Sorting Basics

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

ex1	2 4	7	11	15	: sorted in Asc	par = array values
ex2	15 9	6 6 2	0		: sorted in Desc	par = array values

ex3 1 13 9 6 12 : sort in Asc based on #factors = par

ex3	1	13	9	6	12
#factors	1	2	3	4	6
	(1)	(1,13)	(1,3,9)	(1,2,3,6)	(1,2,3,4,6,12)

Question 1

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

Cost to remove ele : sum of all elements present in array

find min cost to remove all elements.

Note : first calculate the cost, then remove the element.

eg a[3] = [2 1 4]

		cost	
[2 1 4]	remove 2 :	7	(2+1+4)
[1 4]	remove 1 :	5	(1+4)
[4]	remove 4 :	4	
		16	

remove 4 : 7
remove 2 : 3
remove 1 : 1
11

remove 1 : 7 (2+1+4)
remove 2 : 6
remove 4 : 4
17

eg a = [4 6 1]

remove 6 : 11
remove 4 : 5
remove 1 : 1
17

$a = [3\ 5\ 1\ -3]$

remove 5 : 6 (3+5+1-3)

remove 3 : 1 (3+1-3)

remove 1 : -2 (1-3)

remov -3 : -3

2

observation : deleting ele by ele in decreasing order to get min cost ?

$a[4] = \{a, b, c, d\}$

		cost
remove a	:	a+b+c+d
remove b	:	b+c+d
remov c	:	c+d
remove d	:	d

$\rightarrow$ total cost

$\overset{0}{a} + 2\overset{1}{b} + 3\overset{2}{c} + 4\overset{3}{d}$

$a \geq b \geq c \geq d$

$a + 4d$ (1,4)

$1 + 4\times4 = 17$

$4 + 4\times1 = 8$

Code

```
int calculateCost( a[], n) {
    sort-desc ( a)   → TODO in your own language
                        O(NlogN)
    ans = 0
    for( i=0; i<n; ++i) {
        ans += ( i+1) * a[i]
    }
    return ans
}
```

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

SC: O(1)

Question 2 Noble Integers {Data is distinct}

Given N elements, calculate no. of noble integers.

An element in a[] is called noble iff

$\underbrace{\text{no. of elements}}_{\text{count}} < \text{ele.} = \text{ele itself}$

eg	-1	-5	3	5	-10	4
#len	2	1	3	5	0	4

count = 3

	-3	0	2	5
#len	0	1	2	3

count = 1

```
def findNoble(a[], n) {
    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 : sort the array in asc. order

sorted(a) : $\underbrace{a[0] \quad a[1] \ldots \ldots}_{}$, a[i] . . . . . . a[n-1]

[0, i-1] all these ele are less than a[i]

count = i

↘ if (a[i] == i) then a[i] is noble

```
def findNoble(a[], n) {
    sort(a, asc) → O(NlogN)
    ans = 0
    for(i=0; i<n; ++i) {
        if (a[i] == i)
            ++ans
    }
    return ans
}
```

TC : O(NlogN)

SC : O(1)

dry run : -1 -5 3 5 -10 4

sort(a): -10 -5 -1 3 4 5

$$\begin{array}{cccccc} -10 & -5 & -1 & 3 & 4 & 5 \\ 0 & 1 & 2 & \underbrace{3 \quad 4 \quad 5}_{ans = 3} \end{array}$$

Question 3

Count Noble integers : { Data can repeat }

eg

	-10	1	1	3	100
# less	0	1	1	3	4

ans = 3

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

ans = 5

eg

	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

ans = 7

obs1 : If ele. coming for first time — if (a[i] != a[i-1])
count of ele. less than will be = i

obs2 : If ele. repeats, — if (a[i] == a[i-1])
count of ele less than will remain same

Code

```
def findNoble ( a[], n) {

    sort (a, asc)
    ans = 0
    count = 0 # ele less than a[i]
    if (a[0] == 0)  { ans = 1 }
    for (i=1;  i<n;  ++i) {

        if (a[i]  != a[i-1] ) {  # a[i] coming
                                       first time
            count = i
        }
        else {   # a[i] is repeating
            # len will not change
        }

