What is Sorting

Min cost to delete elements

Count of noble elements

Sorting Algo

Selection and Insertion sort

Sorting : Arrangement of data in a particular order based on some parameter

2 3 9 12 17 19	Sorted in asc order (based on value)
19 6 5 2 -1 -19	sorted in desc order (based on value)
1 13 9 6 12 count ↓ ↓ ↓ ↓ ↓ of 1 2 3 4 6 factors	Sorted in asc order (based on count of factors)

Default → asc order based on value

Custom sort

Why sorting ? Searching becomes easier
Analysis
Readability

Inbuilt sort function sort ( )

↓

TC : $O(n \log_2 n)$

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.

Ex [2 1 4]

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

[2 1 4]

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

Ans = 11

Q. [4 6 1]

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

6 4 1
↓ ↓ ↓
6 8 3

Q. [~~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

ans → 2

Order of deletion for min cost

↓

biggest → smallest

10 2

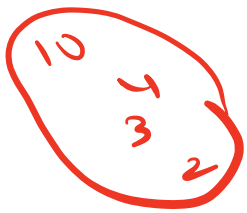
10 → 2

delete 10	10 + 2
delete 2	2

2 → 10

delete 2	2 + 10
delete 10	10

10 (↓1) 2 (↓2) < 2 (↓1) 10 (↓2)



X X X X

[a b c d]

Delete a a + b + c + d

Delete b b + c + d

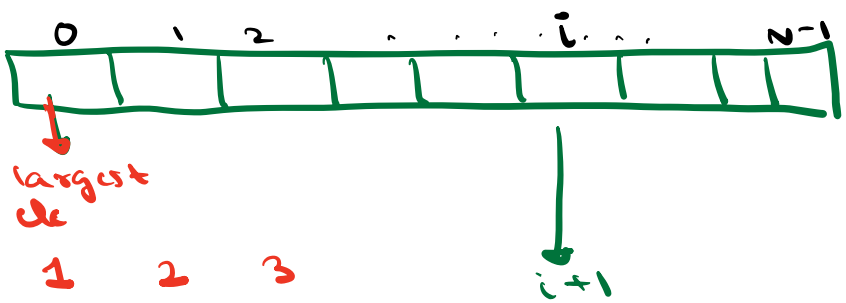
Delete c c + d

Delete d d

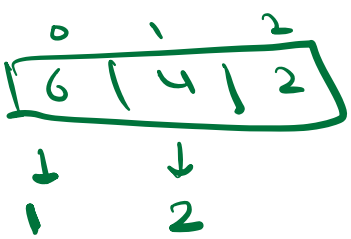
a + 2b + 3c + 4d → Min

a > b > c > d

sort (arr) → desc order



2 & 6 →




```
// arr[], n
reverse_sort(arr);  → O(nlogn)
int cost = 0
for (i=0; i<n; i++) {        → O(n)
    cost += (i+1) * arr[i]
}
return cost
```

TC: O(nlogn)

SC: O(sorting algo space)

O(1) → O(N)

