Lecture # 09

Date: Sept 16, 25 Day: Tuesday

Linear Sorting Algorithms → non-comparison based algo...

if majority of elements in the array is already sorted : ~~line~~ Insertion sort

→ Counting Sort (Linear time)

constraints :

1. Small range
   out of place & stable.

>=
<=
comparison base sorting

if range is larger ; we will not use counting sort bcz T(N) will be in exponential (inefficient)...

2	5	3	0	2	3	0	3
1	2	3	4	5	6	7	8

range = 0 - 5

1. Create an array (frequencies) → size = max of the array.
   → stores frequency of each element

2	0	2	3	0	1
0	1	2	3	4	5

~~2. Copy the elements ...~~

2. Compute the compulative frequencies → to make it stable.
   update the freq array with it..

2	2	4	7	7	8
0	1	2	3	4	5

Arr[i] += Arr[i-1]

3 → last occurance of '3' in the array

0	0	2	2	3	3	3	5
1	2	3	4	5	6	7	8

* if timeComplexity depends on the values of the data :- pseudo polynomials

---

```
CountingSort ( data [1 ...N]) {

1)       K = getMax (data);  — N
       freq [0 ---- k] = {0}  — 1

2)   for ( i = 1 to N )  — N    // count freq of eachelem
           freq [data[i]]++;      treating it as an index

3)    for ( i = 1 to k )  — K
          freq[i] += freq[i-1]

      for  i = N  down to 1     — N     to make it a stable algorithm

      Res [freq [data[i]]] = data[i]

          freq [data[i]] -- ;
```

$T(N) = O(N+k)$

if there are -ive values in the array ; we will pick the minimum. value and add them into the values ----

2	-5	3	0	2	3	0	-3

min = -5 ↓ add kren

7	0	8	5	7	8	5	2

Sort kren

0	2	5	5	7	7	8	8

-5 ↓ subtract kren

-5	-3	0	0	2	2	3	3

Radix Sort

25/09

* digit level sorting, units - tens - thousands...

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

Date:

* should be stable for it to work

```
63          61
61   →      63
67          67
```

* now freq of 6 is 3, if not stable then sorting incorrect
for (i = N downto 1)

```
CountSort ( Arr [1...N] , dpos, N) {

    freq [0 ... 9] = {0}
    for (i = 1 to N) {                          - O(N)
        freq [ (Arr[i]/dpos)% 10 ] ++
    }
   for  i = 1  to 9 {                           - O(1)
        freq[i] += freq[i-1] }

   res [ 1 ... N]
   for ( i = N down to 1 ) {                    - O(N)
        res [ freq [(Arr[i] /dpos) % 10] ] = Arr [i]
        freq [(Arr[i] /dpos) % 10] --
   }
 //copy res into original arr                   - O(N)
}

RadixSort ( Arr [1...N] , N) {

   max = getMax (Arr, N)
   for (dpos = 1; max / dpos > 0; dpos *= 10)
         CountSort (Arr, dpos, N)
}
```

→ $O(d * (N+K))$

$K = 9$

- linear time

- if $d = N$, we have number with same no. of digits ~~d~~ as input size, then $O(N^2)$ → not linear anymore

- if num of digits in input is fixed, like sorting CNICs, tel nums, then radix sort