Sairam INSTITUTIONS

Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

YEAR	SEM
II	III

CS8391

DATA STRUCTURES

(Common to CSE & IT)

UNIT No.5

5.SEARCHING ,SORTING AND HASHING TECHNIQUE

5.4 RADIX SORT - HASHING - HASH FUNCTIONS

Version: 1.XX

5.4 RADIX SORT - HASHING - HASH FUNCTIONS

RADIX SORT

Radix sort is one of the sorting algorithms used to sort a list of integer numbers in order. In radix sort algorithm, a list of integer numbers will be sorted based on the digits of individual numbers. Sorting is performed from least significant digit to the most significant digit.

Radix sort algorithm requires the number of passes which are equal to the number of digits present in the largest number among the list of numbers. For example, if the largest number is a 3 digit number then that list is sorted with 3 passes.

Step by Step Process

The Radix sort algorithm is performed using the following steps...

Step 1 - Define 10 queues each representing a bucket for each digit from 0 to 9.

Step 2 - Consider the least significant digit of each number in the list which is to be sorted.

Step 3 - Insert each number into their respective queue based on the least significant digit.

Step 4 - Group all the numbers from queue 0 to queue 9 in the order they have inserted into their respective queues.

Step 5 - Repeat from step 3 based on the next least significant digit.

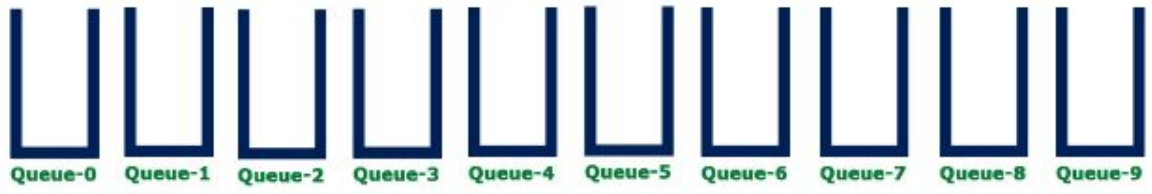
Step 6 - Repeat from step 2 until all the numbers are grouped based on the most significant digit.

Example

Consider the following list of unsorted integer numbers

82, 901, 100, 12, 150, 77, 55 & 23

Step 1 - Define 10 queues each represents a bucket for digits from 0 to 9.



Pass - 1

Step 2 - Insert all the numbers of the list into respective queue based on the Least significant digit (once placed digit) of every number.

82, 901, 100, 12, 150, 77, 55 & 23

Group all the numbers from queue-0 to queue-9 inthe order they have inserted & consider the list for next step as input list.

100, 150, 901, 82, 12, 23, 55 & 77

Pass - 2

Step 3 - Insert all the numbers of the list into respective queue based on the next Least significant digit (Tens placed digit) of every number.

100, 150, 901, 82, 12, 23, 55 & 77

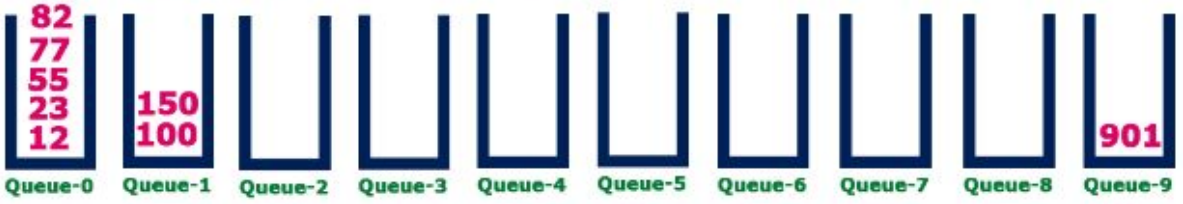
Group all the numbers from queue-0 to queue-9 inthe order they have inserted & consider the list for next step as input list.

100, 901, 12, 23, 150, 55, 77 & 82

Step 4 - Insert all the numbers of the list into respective queue based on the next Least significant digit (Hundres placed digit) of every number.

100, 901, 12, 23, 150, 55, 77 & 82

Pass - 3



Group all the numbers from queue-0 to queue-9 inthe order they have inserted & consider the list for next step as input list.

12, 23, 55, 77, 82, 100, 150, 901

List got sorted in the incresing order.

