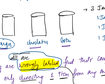
Sorting 1

→ From today daily we will start with a small q/a

1a) → 3 Jars
→ Jar - oranges (only)
→ Jar - chocolates (only)
→ Jar - both



1b) They are currently labelled by only labelling 1 jar

→ their current labelling is not correct

→ Now any one jar has wrong labelling

→ Based on this fact

→ Extract extra information

Both & Jars — oranges — Chocolates — Cases

Chocolates — both & — oranges — Cases

2a) Sorting:

Arranging data in an orderly manner (Asc / Des)

based on a parameter.

Data: 7 1 11 14 19 → 1 7 11 14 19 : increasing order

8 0 -1 -1 -3 → 8 1 5 : decreasing order

2 3 1 6 10 20 → 1 2 3 6 10 20 : 1) 1 is the

for: 2 2 3 4 4 6 : increasing order based on factors

→ Lazy sorting:

→ Lazy operation

→ To avoid extra

2b) Given N array element at every step delete an array element from array. Cost of the operation = (sum of all current array elements)

Repeat this process until array is empty. Find the min cost

Ex: [2 1 4]

1 → Cost = 7 → [2 1]

1 → Cost = 3 → [2 1]

2 → 60 = 2 → [2]

TC: 3 [ ]

Sorting TC) + N

Solutions:

Idea: $a_1 \; a_2 \; a_3 \; a_4$

$a_4 = a_1 + a_2 + a_3 + a_4$

$a_3 = a_1 + a_2 + a_3$

$a_2 = a_1 + a_2$

$a_1 = a_1$

Total cost

$4a_1 + 3a_2 + 2a_3 + 1a_4$

1st min 2nd min 3rd min 4th min

Idea:

Sort array in asc and delete one by one from end

→ It is not needed since sorting only need cost

Sort array in Des and delete one by one from start

2c)

Name	DOI (years)
Rohith	23
Sakshi	49
Vivek	25
Dhanu	18
Varun	10
Dhanu	18
Shivaj	20

Dhanu v Dhanu

Sorting data based on DOI (ascending)

Name	DOI (years)
Varun	10
Dhanu	18
Dhanu	18
Shivaj	20
Rohith	23
Vivek	25
Sakshi	49

Note:

When we sort based on a particular parameter, if for 2 data points value is same, relative order is preserved. (Stable Sorting Algorithm)

Sorting Algorithms:

- Selection
- Bubble
- Insertion
- Merge
- Quick
- Counting sort
- Radix Sort
- Bucket Sort
- Heap Sort
- Topological Sort

Revisited

2d) Given N array element find k smallest elements [Elements are K-1 after sorting]

	0	1	2	3	4	5	6	7
arr[ ]:	6	30	14	2	11	19	8	5

$k > 3$

Note: $k \ll N$ | $k \ll \log N$

$N \log N$ vs $(k.N)$ is better

→ Solution:

Sort & using arr[k-1]

TC: $O(\log N)$

	0	1	2	3	4	5	6	7
arr[ ]:	2	5	6	14	11	19	8	30

Iterate on array find min index → arr[k] ⇔ arr[k]

$i = N-1$

Value (index = i) arr[i] ⇔ arr[k]

$i = N-1$] interval = 5, index = 7

arr[i] ⇔ arr[7]

$i = 6$] interval = 4, index = 6

arr[i] ⇔ arr[6]

If we do this for k times to get our ans ⇒ $(k)(N)$ TC: $(k.N)$ SC: $O(1)$

To make entire array sorted, repeat the above for next elements

N-1 element → N-2 elements

TC: $O(N^2)$] Selection Sort

TC: N (iterate & swap with value index)

$\Rightarrow O(N^2)$

Pseudo Code:

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

→ If there array is already sorted ⇒ Time Complexity: $\frac{(N-1)(N-1)}{2}$

Nos of Iterations:

0 - N-1 → N

1 - N-1 → N-1

2 - N-1 → N-2

$\frac{(N)(N+1)}{2}$

→ Boundary Condition → Total Iterations $\frac{(N)(N+1)}{2}$

→ Angular Sorting algorithm

→ Time complexity

→ Stability: Ex: 2 2 1 [N]

After sorting → 1 2 2

Actual or after sorting → 1 2 2

TODO

How to make it stable

2e) arr[ ]: 3 2 1 8 4 7 -1 6] Note: Swap only Adjacent elements

	0	1	2	3	4	5	6	7
arr[ ]:	3	2	1	8	4	7	-1	6

1 Iteration → [2 3 1 8 4 7 -1 6] → [0-N-1]

2 Iteration → [2 3 4 7 -1 6 8] → 7 & 8

3 Iteration

Repeat this process for N-1 times

Actual data sorted

```
for(int i=0; i<N-1; i++){
  for(int j=0; j<N-i-1; j++){
    if(arr[j] > arr[j+1]){
      swap(arr[j], arr[j+1]);
    }
  }
  if(c==0) break
}
```

→ Avoid optimal path

$i < N-1$ (c)?

