SORTING BY DISTRIBUTION

These sorting algorithms have a key features, which make these sorting algorithm distinguished from others are :→

1. SORTING WITH KEY COMPARISON
2. RUNNING TIME COMPLEXITY IS O(n).

In other words, in these type of sorting, sorting operation is carried out by mean of distribution based on the constituent components in the elements. This is why a sorting method in this ~~call~~ class is called "SORTING BY DISTRIBUTION".

SORTING BY DISTRIBUTION → RADIX SORT

SORTING BY DISTRIBUTION → BUCKET SORT

SORTING BY DISTRIBUTION → COUNTING SORT

RADIX SORTING

A sorting technique which is based on Radix or Base of constituent elements in keys is called Radix Sort.

NUMBER SYSTEM	RADIX	BASE	EX
BINARY	0 and 1	2	0100, 1111
DECIMAL	0, 1, 2, ..., 9	10	326, 1508
ALPHABETIC	a, b, .., z A, B, ... Z	26	MALAY
ALPHA-NUMERIC	A..Z & 0...9	36	5CS5201

RADIXES IN SOME NUMBER SYSTEMS

- Basic principle of Radix Sort come from the primitive sorting method used in the card sorting machine.

- Motivated with the mechanism of card sorting machine, P. Hilderbrandt, H. Isbitz, H. Rising and J. Schwartz proposed the idea of Radix Sort.

<u>ALGORITHM RADIXSORT</u>

Input: An array A[1, 2, .., n], where n is the number of elements.

Output: n elements are arranged in A in ASCENDING ORDER.

Remark: b no. of auxillary arrays each of size n is used.

Steps :-

```
/* For each base in the number system */
For i=1 to c do  ——→ // c denotes the most significant position.
    For j=1 to n do  ——→ // for all elements in the array A.
        x = Extract (A[j], i) → // Get the i^th component in the j^th
                                   element.
        Enque (Q_x, A[j])
    ENDFor
/* Combine all elements from all auxiliary arrays to A (assume A is empty */
For k=0 to (b-1) do
    While Q_k is not empty do
        y = Dequeue (Q_k)
        Insert (A, y)
    EndWhile
EndFor
EndFor
Stop.
```

Now, let us formulate an algorithm for Radix sort.
Let us consider a number system with base b, that is, the constituent element in any key value ranges b/w 0, 1, 2, .. (b-1)

Further assume that number of components in a key value is c and n number of keys values in there in the input list.

The output list will be in Ascending Order of the key values.
In other words, let n keys values are stored in an array $A[1, 2, \dots n]$ and any key $A[i]_{1 \le i \le n} = a_{i1}, a_{i2}, \dots a_{ic}$ where $0 \le a_{ij} < b$ for all $1 \le j \le c$.

In the Radix Sort method, other than the input array, we will need b auxiliary queues. Let the queues be $Q_0[1 \dots n], Q_1[1 .. n], \dots, Q_{b-1}[1 .. n]$.
Note that the maximum size of each queue is n, which is the no. of key ~~value~~ values in the i/p list.

ILLUSTRATION OF THE ALGORITHM RADIXSORT

→ The outermost loop (Step 1 - step 12) in the algorithm RadixSort iterates c no. of items.

→ In each iteration, it distributes n elements into b number of Queues. This is done through an inner iteration (Step 2 - Step 5).

→ Next, starting from the Queue Q_0 to Q_{b-1} all elements a dequeued one by one and inserted into the Array A. This is done in another inner loop, namely, steps 6-11.

→ When the outermost loop (Steps 1-12) completes it all runs, elements appear in the Array A in Ascending Order.

→ In the above mentioned, algorithm, we assume the procedure Enque(Q, X) to enter an element x into an array Q in FIFO (queue order) and y = Dequeue(Q) the procedure to delete an element from Q in FIFO order.

→ Another procedure Extract (A, i) returns the ith component in the element A.

→ The algorithm RadixSort is illustrated in fig. 10. with 10 numbers in decimal system and each number is of 3 digit.

- → There are 3 passes in the algorithms.
- → The different tasks in the first pass are shown in Fig. 10.1 to fig. 10.3.
- → An i/p list containing 10 number each of three digit in Fig 10.1.
- → The result of execution of the INNER LOOP in steps 6-11. This ends the first pass.
- → POINT TO BE NOTED :-

  All elements are arranged in the array A according to the ascending order of their least significant digits.

- → The arrangement of elements in the array A as obtained in the first pass becomes input list to the second pass.