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

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

N = 6

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

ans = 3

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

ans = 1

elem = count of smaller

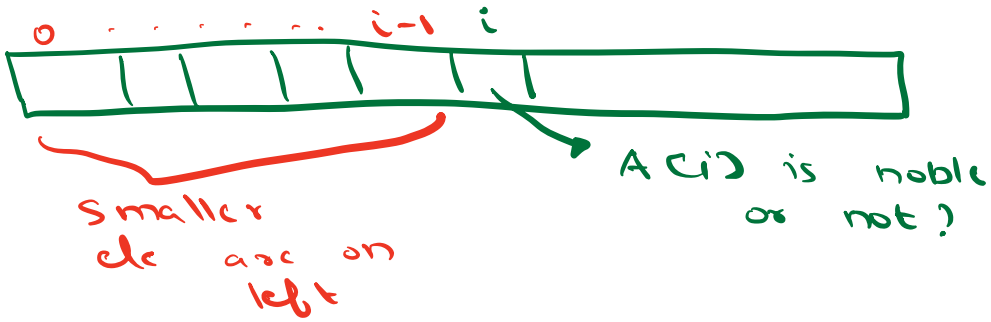
-ve no. = 0 or +ve

BF : For every element, check it is noble

```
int ans = 0
for (i = 0 ; i < n ; i++) {
    // A[i] is noble or not
    int cnt = 0
    for (j = 0 ; j < n ; j++) {
        if (arr[j] < arr[i])
            cnt++
    }
    if (A[i] == cnt)   ans++
}
```

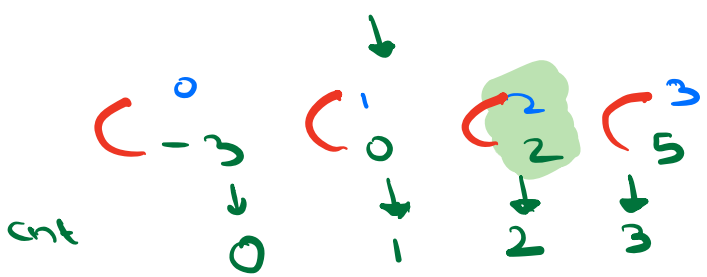
TC: $O(N^2)$
SC: $O(1)$

2. sort data (asc)



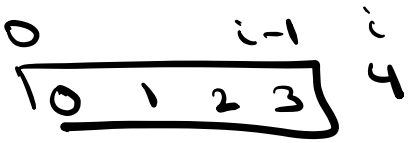
cnt = [0 i-1] = i-y-0+x = i

2 5 0 -3 → -3 0 2 5 (0 1 2 3)




```
sort(arr)                    → nlogn
int ans = 0
for (i=0 ; i<n ; i++) {      → n
    cnt = i
    if (A[i] == cnt)
        ans++
}
return ans
```

TC: O(nlogn)
SC: O(Sorting algo)



10:22

What if there are duplicates?

	0	1	2	3	4
Ex	-10	1	1	3	100
cnt	0	1	1	3	4

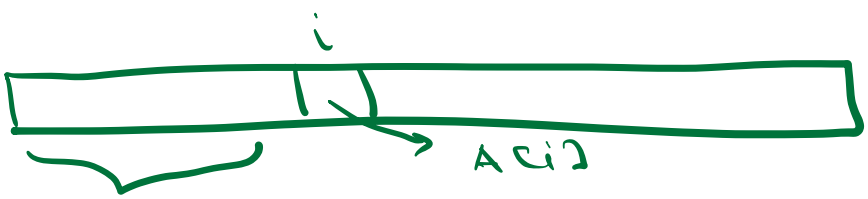
ans = 3

	0	1	2	3	4	5	6	7	8
Ex	-10	1	1	2	4	4	4	8	10
cnt	0	1	1	3	4	4	4	7	8

ans = 5

	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
cnt	0	1	2	2	4	4	4	4	8	8	10	10	10	13