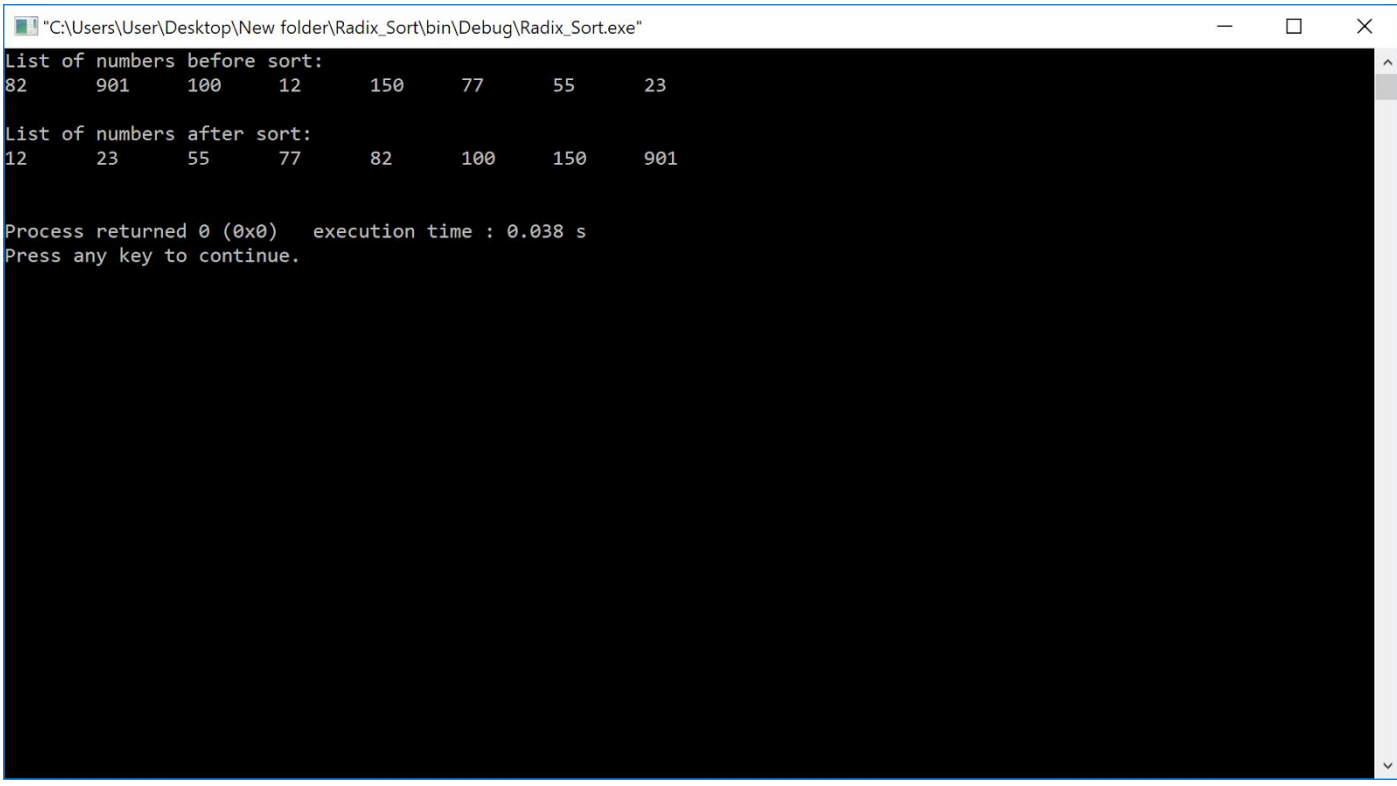
Implementation of Radix Sort

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

int getMax(int list[], int n) {
   int mx = list[0];
   int i;
   for (i = 1; i < n; i++)
      if (list[i] > mx)
         mx = list[i];
   return mx;
}
void countSort(int list[], int n, int exp) {
   int output[n];
   int i, count[10] = { 0 };
   for (i = 0; i < n; i++)
      count[(list[i] / exp) % 10]++;
```

```
    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];
    for (i = n - 1; i >= 0; i--) {
        output[count[(list[i] / exp) % 10] - 1] = list[i];
        count[(list[i] / exp) % 10]--;
    }
    for (i = 0; i < n; i++)
        list[i] = output[i];
}

void radixsort(int list[], int n) {
    int m = getMax(list, n);
    int exp;
    for (exp = 1; m / exp > 0; exp *= 10)
        countSort(list, n, exp);
}
void print(int list[], int n) {
    int i;
    for (i = 0; i < n; i++)
        printf("%d\t", list[i]);
}
int main()
{
    int list[] = { 82, 901, 100, 12, 150, 77, 55, 23 };
    int i, n = sizeof(list) / sizeof(list[0]);
    printf("List of numbers before sort: \n");
    for(i = 0; i<8; i++)
        printf("%d\t", list[i] );
    radixsort(list, n);
    printf("\n\nList of numbers after sort: \n");
```

