Sorting

TABLE OF CONTENTS

Sorting → Arrangement of objects in a particular order wrt specific parameter.

2 8 1 3 ⟶ 1 2 3 8

ascending order

1 13 9 6 12

factors → 1 2 3 4 6

Why →
1) organizing
2) analyzing
3) searching

Question (Elements Removal)

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

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

Find minimum cost to remove all elements.

NOTE : First add the cost of removal and then remove it.

0 1 2
arr - [2 1 4]

index to remove	cost	array
0	2 + 1 + 4 = 7	[1 4]
1	1 + 4 = 5	[4]
2	4 = 4	x

16 x

---

0 1 2
arr - [2 1 4]

index to remove	cost	array
2	2 + 1 + 4 = 7	[2 1]
0	2 + 1 = 3	[1]
1	1 = 1	x

11 ✓

0 1 2
A = [4 6 1]

index	cost	rem array
1	4 + 6 + 1 = 11	[4 1]
0	4 + 1 = 5	[1]
2	1 = 1	x

17

A = [3 5 1 -3]
0 1 2 3

index	cost	rem array
1	3+5+1-3 = 6	[3 1 -3]
0	3+1-3 = 1	[1 -3]
2	1-3 = -2	[-3]
3	-3	X
	2	

Observation → Remove large values.

[a b c d]
0 1 2 3

wt → 1 2 3 4 → (i+1)

Remove	cost
a	a + b + c + d
b	b + c + d
c	c + d
d	d
	a + 2b + 3c + 4d

minimise cost

a > b > c > d

// sort A[] in descending → TC = O(N log(N)) SC = O(N) / O(1)

```
cost = 0
for i → 0 to (N-1) {
    cost += A[i] * (i+1)
}
return cost
```

TC = O(N log(N) + N) = O(N log(N)) SC = O(N)/O(1)

Question (Noble Integers) { Distinct data }

Given N array elements, calculate number of noble integers.

An element ele in arr [ ] is said to be noble if { count of smaller elements = ele itself }

0 1 2 3 4 5
arr - [1 , -5 , 3 , 5 , -10 , 4]

ele < A[i] → 2 1 3 5 0 4 Ans = 3

0 1 2 3
arr - [-3 , 0 , 2 , 5]

elements < A[i] → 0 1 2 3 Ans = 1

Bruteforce → ∀ A[i], find the # elements < A[i] & check for noble integer.
O(N) O(N)

TC = O(N²) SC = O(1)

```
ans = 0
for i → 0 to (N-1) {
    cnt = 0
    for j → 0 to (N-1) {
        if ( A[j] < A[i] ) cnt++
    }
    if ( cnt == A[i] ) ans++
}
return ans
```

	0	1	2	3	4	5
A = [	3	8	2	-1	5	-3]
#e < A[i] →	3	5	2	1	4	0

Ans = 2

move all elements < A[i] on 1 side

⇒ counting may be faster

Arranging data s.t smaller values are on 1 side ⇒ Sorting

(left) → (ascending order)

	0	1	2	3	4	5	
Sorted → A = [	-3	-1	2	3	5	8]	
#e < A[i] →	0	1	2	3	4	5	= index

```
// sort A[] in ascending order
cnt = 0
for i → 0 to (N-1) {
    if ( A[i] == i ) cnt++
}
return cnt
```

TC = O(N log((N) + N) = O(N log(N))

SC = O(N) / O(1)

Question (Noble Integers) : { Data can repeat }

```
                0   1   2   3   4
arr - [ -10 , 1 , 1 , 3 , 100 ]
#e < A[i]  →    0   1   1   3   4        Ans = 3
```

```
                0   1   2   3   4   5   6   7   8
arr - [ -10 , 1 , 1 , 2 , 4 , 4 , 4 , 8 , 10 ]
#e < A[i] →     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
arr - [ -3 , 0 , 2 , 2 , 5 , 5 , 5 , 5 , 8 , 8 , 10 , 10 , 10 , 14 ]
#e < A[i] →     0   1   2   2   4   4   4   4   8   8   10  10  10  13        Ans = 7
```

```
// sort A[] in ascending order
ans = 0        cnt = 0
for i → 0 to (N-1) {
    if (i == 0 || A[i] != A[i-1])
        cnt = i
    if (cnt == A[i])    ans++
}
return ans
```

TC = O(N log (N) + N) = O(N log (N))

SC = O(N) / O(1)

Selection Sort

idea : Select the minimum element and send that elements to correct position by swapping.

