if(ar[0]==0) { c = 1} // noble number { Edge Case }

    for(int i=1; i<n; i++){
        // for ar[i] get no:of elements less than that?
        if(ar[i] != ar[i-1]){ // i!=0, ar[0] != ar[-1], error
            // ar[i] coming from time, # elements d = i
            less = i
        }
        else{ // Skip else
            // ar[i] is repeating, less won't change
            // Do nothing
        }
        if(less == ar[i]){
            c = c+1;
        }
    }
    return c;
}
```

TC: $O(N \log N + N) \approx O(N \log N)$

	0	1	2	3	4	5	6	7	6
arr[9] = {	1	1	2	2	4	4	5	7	7 }
	=	=							
c = 0	*	i = 0	i = 2	i = 2	i = 4	i = 4	i = 6	i = 7	i = 7
less = 0		c = 0	c = 1	c = 2	c = 3	c = 4	c = 4	c = 5	c = 6

ans = 6

Sorting algo: { Bubble Sort }

```
                0   1   2   3   4
// ar[] : { 3   8   6   2   4 }
```

Itr1: 3 6 2 4 8 : 8 at correct position

Itr2: 3 2 4 6 8 : 6, 8 at correct position

Itr3: 2 3 4 6 8 : Complete array sorted

Idea: Obs1

After 1 iteration ⟶ last 1 Element in Correct pos

After 2 iteration ⟶ last 2 Element in Correct pos

After 3 iterations ⟶ last 3 Element in Correct pos

After N-1 iterations ⟶ last N-1 Element in Correct pos ↕ All N Elements in Correct pos

In every iteration, → { // keep doing it N-1 times }
iterate from first to last Compare adj elements,
if, they are not in correct order we swap

Bubble Sort(int ar[]){ TC: O(N²) SC: O(1)

```
int n = ar.length;
for(int i = 1; i < n; i++){ // N-1 times
    for(int j = 0; j < n-1; j++){ // if j = N-1
        if(ar[j] > ar[j+1]){          ↳ ar[N-1] > ar[N]
            Swap ar[j] & ar[j+1]          Error
        }
    }
}
```

→ We can slightly optimize this

// Decreasing:

```
int n = ar.length;
for(int i = 1; i < n; i++){
    for(int j = 0; j < n-1; j++){
        if(ar[j] < ar[j+1]){
            Swap ar[j] & ar[j+1]
        }
    }
}
```

→ has power to decider order of data itself

// If we can, just change the if(condition), it allows our own, sorting order

→ Idea:

```
int n = ar.length;
for(int i = 1; i < n; i++){
    for(int j = 0; j < n-1; j++){
        if( comp(ar[j], ar[j+1]) )
            Swap ar[j] & ar[j+1]
    }
}
```

```
bool comp(int a, int b){
    // Just implement to
    get your own sorted order

    // If a should come before
    b, a b should come before a
}
```

Comparator 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(ar, customComparator)
```

Comparator in Python

```
def compare(a, b):
    if a should come before b : return -1
    if a & b same             : return 0
    if b should come before a : return 1
ar.sort( key = cmp_to_key(compare))
```

Comparator in C++

```
bool comp(a, b){
    if a should come first : return True
    else return False
};
Sort(ar, comp)
```

// Given N, Element sort them in increasing order of their {No: of factors}

If 2 elements have same no: of factors, elements with less value should come first

Note: → { No Extra Space }

ar[ ] ⇒ { 9 3 4 8 16 37 6 13 15 }

#fac 3 2 3 4 5 2 4 2 4

Correct order: { 3 13 37 4 9 6 8 15 16 }

Idea:

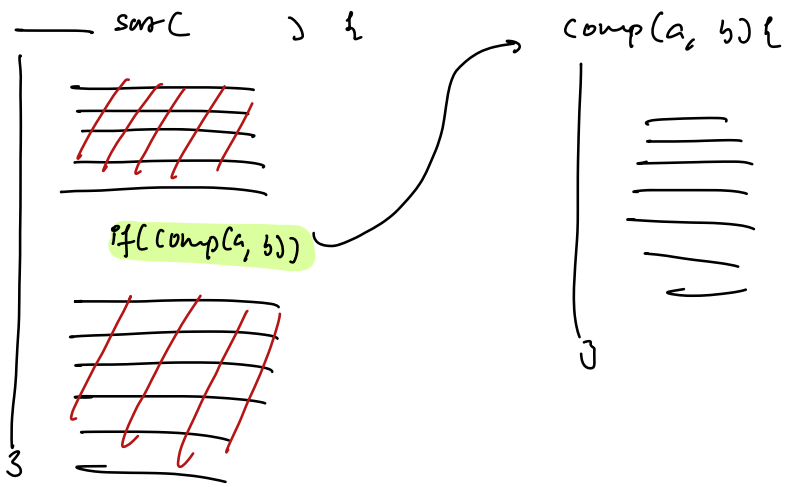
given ar[]

Sort(ar, comp)

your own order

TC → In Adv

```
int comp(int a, int b){ Java/JS/Python
    int fa = factors(a)   // factors in a
    int fb = factors(b)   // Implement factors on your own → TODO
    if(fa < fb){
        // a to come first
        return -1
    }                                   C++ return True
    else if(fa == fb && a < b){
        return -1 // a comes first
    }                                   C++ return True
    else { return 1 // b comes first, }  C++ return false
}
```


sort(
) {
comp(a, b) {
if(comp(a, b))
}
}