ans = 7



cnt → i

Let's try to find those elements where i ≠ cnt

if cur_elem != prev_elem
count = i

TC: O(nlogn)

```
sort(arr)
int cnt = 0
if (arr[0] == 0)   ans++

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

	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

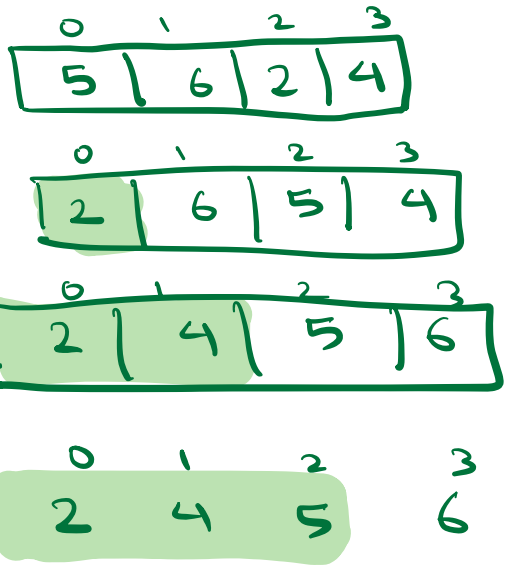
cnt: 0 → 1 → 2 → 2 → 4 → 4 → 4 → 4 → 8

ans: 0 ✗ 1 ✗ 2 ✗ 3

Selection sort

→ All students are in unarranged queue
→ Search for shortest student in the list of arranged students
→ Keep doing process till whole class is done.

idx	search
0	0 - 3
1	1 - 3
2	2 - 3
3	3 - 3

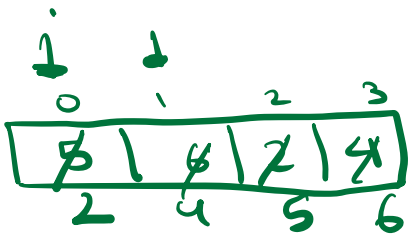


N-1 times

