PRACTICAL – 3

Write user defined functions for the following sorting methods and compare their performance by time measurement with random data and Sorted data.

1. Selection Sort
2. Bubble Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort

CODE:

#include<stdio.h>

#include<time.h>

#include <stdlib.h>

void selection(int arr[], int n) ;

void printArr(int arr[], int n);

int main(){ int n=100;

int arr[n];

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

arr[i]=rand();

}

for(int i=0;i<n;i++){ printf("%d\t",arr[i]);

}

clock\_t t\_start,t\_end,t\_mid; t\_start = clock(); //starting time after initializing data

printf("t\_start=%lu\n",t\_start);

selection( arr, n);

t\_mid=clock(); //mid time after function call printf("t\_mid=%lu\n",t\_mid);

t\_end = t\_mid - t\_start; //ending time after mid - start

printf("t\_end=%lu\n",t\_end);

double time\_taken = ((double)t\_end)/CLOCKS\_PER\_SEC; //clock per sec is a macro

and 1cps= 1 million micro sec

printf("fun() took %f seconds to execute \n", time\_taken);

printArr(arr, n);

return 0;

}

void selection(int arr[], int n)

{ int i, j, small;

for (i = 0; i < n-1; i++) // One by one move boundary of unsorted subarray

{

small = i; //minimum element in unsorted array

for (j = i+1; j < n; j++)

if (arr[j] < arr[small])

small = j;

// Swap the minimum element with the first element

int temp = arr[small]; arr[small] = arr[i];

arr[i] = temp;

}

}

void printArr(int arr[], int n) /\* function to print the array \*/

{ int i;

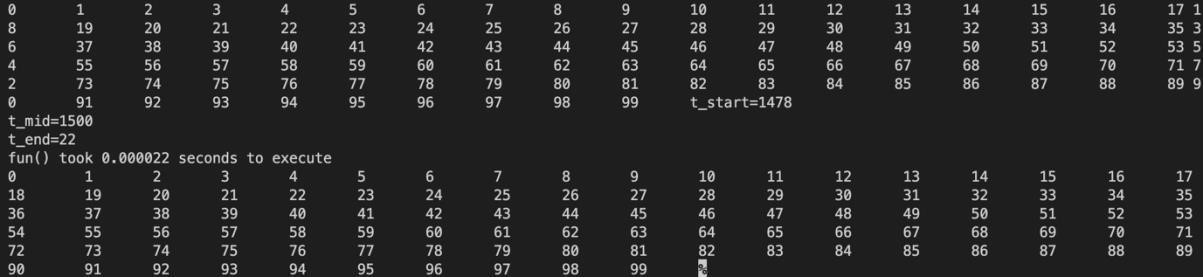
for (i = 0; i < n; i++)

printf("%d\t ", arr[i]);

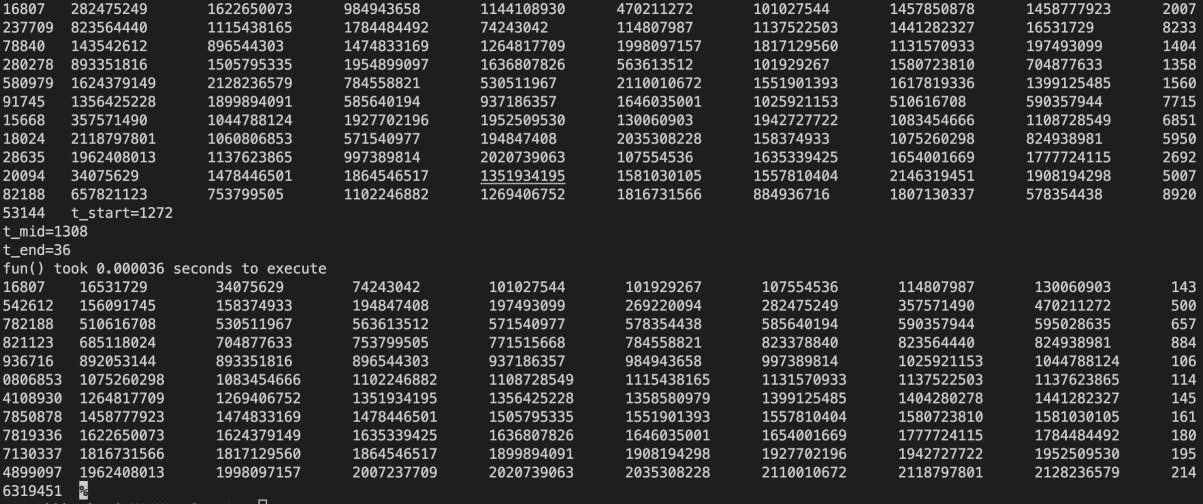
}

OUTPUT:

