Lecture :- Sorting

Agenda

- Basics ✓
- Problems
- Bubble Sort
- comparator

Sorting Arranging data in inc | dec order. on basis of any parameter

Ex: arr[] = [4, 3, 1, 5, 2]

↓ sort | arrange in inc^r order

arr[] = [1, 2, 3, 4, 5] — sorted

Ex2: arr[] = [1, 2, 3, 5, 4] — Not a sorted array

Ex3:

arr[] =	[1	2	3	7	4	9	6]
factors:-	1	2	2	2	3	3	4

— not sorted acc to values of array.

— sorted on basis of factors count

Qu How to sort the array then?

```
int[] arr = new int[n];

Arrays.sort(arr);  // arr has been sorted in
                      incr manner

arr[] = [2, 1, 5, 3, 4]

Arrays.sort(arr);

print(arr elements);   // [1, 2, 3, 4, 5]

Arrays.sort(arr, Collections.reverseOrder());  decr
```

TC: O(nlogn) — Advanced Module [How??]

Qu1 Given arr[n]. At every step remove an array element. Cost of removing an element = sum of array el present in array.

Find min cost to remove all el.

Note:

arr[] = [2 1 4]

case1:

	cost	
remove 2	2 + 1 + 4 = 7	arr[] = [1, 4]
remove 1	1 + 4 = 5	arr[] = [4]
remove 4	4	arr[] = { }
	Total cost = 16	

case2: arr[] = [2 1 4]

	cost	
remove 4	4 + 2 + 1 = 7	[2, 1]
remove 2	2 + 1 = 3	[1]
remove 1	1	[ ]
	Cost = 11 — Ans	

Observation

arr[] = [3 6 2 4]

remove	cost	arr[] updated
3	3 + 6 + 2 + 4	[6 2 4]
6	6 + 2 + 4	[2 4]
2	2 + 4	[4]
4	4	[ ]
	3 * 1 + 6 * 2 + 2 * 3 + 4 * 4	

Total sum –

1.) last el of arr contributes the max. [min el of array should be at last]

2.) first el of arr contributes the min [max el of array should be at first]

Approach:

Sort the array in dec order.

To minimise our cost –

max contribution — least el of array

min contribution — highest el of array

arr[] = [3 6 2 4]

↓ sort (dec

$$arr[] = [\overset{0}{6} \quad \overset{1}{4} \quad \overset{2}{3} \quad \overset{3}{2}]$$

	cost	arr
remove 6	6 + 4 + 3 + 2	[4 , 3 , 2]
remove 4	4 + 3 + 2	[3 , 2]
remove 3	3 + 2	[2]
remove 2	2	[ ]

$$\underset{arr[0]}{6} * \overset{(0+1)}{1} + \underset{arr[1]}{4} * \overset{(1+1)}{2} + \underset{arr[2]}{3} * \overset{(2+1)}{3} + \underset{arr[3]}{2} * \overset{(3+1)}{4}$$

```
            int minCost( int[] arr) {
                int n = arr.Length;
O(nlogn) ←      Arrays. sort( arr, Collections. reverseOrder());
                int sum = 0;
O(n)     ←      for( i=0; i<n; i++) {
                    sum = sum + [arr[i] * (i+1)]
                }
                return sum;
            }
```

TC: O(nlogn) + O(n) ≃ O(nlogn)

SC: O(1).

Qu2 Noble Integer(All data is distinct).

Given arr[n] , calculate no. of noble integers.

Noble int :- No of elements in arr < lesser than el = el itself.

	0	1	2	3	4	5	
arr[] =	[-1	-5	3	5	-10	4]	ans = 3.
# less	[2	1	3	5	0	4]	

	0	1	2	3	4	5	
arr[] =	[5	8	9	14	16	23]	→ sorted.

idx	el	# of el less than el itself
0	5	0
1	8	1 [5]
2	9	2 [5, 8]
3	14	3 [5, 8, 9]
4	16	4 [5, 8, 9, 14]
5	23	5 [5, 8, 9, 14, 16]

for a sorted array -

ith idx → No of lesser el = i

Example:

arr[] = [-1 -5 3 5 -10 4]

↓ sort(arr)

	0	1	2	3	4	5
arr[] = [	-10	-5	-1	3	4	5]
#lesser = [	0	1	2	3	4	5]

```
int    countNoble( int[] arr) {

nlogn. ←  Arrays.sort(arr);

          int  cnt = 0;

          for( i=0; i< arr.length; i++) {

                if( arr[i]  == i )  {
                                 └→ no of lesser el than arr[i]

                      cnt ++;
                }
          }

       return cnt;
}
```

TC: O(n log n)

SC: O(1)

follow up: Data can repeat

1.)
$$
\begin{array}{lcccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
arr[\,] = [& 0 & 2 & 2 & 3 & 3 & 6\,] \\
\#less = [& 0 & 1 & 2 & 3 & 4 & 5\,]
\end{array}
$$

→ ans = ~~2~~ 1

2.)
$$
\begin{array}{lcccccccccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
arr[\,] = [& -3 & 0 & 2 & 2 & 5 & 5 & 5 & 5 & 8 & 8 & 10 & 10 & 10 & 14\,] \\
\#less = [& 0 & 1 & 2 & 2 & \underline{\underline{4}} & 4 & 4 & \underline{\underline{4}} & 8 & 8 & \underline{\underline{10}} & 10 & \underline{\underline{10}} & 13\,] \\
 & & & [-3, 0] & & & & & & & & & & &
\end{array}
$$