	0	1	2	3	4	5
A = [	3	8	2	-1	5	-3]

Find max element in A[] → TC = O(N) SC = O(1)

Find second max element → TC = O(2N) = O(N)

SC = O(2) = O(1)

Find third max element → TC = O(3N) = O(N)

SC = O(3) = O(1)

⋮

Find K^{th} largest element → TC = O(K * N) ✓

SC = O(K) → O(1) ✓

	0	1	2	3	4	5
A = [	~~3~~	~~8~~	2	~~-1~~	~~5~~	~~-3~~]
	-1	-3		3	5	8

K elements is sorted position

∴ if K = N-1 ⇒ sorted array

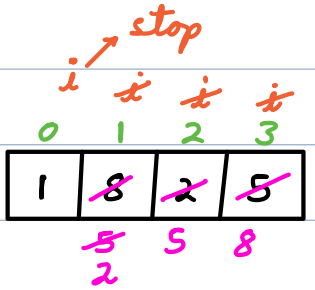
	0	1	2	3	4	5
A = [	~~3~~	~~8~~	~~2~~	~~-1~~	~~5~~	~~-3~~]
	~~-1~~	~~-3~~	2	3	5	8
	-3	-1				

Selection sort

< / > Code

```
for i → (N-1) to 1 {    // 0 — i
    m = 0    // index of max element
    for j → 1 to i {
        if (A[j] > A[m])  m = j
    }
    swap (A, m, i)
}
```

index



m = ~~0~~ 1

$TC = O(N^2)$

$SC = O(1)$

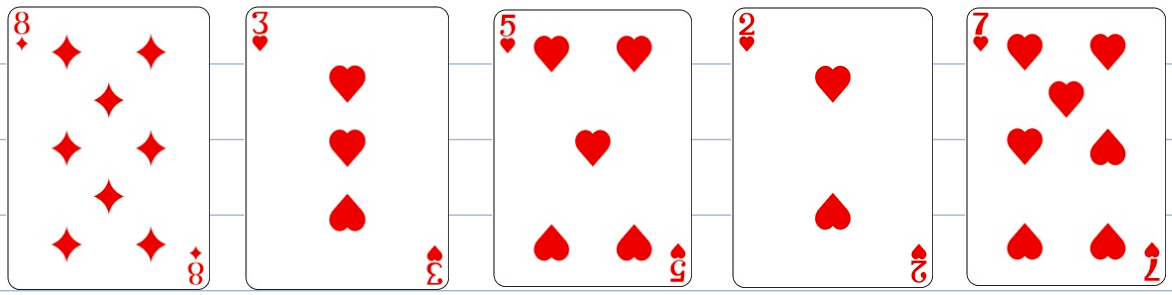
- Max → last
- Max → first
- Min → last
- Min → first

Descending

Ascending

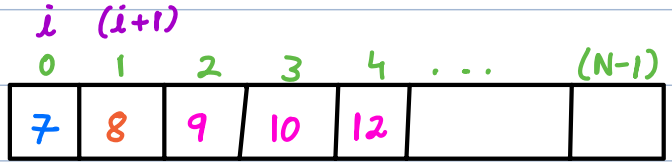
H.W → Study Bubble Sort.

Insertion Sort (Arrangement of playing cards)



Why used → It can sort running stream of data.

i/p → 7 9 12 10 8 (x)



n = 4

for any input → min swaps = 0

max swaps = # of elements in array

```
n = 0
for Vinputs, x {
    i = n-1                    // 0 ___ i (current array)
    while (i >= 0) {
        if (A[i] > x)
            A[i+1] = A[i]      // shift right
        else
            break              // i → index of small/equal
        i--
```

```
    }  A[i+1] = x
}  return A
```

TC = O(N²) SC = O(1)