```
    print(list, n);
    printf("\n\n");
    return 0;
}
```

Output



HASHING

Hashing is a technique that is used to store, retrieve and find data in the data structure called Hash Table. It is used to overcome the drawback of Linear Search (Comparison) & Binary Search (Sorted order list). It involves two important concepts-

- Hash Table
- Hash Function

Hash table

- A hash table is a data structure that is used to store and retrieve data (keys) very quickly.
- It is an array of some fixed size, containing the keys. Hash table run from 0 to Tablesize – 1.
- Each key is mapped into some number in the range 0 to Tablesize – 1.

This mapping is called a Hash function.

· Insertion of the data in the hash table is based on the key value obtained from the hash function.

· Using the same hash key value, the data can be retrieved from the hash table by few or more Hash key comparisons.

· The load factor of a hash table is calculated using the formula: (Number of data elements in the hash table) / (Size of the hash table)

Factors affecting Hash Table Design

· Hash function

· Table size

· Collision handling scheme

Hash function:

It is a function, which distributes the keys evenly among the cells in the Hash Table.

Using the same hash function we can retrieve data from the hash table. Hash function is used to implement a hash table.

The integer value returned by the hash function is called a hash key.

If the input keys are integer, the commonly used hash function is

H (key) = key % Tablesize

A simple hash function

```
typedef unsigned int index;
index Hash ( const char *key , int Tablesize )
{
  unsigned int Hashval = 0 ;
  while ( * key ! = „ \0 „ )
      Hashval + = * key ++ ;
      return ( Hashval % Tablesize ) ;
}
```

Types of Hash Functions

1. Division Method

2. Mid Square Method
3. Multiplicative Hash Function
4. Digit Folding

1. Division Method:

It depends on the remainder of division.

Divisor is Table Size.

Formula is **(H (key) = key % table size)**

E.g. consider the following data or record or key (36, 18, 72, 43, 6) table size = 8

Assume a table with 8 slots:

Hash key = key % table size

4 = 36 % 8

2 = 18 % 8

0 = 72 % 8

3 = 43 % 8

6 = 6 % 8

[0]	72
[1]	
[2]	18
[3]	43
[4]	36
[5]	
[6]	6
[7]	

2. Mid Square Method:

We first square the item, and then extract some portion of the resulting digits. For example, if the item were 44, we would first compute 44^2=1,936. Extract the middle two digit 93 from the answer. Store the key 44 in the index 93.

3. Multiplicative Hash Function:

Key is multiplied by some constant value.

Hash function is given by,

H(key)=Floor (P * (key * A)) P = Integer constant [e.g. P=50]

A = Constant real number [A=0.61803398987],suggested by Donald Knuth to use this constant

E.g. Key 107 H(107)=Floor(50*(107*0.61803398987))

=Floor(3306.481845)

H(107)=3306

4.Digit Folding Method:

The folding method for constructing hash functions begins by dividing the item into equal-size pieces (the last piece may not be of equal size). These pieces are then added together to give the resulting hash key value. For example, if our item was the phone number 436-555- 4601, we would take the digits and divide them into groups of 2 (43, 65, 55, 46, 01). After the addition, 43+65+55+46+01, we get 210. If we assume our hash table has 11 slots, then we need to perform the extra step of dividing by 11 and keeping the remainder. In this case 210 % 11 is 1, so the phone number 436-555-4601 hashes to slot 1.

Collision:

If two more keys hashes to the same index, the corresponding records cannot be stored in the same location. This condition is known as collision.

Characteristics of Good Hashing Function:

- It should be Simple to compute.
- Number of Collision should be less while placing record in Hash Table.
- Hash function with no collision è Perfect hash function.
- Hash Function should produce keys which are distributed uniformly in hash table.

The hash function should depend upon every bit of the key. Thus the hash function that simply extracts the portion of a key is not suitable.

Collision Resolution Strategies / Techniques (CRT):

If collision occurs, it should be handled or overcome by applying some technique. Such a technique is called CRT.

There are a number of collision resolution techniques, but the most popular are:

- **Separate chaining (**Open Hashing**)**
- **Open addressing**. (Closed Hashing)

1.LinearProbing

2.Quadratic

3.ProbingDouble hashing