Sorted data



Random data



2. #include<stdio.h>

#include<time.h>

void bubble(int arr[], int n);

int main(){

int n=100; int arr[n];

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

arr[i]=rand();

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

printf("%d\n",arr[i]);

}

clock\_t t\_start,t\_end,t\_mid; t\_start = clock(); //starting time after initializing data

printf("t\_start=%lu\n",t\_start);

bubble(arr, n);

t\_mid=clock(); //mid time after function call printf("t\_mid=%lu\n",t\_mid);

t\_end = t\_mid - t\_start; //ending time after mid - start

printf("t\_end=%lu\n",t\_end);

double time\_taken = ((double)t\_end)/CLOCKS\_PER\_SEC; //clock per sec is a macro

and 1cps= 1 million micro sec

printf("fun() took %f seconds to execute \n", time\_taken);

return 0;

}

void bubble(int arr[], int n) // function to implement bubble sort

{

int i, j, temp;

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

{

for(j = i+1; j < n; j++)

{

if(arr[j] < arr[i])

{ temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

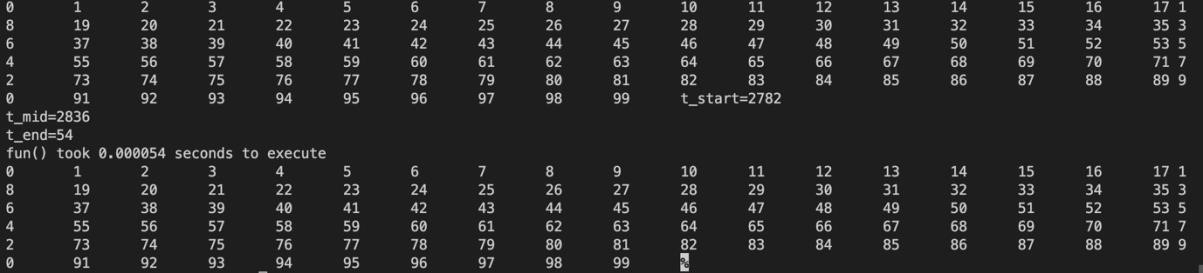
}

}

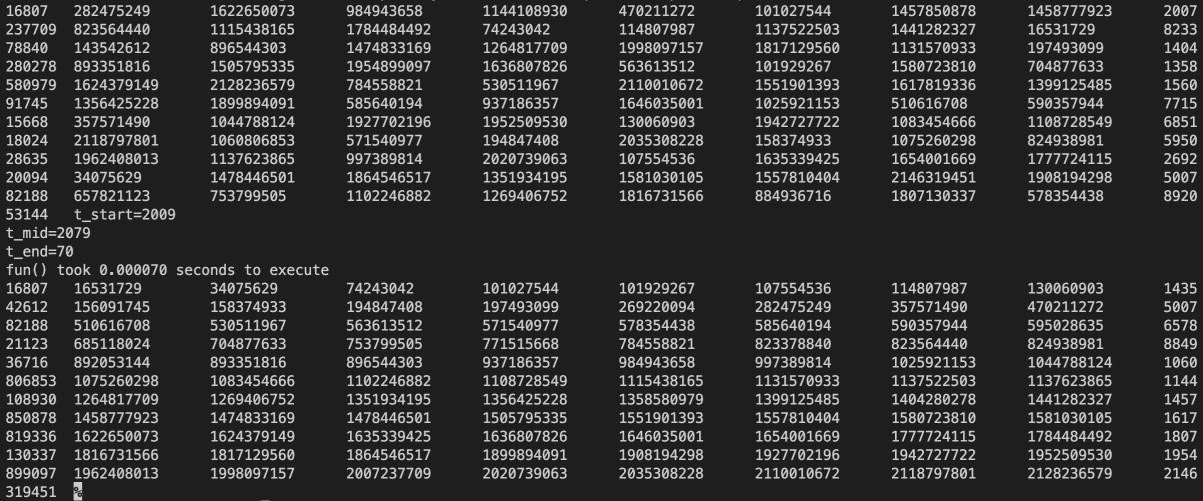
}

OUTPUT:

