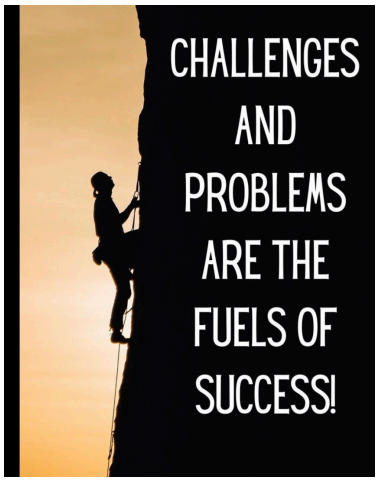
Today's Quote



Today's content

- understand sorting
- few problems on sorting
- 1 sorting algo
  - Comparator

Sorting → Arranging data into inc/dec order based on parameter.

arr → [2, 4, 7, 11, 25] sorted in inc. order, par = value

arr → [70, 50, 11, 6, -2] sorted in dec. order, par = value

arr → [1 2 3 7 4 9 6] : sorted, par = factors.

#factors 1 2 2 2 3 3 4

Inbuilt library functions →

- sort() , in every language.
- How? logic? ⇒ { In advance module }
- T.C → O(NlogN) , N = no. of elements.

Q) 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: first add the cost of removal & then remove it.

Eg: arr[3] = {2 1 4}
(0 1 2)

	Cost of Removal
remove 2:	7
remove 1:	5
remove 4:	4
Total cost =	16

arr[3] = {2 1 4}
(0 1 2)

	Cost of Removal
remove 4:	7
remove 2:	3
remove 1:	1
Total cost =	11

Q. arr = {3, 6, 2, 4}

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

arr = [6, 4, 3, 2]
(0 1 2 3)
* * * *
1 2 3 4

(6*1) + (4*2) + (3*3) + (2*4)

= 31

Observation Start deleting the elements in decreasing order. in order to get the min-cost. ?

arr[4] : { a b c d }

cost.

remove a : a+b+c+d

remove b : b+c+d

remove c : c+d

remove d : d

a + 2b + 3·c + 4·d

- a → largest element
- b → 2nd largest
- c → 3rd largest
- d → 4th largest

pseudo-code

```
int minCost ( arr, N ) {
    Sort( arr, desc)   // sort the arr in desc. order
                       // Syntax - check out in your own language.

    cost = 0
    for ( i = 0 ; i < N ; i++) {
        cost += (arr[i] * (i+1));
    }

    return cost
}
```

T.C → O(NlogN)
S.C → O(1)

Q) Noble Integers {distinct data}

Given N array element, calculate no. of noble integers.

An element ele in arr[] is said to be noble if

{count of elements less than ele = ele itself}

Eg → { 1 (0), -5 (1), 3 (2), 5 (3), -10 (4), 4 (5) } → ans = 3.

	1 (0)	-5 (1)	3 (2)	5 (3)	-10 (4)	4 (5)
#smaller	2	1	3	5	0	4

observation

-ve integers can't be noble integers.

Eg → { -3 (0), 0 (1), 2 (2), 5 (3) } → ans = 1.

	-3 (0)	0 (1)	2 (2)	5 (3)
#smaller	0	1	2	3

Eg → { -10 (0), -5 (1), 1 (2), 3 (3), 4 (4), 5 (5), 10 (6) } → ans = 3.

	-10 (0)	-5 (1)	1 (2)	3 (3)	4 (4)	5 (5)	10 (6)
#smaller	0	1	2	3	4	5	6

Brute Force Solution: for every element, iterate & find count of smaller elements.

```
int nobleIntegers(arr, N){
    count = 0
    for(i=0; i<N; i++){
        less = 0
        for(j=0; j<N; j++){
            if(arr[j] < arr[i]){ less++ }
        }
        if(arr[i] == less){ count++ }
    }
    return count
}
```

T.C → $O(N^2)$

S.C → $O(1)$

Optimisation

Sort the array in ascending / increasing order.

After sorting, count of smaller elements for arr[i] is i.

```
int nobleIntegers ( arr, N ) {
    sort( arr, asc)
    int count = 0
    for ( i = 0; i < N ; i++){
        if (arr[i] == i ) {
            count++
        }
    }
    return count
}
```

T.C → O(NlogN)
S.C → O(1)

arr →

-3	0,	2,	5.
0	1	2	3

Q: Noble integers : {data can repeat}

Eg: { 0 2 2 3 3 6 }
(indices: 0 1 2 3 4 5)
#smaller: 0 1 1 3 3 5

Ans = 3.

Eg: { -10 1 1 1 4 4 4 7 10 }
(indices: 0 1 2 3 4 5 6 7 8)
#smaller: 0 1 1 1 4 4 4 7 8

ans = 7.

Q: { -3 0 2 2 5 5 5 5 8 8 10 10 10 14 }
(indices: 0 1 2 3 4 5 6 7 8 9 10 11 12 13)
#smaller: 0 1 2 2 4 4 4 4 8 8 10 10 10 13

ans = 7

Observations :

① If the element is repeating, if (arr[i] == arr(i-1))
count of smaller elements is going to remain same.