```
for (i=0 ; i < n-1 ; i++) {
    // search from i till n-1
    min_el = ∞ / INT_MAX
    min_idx = -1
    for (j=i ; j < N ; j++) {
        if (arr[j] < min_el) {
            min_el = arr[j]
            min_idx = j
        }
    }
    swap (arr[i], arr[min_idx])
}
```

TC: $O(N^2)$

SC: $O(1)$



i: 0 → j: 0 → 3

min_el = 5 → 2

min_idx = 1 → 2

i: 1 → j: 1 → 3

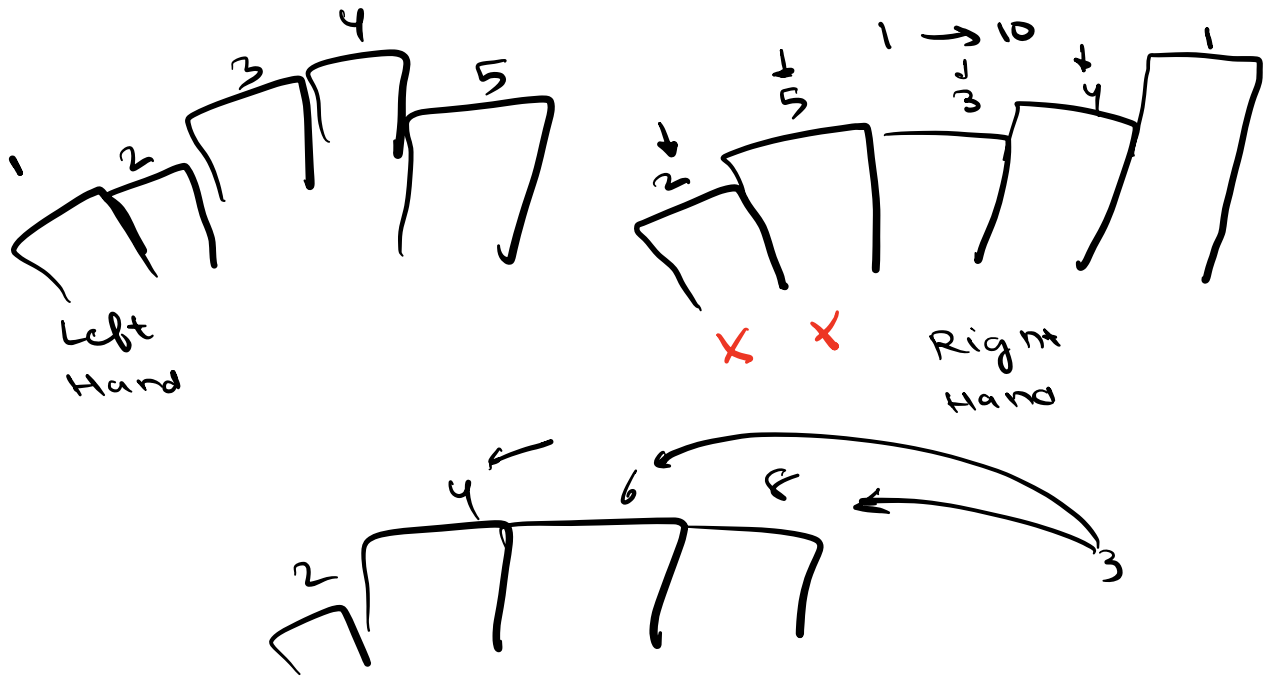
min_el = 4

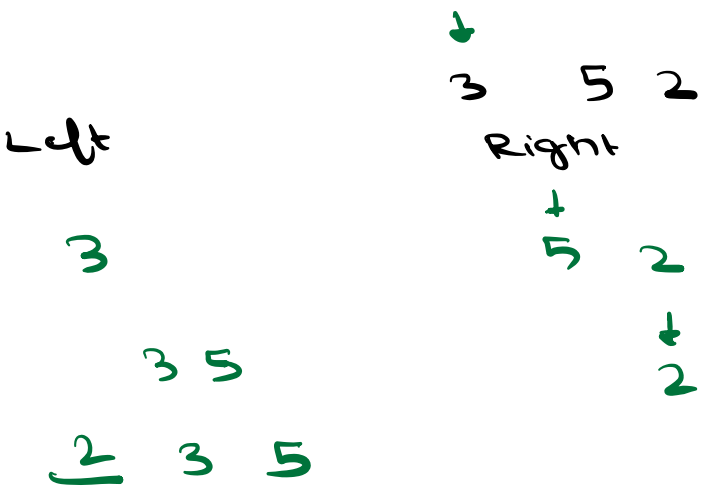
min_idx = 3

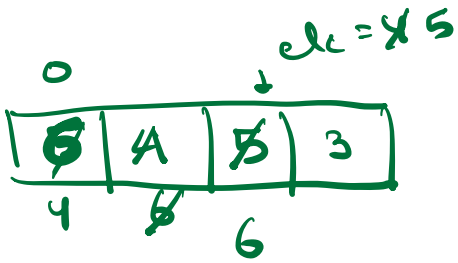
Inplace sorting algo → SC: $O(1)$

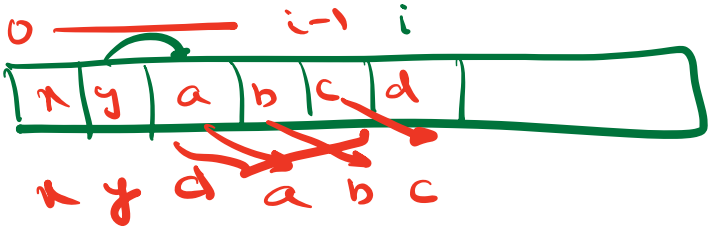
Is Selection sort inplace? ✓

Insertion sort





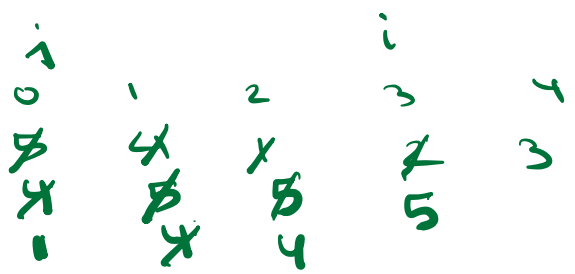


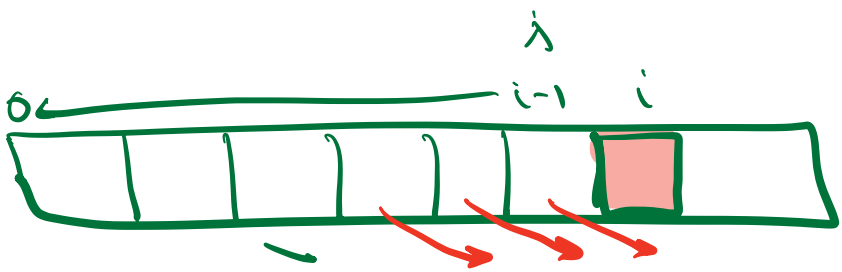


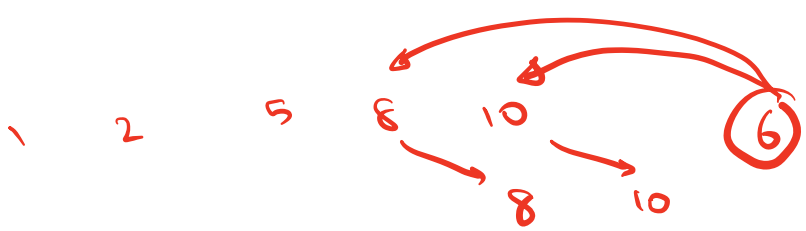