Sorted data



Random data



3. #include<stdio.h>

#include<time.h>

#include <stdlib.h>

void insertion(int arr[], int n);

void printArr(int arr[], int n);

int main(){ int n=100;

int arr[n];

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

arr[i]=rand();

}

for(int i=0;i<n;i++){ printf("%d\t",arr[i]);

}

clock\_t t\_start,t\_end,t\_mid; t\_start = clock(); //starting time after initializing data

printf("t\_start=%lu\n",t\_start);

insertion(arr, n);

t\_mid=clock(); //mid time after function call printf("t\_mid=%lu\n",t\_mid);

t\_end = t\_mid - t\_start; //ending time after mid - start

printf("t\_end=%lu\n",t\_end);

double time\_taken = ((double)t\_end)/CLOCKS\_PER\_SEC; //clock per sec is a macro

and 1cps= 1 million micro sec

printf("fun() took %f seconds to execute \n", time\_taken);

printArr(arr, n);

return 0;

}

void insertion(int arr[], int n) /\* function to sort an aay with insertion sort \*/

{

int i, j, temp; for (i = 1; i < n; i++) { temp = arr[i]; j = i

- 1;

while(j>=0 && temp <= arr[j]) /\* Move the elements greater than temp to one position

ahead from their current position\*/

{

arr[j+1] = arr[j];

j = j-1;

}

arr[j+1] = temp;

}

}

void printArr(int arr[], int n) /\* function to print the array \*/

{ int i;

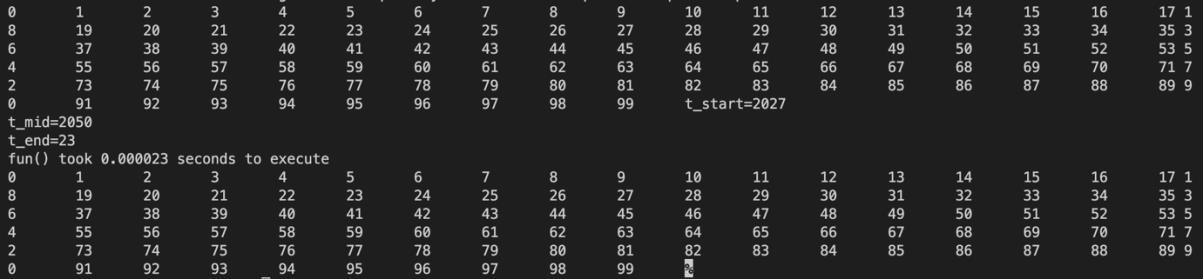
for (i = 0; i < n; i++)

printf("%d\t", arr[i]);