→ Distribution of elements and combining there after in the second pass are shown in Fig 10.4 - Fig 10.5 respectively. At the end of the second pass all the elements in the array A are arranged in Ascending order with respect to their last 2 digits.

→ The result of the third pass is shown in fig 10.6 to fig. 10.7

Finally list appeared in Sorted Order.

A: 136, 487, 358, 469, 570, 247, 598, 639, 205, 609

(a) Input list

$Q_0 \rightarrow$ 570

$Q_1 \rightarrow$

Q_2

Q_3

Q_4

$Q_5 \rightarrow$ 205

$Q_6 \rightarrow$ 136

$Q_7 \rightarrow$ 487, 247

$Q_8 \rightarrow$ 358, 598

$Q_9 \rightarrow$ 469, 639, 609

(b) Distribution of element into 10 Auxiliary Arrays.

A: 570, 205, 136, 487, 247, 358, 598, 469, 639, 609

$Q_0 \rightarrow$ 205, 609

Q_1

Q_2

$Q_3 \rightarrow$ 136, 639

$Q_4 \rightarrow$ 247

$Q_5 \rightarrow$ 358

$Q_6 \rightarrow$ 469

Q_7 – 570

Q_8 – 487

Q_9 – 598

(b) Distribution of element in 10 auxiliary arrays in Pass 2.

A: 205, 609, 136, 639, 247, 358, 469, 570, 487, 598.

$Q_0 \rightarrow$

$Q_1 \rightarrow$ 136

$Q_2 \rightarrow$ 205 247

$Q_3 \rightarrow$ 358

$Q_4 \rightarrow$ 469, 487

$Q_5 \rightarrow$ 570, 598

$Q_6 \rightarrow$ 609, 639

$Q_7 \rightarrow$

$Q_8 \rightarrow$

$Q_9 \rightarrow$

A: 136, 205, 247, 358, 469, 487, 570, 598, 609, 639,

Radix Sort

Introduction:-

→ Radix Sort → linear sorting algo. for integers
→ uses the concept of sorting names in Alphabetical Order.

→ Radix Sort a.k.a Bucket Sort.

→ Observe that words are first sorted algo. that sort according to the first letter of the name. That is, 26 classes are used to arrange the names, where the first class stores the names that begin with A, the second class contains the name with B, and so on.

→ During the second pass, names are grouped according to the second letter. After the second pass, names are sorted on the first two letters. This process is continued till the n^{th} pass, where n is the length of the name with maximum number of letters.

→ After every pass, all the names are collected in order of Buckets. That is, first pick up the names in the first bucket that contains the names from the second bucket and so on.

→ When the Radix Sort is used on INTEGERS, SORTING is done on each of the digits in the numbers. The sorting procedure proceeds by sorting the least significant to the most significant digits.

→ Section
→ Venue - 1 and 2
→ CS4A - Class Room No - C-12./Lab-
→

ALGORITHM FOR RADIX SORT (ARR,N)

Step1: Find the largest number in ARR as LARGE

Step2: [INITIALIZE] SET NOP= Number of digits in LARGE

Step3: SET PASS=0

Step4: Repeat Step 5 while PASS <= NOP-1

Step5: SET I=0 and INITIALIZE Buckets

Step6: Repeat Steps 7 to 9 while I < N-1

Step7: SET DIGIT = digit at PASSth place in A[I]

Step8: Add A[I] to the bucket numbered DIGIT

Step9: INCREMENT bucket count for Bucket numbered DIGIT

[END OF LOOP]

Step 10: collect the number of Bucket

[END OF LOOP]

Step 11: END

COMPLEXITY OF RADIX SORT

→ To calculate the complexity of Radix Sort Algo., assume there are n numbers that have to be Sorted and k is the number of digits in the largest number.

In this case, the radix sort algorithm is called a total of k times.

The inner loop is executed n times. Hence, the entire radix sort algorithm takes O(kn) times to execute.

→ When applied on a data set of finite size, then the algo. runs in O(n) Asymptotic Time.

In the first pass, the number are sorted according to digits at Ones place.

Numbers	0	1	2	3	4	5	6	7	8	9
345						345				
654					654					
924					924					
123				123						
567								567		
472			472							
555						555				
808									808	
911		911								

After this pass, the number are collected bucket by bucket. The new list thus formed is used as an input for the next pass. In the second pass, the numbers are sorted according to the digit at the tens place.

NO.	0	1	2	3	4	5	6	7	8	9
911		911								
472			~~472~~					472		
123			123							
654						654				
924			924							
345					345					
555						555				
567							567			
808	808									

In the third pass, the numbers are sorted according to the digit at hundered places.