To avoid throwing the actual part

→ Bubble Sort: TC: $O(N^2)$

SC: $O(1)$

→ (Replace Sorting Algorithm)

Stable: YES

Ex: 2 1 2

1 2 2

Iterations:

Worst Case: $(N)(N-1)/2$

Best Case: $(N)(N-1)/2$

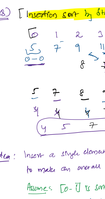
Using Flag condition ⇒ $O(N)$

→ 1 2 2 5 4

Iteration 1: 1 2 2 4 5 : swap = 1

Iteration 2: 1 2 2 4 5 : swap = 0 break

3a) [Insertion Sort by Hand by Concept]



Idea: Insert a single element to entire sorted array to make the array sorted

Assume: [0-i] is sorted

Element we want to move [i+1]

After insertion [0-(i+1)]

```
for(int i=0; i<N-1; i++){
  //Assume 0-i is sorted
  int ele = arr[i+1];
  for(int j=i; j>=0; j--){
    if(arr[j] > arr[j+1]){
      swap(arr[j], arr[j+1])
    }
    else{
      break;
    }
  }
}
//
```

// Inserting an element to a sorted array

Replace

Shuffle

Throwing

Worst: $(N)(N-1)/2$

Best: N

Insertion Steps:

Inserting a single element to sorted array

3b) [Merge Sort]

A[ ] = [2 4 5 6 10 11]

B[ ] = [3 7 8 9 11 14 18 20 22 25]

C[ ] = [2 3 4 5 6 7 8 9 10 11 14 18 20 22 25]

Solution:

1) Copy A → C + Copy B → C → $O(M+N)$

2) Copy A → C + Insert B → C → $M+N$

3) Copy B → C + Insert A → C → $N+M$

4) Iterate on Array A

Iterate on Array B

TC: $O(M+N)$ SC: $O(M+N)$

```
void merge(int A[], int M, int B[], int N, int C[], int M+N){
  int p1=0, p2=0, p3=0;
  while(p1<M && p2<N){
    if(A[p1] < B[p2]){
      C[p3] = A[p1]
      p1++; p3++;
    }
    else{
      C[p3] = B[p2]
      p2++; p3++;
    }
  }
  //copy remaining data from array A or array B
  while(p1 < M){ C[p3++] = A[p1++] }
  while(p2 < N){ C[p3++] = B[p2++] }
}
```

A[ ] = 2 4 5 6

B[ ] = 3 7 8 9 11 14 18 20 22 25

C[ ] = 2 3 4 5 6 7 8 9 10 11

→ Future Section

Monday / Tuesday — Thursday / Friday — Sunday

When ever we will start → Contest discussion

Monday / Tuesday / Friday / Sunday

Concept:

N miracles are any i^{th} → A[i] chap

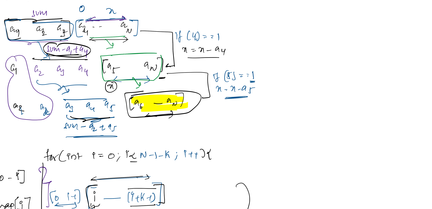
walk[N] →

chap[N] →

pills: K

↳ We can be a weak one

↳ walk[i]



→ 2 prefix array

sum1[ ] ⇒ // sum of all array [0 - i]

sum2[i] ⇒ // $\sum_{j=0}^{i}$ chap[j] + chap[j]

chap[ ] = 2 3 5 3 5 4

chap[ ] = 1 1 0 1 0 0

sum1[ ] = 2 5 10 13 18 22

sum2[ ] = 2 5 5 8 8 8

```
for(int i=0; i<N-1-K; i++){
  sum+= //sum1[i+K-1] - sum1[i-1]
  sum+= //sum2[i-1]
  sum+= //sum2[N-1] - sum2[i+K-1]
}
```

[0 - i-1]

[i - (i+K-1)]

[0 - i-1]

[i+K - N-1]