        if (a[i] == count)
            ++ans

    }
    return ans,
}
```

→ else part not needed only written for understanding

TC: O (NlogN)

SC: O(1)

Sort → Selection Sort

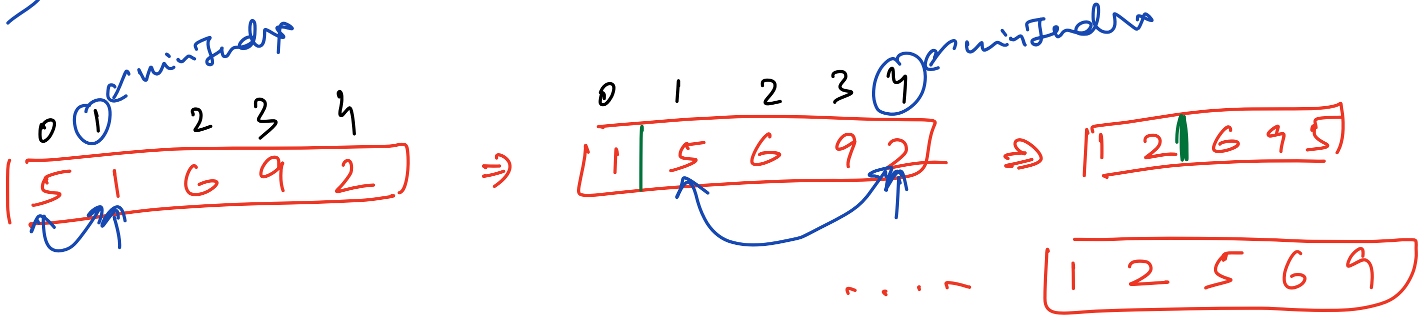
1. Pick smallest and place in front.
2. Pick second-smallest & ~~place~~ place in 2nd pos.

   ⋮ so on

```
void selectionSort( a[] , n ) {

    for ( i=0; i<n; ++i ) {
        minIndex = i;

        for ( j=i+1; j<n; ++j ) {
            if ( a[j] < a[minIndex] ) {
                minIndex = j
            }
        }

        swap( a[minIndex], a[i] )

