**Lab3**

**Prasanna Natarajan**

**1410110298**

**Code:**

Lab3.c

/\*

Name: Prasanna Natarajan

Roll No.: 1410110298

Description: Compares three kinds of sorting algorithm namely, Bubble Sort, Merge Sort and Radix Sort.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

// function declarations

void bubbleSort(int a[],int n);

void radixSort(int a[], int n);

void merge(int a[], int l, int m, int r);

void join(int a[], int l, int r);

void mergeSort(int a[], int l, int r);

int main()

{

int a[100000]; // input array

int i = 0;

srand(1);

for(i=0;i<100000;i++){

a[i]=(rand()%1000); // filling the input array with random numbers

}

clock\_t begin1 = clock();

//bubbleSort(a,1000);

radixSort(a,100000); // calling radix sort with input array and size of the array

//mergeSort(a,0,5);

clock\_t end1 = clock();

double time\_spent = (double)(end1-begin1)/CLOCKS\_PER\_SEC;

printf("Time for execution of radix sort = %lf\n",time\_spent);

clock\_t begin2 = clock();

bubbleSort(a,100000); // calling bubble sort with input array and size of the array

//radixSort(a,1000);

//mergeSort(a,0,5);

clock\_t end2 = clock();

double time\_spent2 = (double)(end2-begin2)/CLOCKS\_PER\_SEC;

printf("Time for execution of bubble sort = %lf\n",time\_spent2);

clock\_t begin3 = clock();

//radixSort(a,1000);

mergeSort(a,0,100000);// calling merge sort with input array, left most element's index and right most element's index+1

clock\_t end3 = clock();

double time\_spent3 = (double)(end3-begin3)/CLOCKS\_PER\_SEC;

printf("Time for execution of merge sort = %lf\n",time\_spent3);

return 0;

}

// bubbleSort

// inputs : array to be sorted

// size of the array

void bubbleSort(int a[],int n){

int i,j,temp;

for(i=0;i<n-1;i++){

for(j=0;j<n-i-1;j++){

if(a[j] >= a[j+1]){

temp = a[j];

a[j] = a[j+1];

a[j+1] = temp;

}

}

}

/\*for(i=0;i<n;i++)

printf("%d ",a[i]); // printing the values of sorted array for debuging

printf("\n");\*/

}

// radixSort

// inputs : array to be sorted

// size of the array

void radixSort(int a[], int n){

int i,temp,j,k;

int c[10],div=10,counter=0;

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

c[i] = 0;

int space[10][100000]; // to store the hash table

for(i=0;i<10;i++){

for(j=0;j<100000;j++){

space[i][j] = -1;

}

}

for(j=1;j<4;j++){ // restircting j to 4 since the range of inputs in from 0 to 999

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

temp = a[i]%((int)pow(div,j));

if(j>=2) temp = temp/(int)pow(div,j-1);

//printf("temp = %d\n",temp);

space[temp][c[temp]++] = a[i];

}

for(i=0;i<10;i++){

for(k=0;k<999999;k++){

if(space[i][k] != -1){

//printf("counter = %d\t",counter);

a[counter] = space[i][k];

//printf("a = %d\n",a[counter]);

counter++;

}

}

}

counter = 0;

for(i=0;i<10;i++){

for(k=0;k<999999;k++){

space[i][k] = -1;

}

}

}

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

printf("%d ",a[i]); // printing for debugging

}

printf("\n");\*/

}

// merge function: to sort the elements in the right order

void merge(int a[], int l, int m, int r){

int i,j,k;

int Left[m-l+1], Right[r-m];

for (i = 0; i < m-l+1; i++)

Left[i] = a[l + i];

for (j = 0; j < r-m; j++)

Right[j] = a[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < m-l+1 && j < r-m){

if (Left[i] <= Right[j]){

a[k] = Left[i];

i++;

}

else{

a[k] = Right[j];

j++;

}

k++;

}

while (i < m-l+1){

a[k] = Left[i];

i++;

k++;

}

while (j < r-m){

a[k] = Right[j];

j++;

k++;

}

}

// join function: Recursive function to split the elements into two halves

void join(int a[], int l, int r){

if (l < r){

int m = l+(r-l)/2;

join(a, l, m);

join(a, m+1, r);

merge(a, l, m, r);

}

}

// mergeSort

// inputs : array to be sorted

// index of left most and right most element

void mergeSort(int a[], int l, int r){

join(a,l,r);

int i=0;

/\*for(i=0;i<r;i++)

printf("%d ",a[i]); // printing for debuggins

printf("\n");\*/

}

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **time by bubble sort** | **time by merge sort** | **time by radix sort** |
| **1000** | 0 | 0 | 0 |
| **10000** | 0.484375 | 0 | 0 |
| **100000** | 43.640625 | 0.05125 | 0.046875 |

**Results:**