Name : Rahul Thapar

ID : 1410110321

Write three functions separately that the main function calls one after another, where each function sorts the same array using a different algorithm, namely:

1. Radix sort

2. Bubble sort

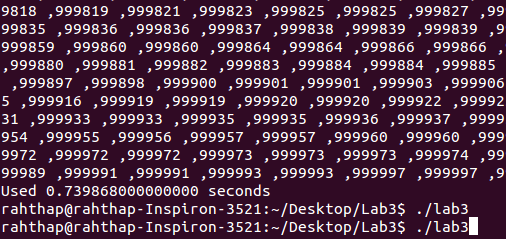
3. Merge sort

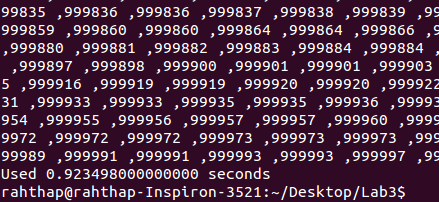
Compute the average run time for each of the three techniques separately, for n = 103, 106

Solution :

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Sorting Algorithm | Data Set  (value of n) | Time Taken (seconds) |
| 1. | Radix Sort | 103 | 0.000818000000000 |
| 2. | Bubble Sort | 103 | 0.002820000000000 |
| 3. | Merge Sort | 103 | 0.000321000000000 |

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Sorting Algorithm | Data Set  (value of n) | Time Taken (seconds) |
| 1. | Radix Sort | 106 | 0.739868000000000 |
| 2. | Bubble Sort | 106 | Unable to process |
| 3. | Merge Sort | 106 | 0.923498000000000 |





CODE :

/\*\*\*

\*

\* @author : Rahul Thapar

\* Date : 25 January 2017

\* ID : 1410110321

\*

\*\*\*/

#include<stdio.h>

#include<string.h>

#include<time.h>

#include<unistd.h>

#define MAX 1000000

void printArray(int \* array, int size){

int i;

printf("{ ");

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

printf("%d ,", array[i]);

printf("}\n");

}

int findlargest\_Number(int \* array, int size){

int i;

int largest\_Number = -1;

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

if(array[i] > largest\_Number)

largest\_Number = array[i];

}

return largest\_Number;

}

void partition(int arr[],int lower\_value,int higher\_value){

int mid;

if(lower\_value<higher\_value){

mid=(lower\_value+higher\_value)/2;

partition(arr,lower\_value,mid);

partition(arr,mid+1,higher\_value);

mergeSort(arr,lower\_value,mid,higher\_value);

}

}

// RADIX SORT

radixSort(int \* array, int size){

printf("\n\nRunning Radix Sort .........!\n\n");

int i;

int semiSorted[size];

int significant\_Digit = 1;

int largest\_Number = findlargest\_Number(array, size);

while (largest\_Number / significant\_Digit > 0){

int bucket[10] = { 0 };

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

bucket[(array[i] / significant\_Digit) % 10]++;

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

bucket[i] += bucket[i - 1];

for (i = size - 1; i >= 0; i--)

semiSorted[--bucket[(array[i] / significant\_Digit) % 10]] = array[i];

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

array[i] = semiSorted[i];

significant\_Digit \*= 10;

}

printArray(array, size);

}

// BUBBLE SORT

bubbleSort(int \*array,int size){

printf("\n\nRunning bubble Sort .........!\n\n");

int c,d,swap;

for (c = 0 ; c < ( size - 1 ); c++)

{

for (d = 0 ; d < size - c - 1; d++)

{

if (array[d] > array[d+1]) /\* For decreasing order use < \*/

{

swap = array[d];

array[d] = array[d+1];

array[d+1] = swap;

}

}

}

printArray(array,size);

}

// MERGE SORT

mergeSort(int arr[],int lower\_value,int mid,int higher\_value){

int i,m,k,l,flag[MAX];

l=lower\_value;

i=lower\_value;

m=mid+1;

while((l<=mid)&&(m<=higher\_value)){

if(arr[l]<=arr[m]){

flag[i]=arr[l];

l++;

}

else{

flag[i]=arr[m];

m++;

}

i++;

}

if(l>mid){

for(k=m;k<=higher\_value;k++){

flag[i]=arr[k];

i++;

}

}

else{

for(k=l;k<=mid;k++){

flag[i]=arr[k];

i++;

}

}

for(k=lower\_value;k<=higher\_value;k++){

arr[k]=flag[k];

}

}

void main()

{

int array[MAX];

int i,n=0;

int size;

clock\_t start, stop;

//FILE\* f1 = fopen("Random.txt", "r"); // Corresponds to data set of n = 10^3

FILE\* f1 = fopen("Random\_1.txt", "r"); // Corresponds to data set of n = 10^6

i=0;

while( fscanf(f1, "%d,", &n) > 0 ) // parse %d follower\_valueed by ','

{

array[i++] = n;

}

start = clock();

radixSort(array,MAX);

stop = clock();

printf("Used %0.15f seconds \n", (double)(stop-start)/CLOCKS\_PER\_SEC);

start = clock();

bubbleSort(array,MAX);

stop = clock();

printf("Used %0.15f seconds \n", (double)(stop-start)/CLOCKS\_PER\_SEC);

//mergeSort(array,MAX);

start = clock();

partition(array,0,MAX-1);

printf("\n\nRunning Merge Sort .........!\n\n");

printArray(array,MAX);

stop = clock();

printf("Used %0.15f seconds \n", (double)(stop-start)/CLOCKS\_PER\_SEC);

}