→ If el is coming for the first time
  └ count of lesser el = i

} above algorithm

→ If el is repeated —
  └ count of lesser el = count lesser el for its first occurence

Dry run

	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]

idx.	first |repeat	# of el less than el itself
0 [-3]	first	0
1 [0]	first	1
2 [2]	first	2
3[2]	Repeat	count of first occ of 2 = 2.
4 [5]	first	4
5 [5]	Repeat	count of first occ of 5 = 4.
6 [5]	Repeat	Count of first occ of 5 = 4
7 [5]	Repeat	"
8 [8]	first	8
9 [8]	Repeat	count of first occ of 8 = 8.
⋮	⋮	⋮

Pseudo- code

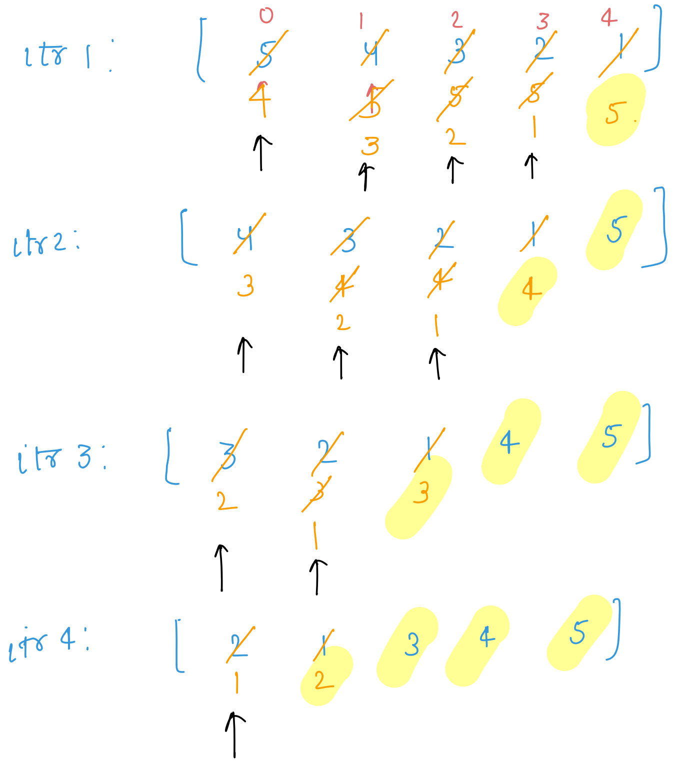
```
int nobleIntegersfinal ( int[] arr) {
    int n = arr.length;
    Arrays.sort(arr);

    int cnt = 0;
    int firstoccCount = 0;
    // handle 0th idx alone
    // 0th idx will only be noble. if
       and only if arr[0] == 0.
    if ( arr[0] == 0) {
        cnt ++;
    }

    for( i = 1; i < arr.length; i++) {
        if ( arr[i] != arr[i-1]) { // first time
            firstoccCount = i
        } else {            // Repeated el.
            // Do nothing
        }
        if( arr[i] == first occcount; {
            cnt++;
        }
    }
    return cnt;
}
```

TC: O(n logn)
SC: O(1)

Bubble sort — Brute force sorting algo.

Idea: arr[] = [5 4 3 2 1] — inc^r order.

0	1	2	3	4
5	4	3	2	1



Observation: arr[5] → 4 itr, array was sorted

n = 5 → n-1 itr needed

arr[5] ⟹

i ⇒ itr		j	
1	——	[0 - 3]	[0 to n-i-1] 0 to 5-1-1 = 0 to 3
2	——	[0 - 2]	[0 to n-i-1] i=2
3	——	[0 - 1]	[0 to n-i-1] i=2
4	——	[0-0]	

1 to n-1

Sorting arr in inc^r order on basis of values.

```
void bubblesort(int[] arr) {
    for(i=1; i<=n-1; i++) {
        for(j=0; j<=n-1-i; j++) {
            if(arr[j] > arr[j+1]) {
                swap(arr, j, j+1);
            }
        }
    }
}
```

Responsible for swapping ⟵ if(arr[j] > arr[j+1])

swap(arr, j, j+1); — h|w O(1)

TC: $O(n^2)$

SC: $O(1)$

Applications:

arr[j] > arr[j+1]

arr[j] − arr[j+1] > 0

arr[j]   arr[j+1]

↑

if arr[j] > arr[j+1]

ans greater than 0

```
void bubblesort(int[] arr) {
    for(i=1; i<=n-1; i++) {
        for(j=0; j<=n-1-i; j++) {
            if(arr[j] - arr[j+1] > 0) {
                swap(arr, j, j+1);
            }
        }
    }
}
```

Responsible for swapping ⟵ if(arr[j] - arr[j+1] > 0)

swap(arr, j, j+1); — h|w O(1)

TC: $O(n^2)$

SC: $O(1)$

comparing acc to values of array

```
int compare( int a, int b) {
        if( a > b) {
            return 1; // 1 is greater than 0.
        }
        else if( a < b) {
            return -1;
        } else {
            return 0;
        }
}
```

```
void bubblesort(int[] arr) {
    for( i =1 ; i<=n-1; i++) {
        for( j = 0; j<= n-1-i; j++) {
            if( compare( arr[j], arr[j+1]) > 0)
                swap( arr, j, j+1);   — n|w  O(1)
            }
        }
    }
}
```

Responsible for swapping ← if(compare(arr[j], arr[j+1]) > 0)

TC: $O(n^2)$

SC: $O(1)$

Qu sort on basis of factors.

Refer the code attached

Thankyou