② If the element is coming for the first time, if(arr[i] != arr[i-1])
count of smaller elements = i

pseudo-code

```
int noble Integers ( arr , N ) {
    sort(arr, asc.)
    smaller = 0 , ans = 0
    if ( arr[0] == 0) { ans ++ }

    for( i = 1 ; i < N ; i++) {
        if ( arr[i] != arr[i-1]){
            smaller = i
        }
        else {
            //arr[i] is repeating.
            // do nothing
        }
        if (arr[i] == smaller) { ans ++ }
    }
    return ans ;
}
```

arr → | 1 | 2 | 2 | 3 | 3 | 6 |

0 1 2 3 4 5

1 1 3 3 5

smaller = ~~0~~ ~~1~~ ~~2~~ 3

ans = ~~0~~ ~~1~~ 2

T.C → O(NlogN)
S.C → O(1)

{ Break → 8:40 AM → 8:50 AM }.

Sorting algo [Bubble sort]

arr → { 3 6 2 8 4 } (8 at 4, 6 at 2, 2 at 1, 4 at 4)

① { 3 2 4 6 8 } : 8 at its correct position.

② { 2 3 4 6 8 } : 6 & 8 at their correct positions

③ { 2 3 4 6 8 } : 4, 6, 8 at their " positions.

observation

After 1 iteration → last 1 element at correct position

After 2nd iteration → last 2 elements at " "

After 3rd iteration → " 3 " " " "

After N-1th iteration → last N-1 elements at correct position.

All elements will be present at their correct positions.

idea & pseudo-code {keep doing this N-1 times}

→ compare 2 consecutive elements, if they are not present in inc. order swap them.

```
bubbleSort(arr, N) {
    for(i = 1 ; i < N ; i++){
        for(j = 0 ; j < N-1 ; j++){        // we can slightly optimize this.
            if(arr[j] > arr[j+1]){
                swap(arr[j] & arr[j+1])
            }
        }
    }
}
```

T.C → $O(N^2)$

S.C → $O(1)$

// Decreasing

```
bubbleSort( arr, N) {
    for( i = 1 ; i < N ; i++){
        for( j = 0 ; j < N-1 ; j++){
            if( arr[j] < arr[j+1] ){
                swap(arr[j] & arr[j+1])
            }
        }
    }
}
```

has power to decide the order of the data.

obs.→ if we simply change this condition, it allows us to get desired order.

idea.→ write a function for this condition.

```
bubbleSort( arr, N) {
    for( i = 1 ; i < N ; i++){
        for( j = 0 ; j < N-1 ; j++){
            if( comp(arr[j] , arr[j+1]) ){
                swap(arr[j] & arr[j+1])
            }
        }
    }
}
```

```
boolean comp ( a , b ){
    // Just implement this & you will get your desired order.
    // if a should come before b or vice versa.
}
```

[a-b < 0 → a should come before b.]

Comparator in Java / JS / C#

```
int Comparator customComparator = ( Integer a, Integer b) {
    if you want, a should appear before b : { return -1 }
    if you want, a & b are same           : { return 0 }
    if you want, b should appear before a : { return 1 }
};
Arrays.sort(arr, comp)
```

Python

```
def compare (a, b):
    if you want, a should appear before b : { return -1 }
    if you want, a & b are same           : { return 0 }
    if you want, b should appear before a : { return 1 }

arr.sort ( key = cmp_to_key (compare))
```

in C++.

```
boolean comp(a, b) {
    if a should appear before : return true
    else                      return false
};
sort (ar, comp)
```

Q1 Given N elements. Sort them in increasing order of their no. of factors.

Note: If two elements have same no. of factors, element with less value should come first.

→ No extra space is allowed.

Eg→ arr: { 9 3 4 8 16 37 6 13 15 }
0 1 2 3 4 5 6 7 8

factors: { 3 2 3 4 5 2 4 2 4 }

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

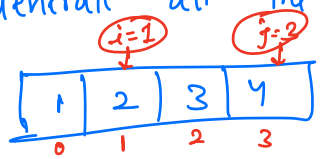
idea.

sort(arr, comp);

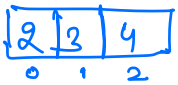
{T.C → advanced}.

```
int comp ( int a , int b) {
    int fa = factors(a)        // todo.
    int fb = factors(b)
    if ( fa < fb ){
        //a should come first
        return -1
    }
    else if ( fa == fb && a < b) {
        // a should come first
        return -1
    }
    else {   // b should come first
        return 1
    }
}
```

2-D Array – Generate all the subarrays.

arr – 

≅

a → 


```
rows = n(n+1)/2
int[][] arr = new int[rows][];
int t = 0;
for(i=0; i<N; i++){
    for(j=i; j<N; j++){
        int[] a = new int[j-i+1];
        l = 0;
        for(k=i; k<=j; k++,l++){
            a[l] = arr[k];
        }
        arr[t] = a;
        t++;
    }
}
```

```
[ [1],
  [1,2],
  [1,2,3],
  [1,2,3,4],
  [2],
  [2,3],
  [2,3,4],
  [3],
  [3,4],
  [4] ]
```

req. ans.

Store subarray from i, j
⇓
length of subarray from i to j ⇒ j-i+1

Try this out!