    }
```

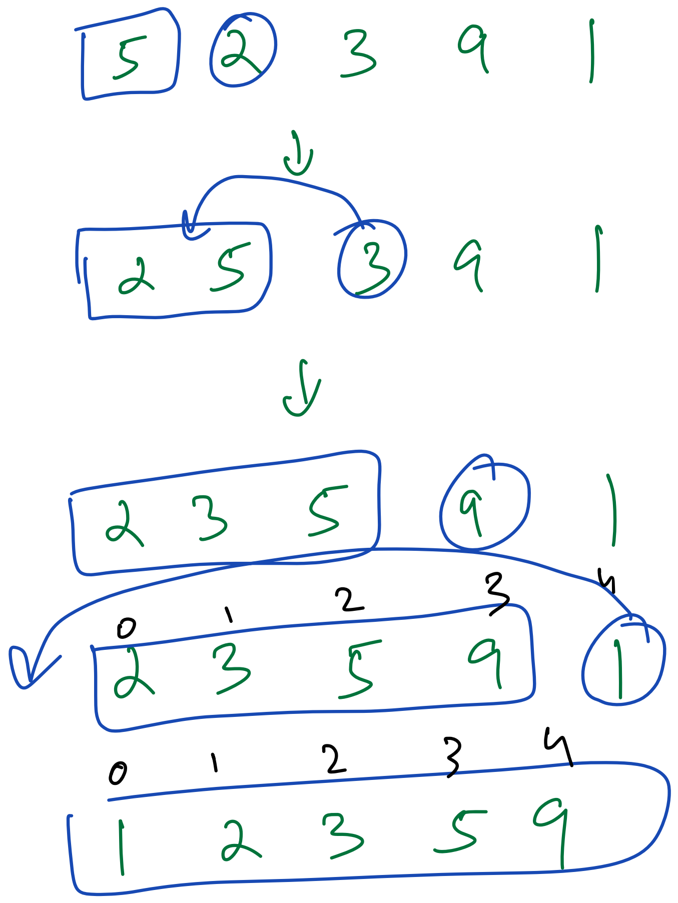
TC : $O(N^2)$

SC : $O(1)$



Sort — Insertion Sort

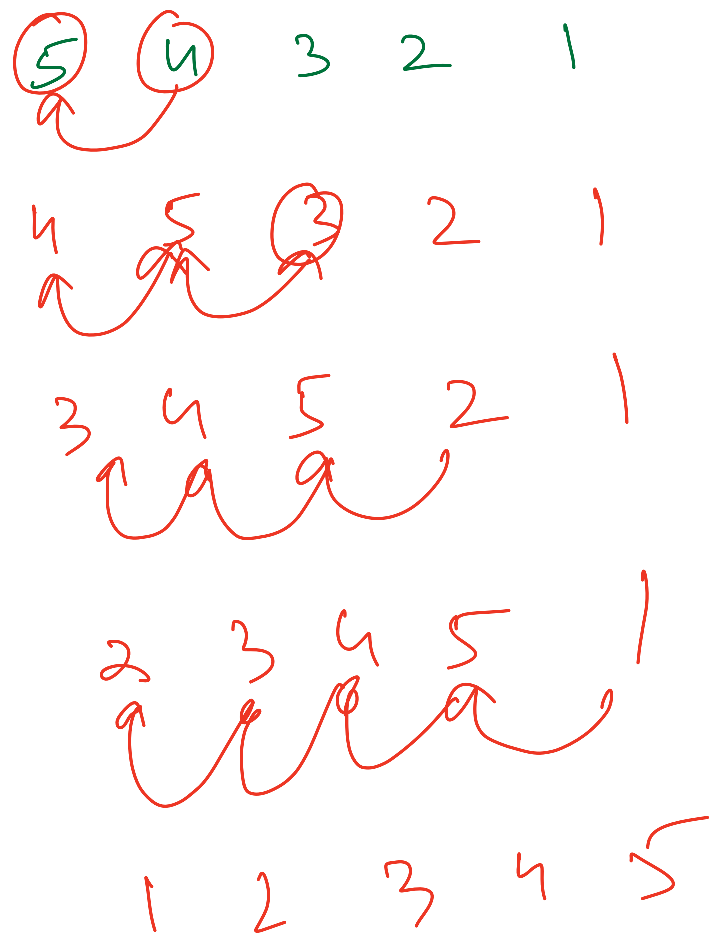
eg



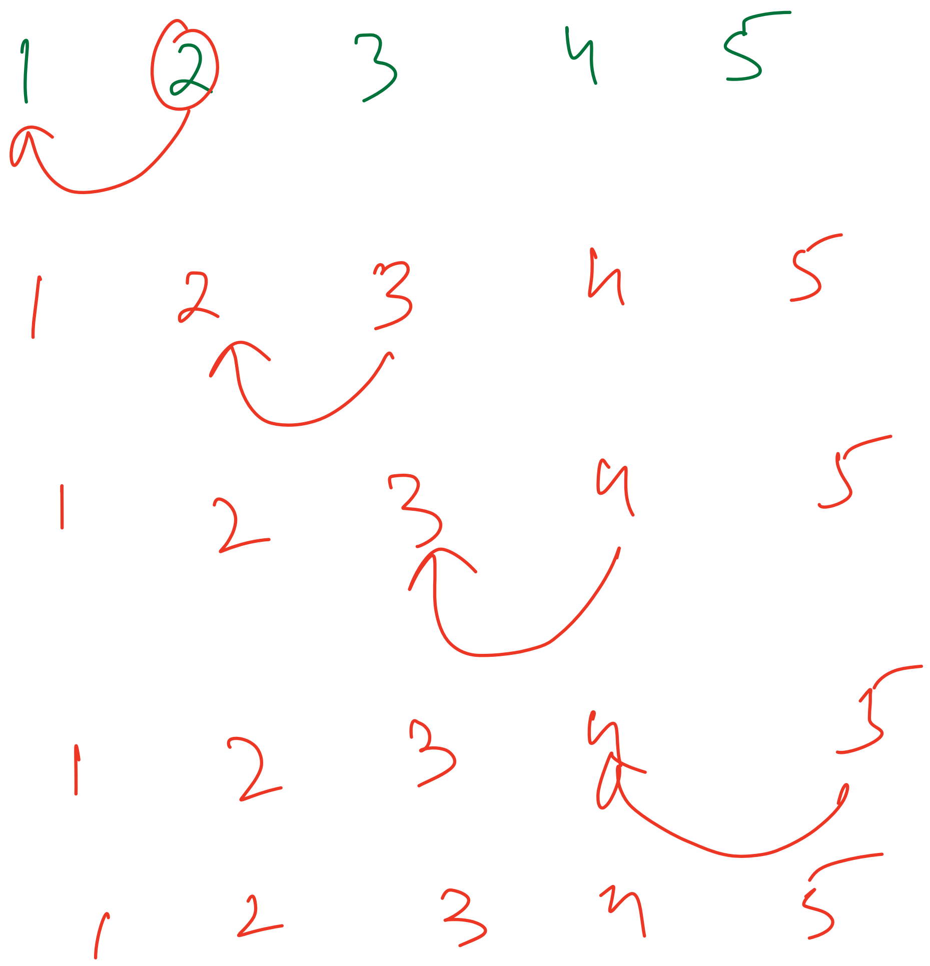
```
void insertionSort( a[] , n ) {

    for ( i=1; i<n; ++i ) {

        j=i;
        while ( j>0 && a[j] < a[j-1] ) {
            swap( a[j], a[j-1] )
            j--
        }
    }
}
```

TC:

SC : O(1)


5 4 3 2 1
4 5 3 2 1
3 4 5 2 1
2 3 4 5 1
1 2 3 4 5

$O(N^2)$


1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

TC: $O(N)$