```
for (i = 1 ; i < n ; i++) {
    int curele = arr[i]
    j = i-1
    while (j >= 0 && arr[j] > curele) {
        arr[j+1] = arr[j]
        j--
    }
    arr[j+1] = curele
}
```

TC: O(N²)
SC: O(1)

curele = x, y, 2

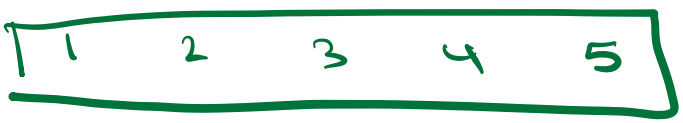




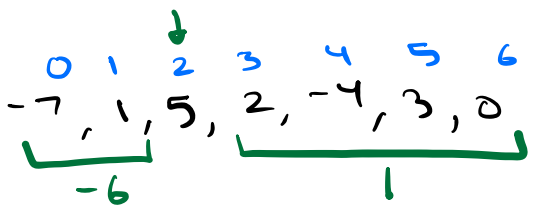


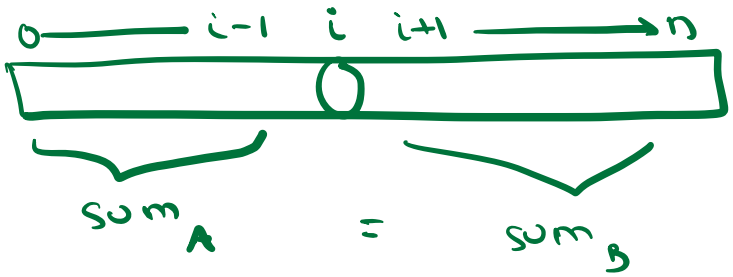
Sorted data

insertion > selection sort



Doubts





vector <vector <int>> v (row, vector <int> (col, 0));

2 x 4

	0	1	2	3
0	8	9	10	11
1	1	2	3	4

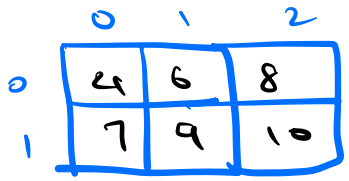
size of outer list → 2

size of inner list → 4

[[8, 9, 10, 11] , [1, 2, 3, 4]]

4 4

1 2



2 x 3

	0	1
0	4	7
1	6	9
2	8	10

3 x 2

mat[M][N] ↓ trans[N][M]

mat[2][3]

```
for (row = 0 ; row < M ; row++)
    for (col = 0 ; col < n ; col++)
        trans[col][row] = mat[row][col]
```