DSA ASSIGNMENT - 2

Name: Vaishnav Eega

Roll No: BT22CSH019

IMPLEMENTATION OF RADIX SORT USING ARRAYS:

*#include* <stdio.h>

*#include* <stdlib.h>

*void* *display\_array*(*int* a[], *int* n) {

*printf*("size: %d\n", n);

*for* (*int* *i* *=* 0; *i* *<* n; *i++*) {

*printf*("%d ", a[*i*]);

    }

*printf*("\n");

}

*void* *count\_sort*(*int* a[], *int* n, *int* pos) {

*int* *c*[10] *=* {0};

*for* (*int* *i* *=* 0; *i* *<* n; *i++*)

*c*[(a[*i*] */* pos) *%* 10]*++*;

*for* (*int* *i* *=* 1; *i* *<* 10; *i++*)

*c*[*i*] *+=* *c*[*i* *-* 1];

*int\** *b* *=* (*int\**)*malloc*(n *\** *sizeof*(*int*));

*for* (*int* *i* *=* n *-* 1; *i* *>=* 0; *i--*) {

*b*[*--c*[(a[*i*] */* pos) *%* 10]] *=* a[*i*];

    }

*for* (*int* *i* *=* 0; *i* *<* n; *i++*) {

        a[*i*] *=* *b*[*i*];

    }

*free*(*b*);

}

*void* *radixsort*(*int* a[], *int* n) {

*int* *max* *=* a[0];

*for* (*int* *i* *=* 1; *i* *<* n; *i++*) {

*if* (a[*i*] *>* *max*) {

*max* *=* a[*i*];

        }

    }

*int* *pos* *=* 1;

*while* ((*max* */* *pos*) *>* 0) {

*count\_sort*(a, n, *pos*);

*pos* *\*=* 10;

    }

}

*int* *main*() {

*int* *n*;

*printf*("Enter number of elements in array");

*scanf*("%d", *&n*);

*int* *a*[*n*];

*for*(*int* *i=*0;*i<n*;*i++*)

    {

*printf*("Enter element %d: " ,*i+*1);

*scanf*("%d", *&a*[*i*]);

    }

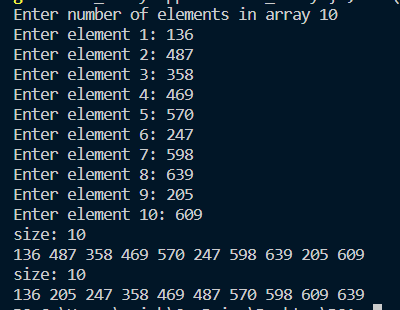
*display\_array*(*a*, *n*);

*radixsort*(*a*, *n*);

*display\_array*(*a*, *n*);

}

OUTPUT :



TIME COMPLEXITY:

O(d\*n)

Where d is no of digits in max element of the array and n is the number of elements in the array

Best case:

is when the elements of the array almost have same no of digits and the distribution into buckets is uniform. In this case, each digit iteration (looping through the array once) will take linear time, and there will be a constant number of iterations (equal to the number of digits in the maximum element). Therefore, the time complexity in the best case is O(d \* n) .

Worst case:

The worst case occurs when the distribution into buckets is uneven, leading to more iterations. However, the number of iterations is still bounded by the number of digits. So, the worst-case time complexity is also O(d \* n).

Average Case:

The average case time complexity also turns out to be O(d \* n)

IMPLEMENTATION OF RADIX SORT USING LINKED LIST:

*#include* <stdio.h>

*#include* <stdlib.h>

*struct* node

{

*int* *val*;

*struct* node *\*next*;

};

*void* *display\_array*(*int* a[], *int* n)

{

*printf*("size : %d\n", n);

*for* (*int* *i* *=* 0; *i* *<* n; *i++*)

    {

*printf*("%d ", a[*i*]);

    }

*printf*("\n");

}

*void* *printll*(*struct* node *\**head)

{

*if* (*!*head)

    {

*printf*("empty ll\n");

*return*;

    }

*struct* node *\*ptr* *=* head;

*while* (*ptr* *!=* *NULL*)

    {

*printf*("%d ", *ptr*->*val*);

*ptr* *=* *ptr*->*next*;

    }

*printf*("\n");

}

*struct* node *\*insert*(*struct* node *\**head, *int* value)

{

*if* (head *==* *NULL*)

    {

        head *=* (*struct* node *\**)*malloc*(*sizeof*(struct node));

        head->*val* *=* value;

        head->*next* *=* *NULL*;

    }

*else*

    {

*struct* node *\*temp* *=* (*struct* node *\**)*malloc*(*sizeof*(struct node));

*temp*->*val* *=* value;

*temp*->*next* *=* *NULL*;

*struct* node *\*ptr* *=* head;

*while* (*ptr*->*next* *!=* *NULL*)

        {

*ptr* *=* *ptr*->*next*;

        }

*ptr*->*next* *=* *temp*;

    }

*return* head;

}

*void* *bucketsort*(*int* a[], *int* n, *int* pos)

{

*struct* node *\*bins*[10];

*for* (*int* *i* *=* 0; *i* *<* 10; *i++*)

    {

*bins*[*i*] *=* *NULL*;

    }

*for* (*int* *i* *=* 0; *i* *<* n; *i++*)

    {

*int* *digit* *=* (a[*i*] */* pos) *%* 10;

*bins*[*digit*] *=* *insert*(*bins*[*digit*], a[*i*]);

    }

*int* *i* *=* 0;

*int* *j* *=* 0;

*while* (*i* *<* 10)

    {

*while* (*bins*[*i*] *!=* *NULL*)

        {

            a[*j++*] *=* *bins*[*i*]->*val*;

*struct* node *\*temp* *=* *bins*[*i*];

*bins*[*i*] *=* *bins*[*i*]->*next*;

*free*(*temp*);

        }

*i++*;

    }

}

*void* *radixsort*(*int* a[], *int* n)

{

*int* *max* *=* a[0];

*for* (*int* *i* *=* 1; *i* *<* n; *i++*)

    {

*if* (a[*i*] *>* *max*)

        {

*max* *=* a[*i*];

        }

    }

*int* *pos* *=* 1;

*while* (*max* *>* 0)

    {

*bucketsort*(a, n, *pos*);

*display\_array*(a, n);

*pos* *\*=* 10;

*max* */=* 10;

    }

}

*int* *main*()

{

*int* *a*[] *=* {136, 487, 358, 469, 570, 247, 598, 639, 205, 609};

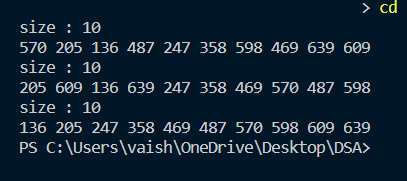
*int* *n* *=* *sizeof*(*a*) */* *sizeof*(*int*);

*radixsort*(*a*, *n*);

*return* 0;

}

OUTPUT :



TIME COMPLEXITY:

O(d\*n)

Best Case: In the best case, the distribution of elements across the buckets is perfectly balanced for every digit position. The time complexity for inserting each element into its bucket is O(1), and the number of iterations is determined by the number of digits in the maximum element. Therefore, the best-case time complexity is O(d \* n), where d is the number of digits in the maximum element.

Worst Case: In the worst case, the distribution is uneven, and elements are all placed in the same bucket for each digit position. The insertion into buckets takes linear time for each element, and there are d iterations for d-digit numbers. So, the worst-case time complexity is still O(d \* n).

Average Case: The average case time complexity is also O(d \* n), where d is the average number of digits in the elements. This remains consistent with the best and worst cases.