NUMBER	0	1	2	3	4	5	6	7	8	9	
808									808		
911										911	
123		123									
924										924	~~92~~
345				345							
654							654				
555						555					
567						567	~~567~~				
472					472						

The numbers are collected by the bucket. The new list formed is the final sorted result.
After the third pass, the list can be given as

123, 345, 472, 555, 567, 654, 808, 911, 924

FINAL RESULT.

PROS AND CONS OF RADIX SORT

→ Radix Sort is one of the fastest sorting algo. for numbers or strings of letters.

→ But there are certain trade-offs for radix sort that can make it less preferable as compared to other sorting algorithms.

→ Radix Sort takes more space than other sorting algorithms. Besides the array of numbers, we need 10 Buckets to sort numbers, 26 buckets to sort strings containing only characters, and at least 40 buckets to sort a string containing alphanumeric characters.

→ Another drawbacks → Radix Sort → it dependent on digits or letter. This feature comprises with the flexibility to sort input of any data type.

Code →

```
# include <stdio.h>
# include <conio.h>
# define size 10
int largest (int arr[], int n);
void radix-sort (int arr[],int n);
void main()
 {
  int arr[size], i, n;
  printf ("\n Enter the no. of elements in the array:");
  scanf ("%d", &n);
  printf ("\n Enter the number of the array:");
  for (i=0; i<n; i++)
    {
      scanf ("%d", &arr[i]);
    }
  radix_sort (arr, n);
  printf ("\n The sorted array is: \n");
  for (i=0; i<n; i++)
  printf ("%d\t", arr[i]);
  getch();
 }

int largest (int arr[], int n)
 { int large= arr[0], i;
   for (i=1; i<n; i++)
    {
      if (arr[i] >large)
         large = arr[i];
    }
    return large;
 }
```

```
void radix_sort(int arr[], int n)
{
  int bucket[size][size], bucket_count[size];
  int i, j, k, remainder, NOP=0, divisor=1, large, pass;
  large = largest(arr, n);
  while (large>0)
  {
    NOP++;
    large /= size;
  }
  for (pass=0; pass<NOP; pass++)   // INITIALIZE THE BUCKET.
  {
    for (i=0; i<size; i++)
      bucket_count[i]=0;
    for (i=0; i<n; i++)
    {
       // sort the numbers according to the digits at passth place
       remainder = (arr[i]/divisor) % size;
       bucket[remainder][bucket_count[remainder]] = arr[i];
       bucket_count[remainder] += 1;
    }
    // Collect the numbers after PASS pass.
       i=0;
       for (k=0; k<size; k++)
       {
          for(j=0; j<bucket_count[k]; j++)
          {
             arr[i] = bucket[k][j];
             i++;
          }
       }
       divisor *= size;
} } }
```

Analysis of Radix Sort

- Not involve any comparison of key. Further, it was also mentioned that the runtime complexity of the radix sort complexity in $O(n)$.

- RUN TIME = mainly due to two operations
  - (a) Distribution of key elements.
  - (b) Combination.

  It can be easily checked that the RUN TIME remains INVARIANT irrespective of the order of the elements in the list.

- Time Requirement

Let a = time to extract a component from an element.
e = time to enqueue an element in an array.
d = time to dequeue an element from an array.

Time for DISTRIBUTION OPERATION = $(a+e)n$ [in step 2-5]
Time for COMBINATION = $(d+e)n$ [in Step 6-12]
Since these two operations are iterated c times (steps 1-12), the total time of computation is given by

$$T(n) = \{(a+e)\times n + (d+e)\times n\} \times c$$
$$= (a+d+2e)\times c \times n$$

Since a, d, e and c all are constants for a number system we have $T(n) = O(n)$.

Storage Space Requirement

- Radix sort is not an INPLACE sorting method.
- It requires b auxilliary array to maintain b queues, where b denotes the base of the number system.
- The size of each array is n, so that in Worst Case it can accomodate all n elements.

Thus, the total space storage required in Radix sort is

$$S(n) = b \times n$$

Thus $S(n) = O(n)$, b being a constant.

ANALYSIS OF THE ALGO. RADIX SORT

Memory	Time	Remark
$S(n) = b \times n$	$T(n) = (a+d+2e) \cdot c \cdot n$	b - denotes the base of the number system, a, c, d, e are constant.

Storage and Time Complexities

Complexity For	Complexity	Remark
Memory	$S(n) = O(n)$	Irrespective of the arrangement of elements.
Time	$T(n) = O(n)$	