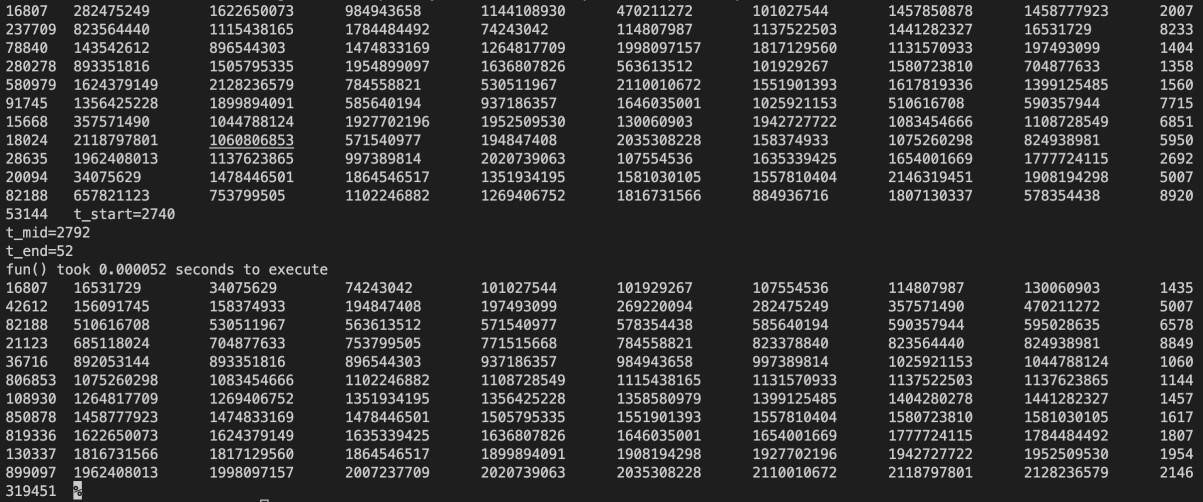
}

OUTPUT:

Sorted data



Random data



4.

#include<stdio.h>

#include<time.h>

#include <stdlib.h>

void merge(int arr[], int beg, int mid, int end) ;

void mergeSort(int arr[], int beg, int end); void

printArr(int arr[], int n);

int main(){ int n=100;

int arr[n],i;

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

arr[i]=rand();

}

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

printf(" %d\t", arr[i]);

}

clock\_t t\_start,t\_end,t\_mid; t\_start = clock(); //starting time after initializing data

printf("t\_start=%lu\n",t\_start);

mergeSort(arr, 0, n-1);

t\_mid=clock(); //mid time after function call printf("t\_mid=%lu\n",t\_mid);

t\_end = t\_mid - t\_start; //ending time after mid - start

printf("t\_end=%lu\n",t\_end);

double time\_taken = ((double)t\_end)/CLOCKS\_PER\_SEC; //clock per sec is a macro

and 1cps= 1 million micro sec

printf("fun() took %f seconds to execute \n", time\_taken);

printArr(arr, n);

return 0;

}

void mergeSort(int a[], int beg, int end)

{

if (beg < end)

{

int mid = (beg + end) / 2;

mergeSort(a, beg, mid);

mergeSort(a, mid + 1, end);

merge(a, beg, mid, end);

}

}

void merge(int arr[], int beg, int mid, int end)

{

int i, j, k; int n1 =

mid - beg + 1; int n2 = end - mid;

int LeftArray[n1], RightArray[n2]; //temporary arrays

/\* copy data to temp arrays \*/ for (int i = 0; i < n1; i++) LeftArray[i] = arr[beg + i]; for

(int j = 0; j < n2; j++)

RightArray[j] = arr[mid + 1 + j];

i = 0, /\* initial index of first sub-array \*/ j = 0; /\* initial index of second sub-array \*/

k = beg; /\* initial index of merged sub-array \*/

while (i < n1 && j < n2)

{

if(LeftArray[i] <= RightArray[j])

{

arr[k] = LeftArray[i];

i++; } else

{

arr[k] = RightArray[j]; j++; } k++;

} while

(i<n1)

{

arr[k] = LeftArray[i]; i++;

k++;

}

while (j<n2)

{

arr[k] = RightArray[j]; j++;

k++;

}

}

void printArr(int arr[], int n) /\* function to print the array \*/

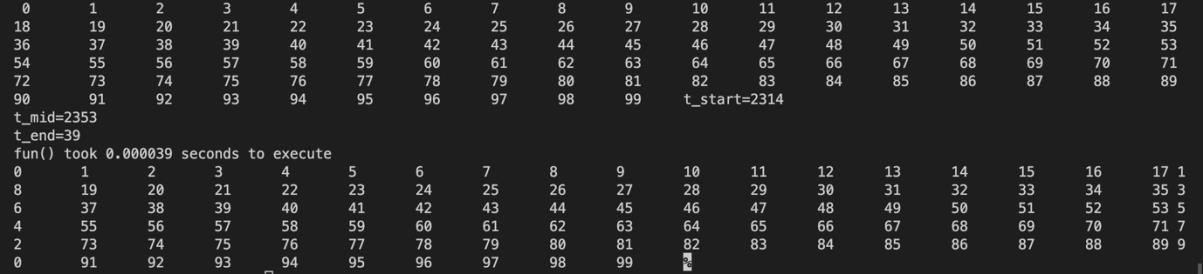
{ int i;

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

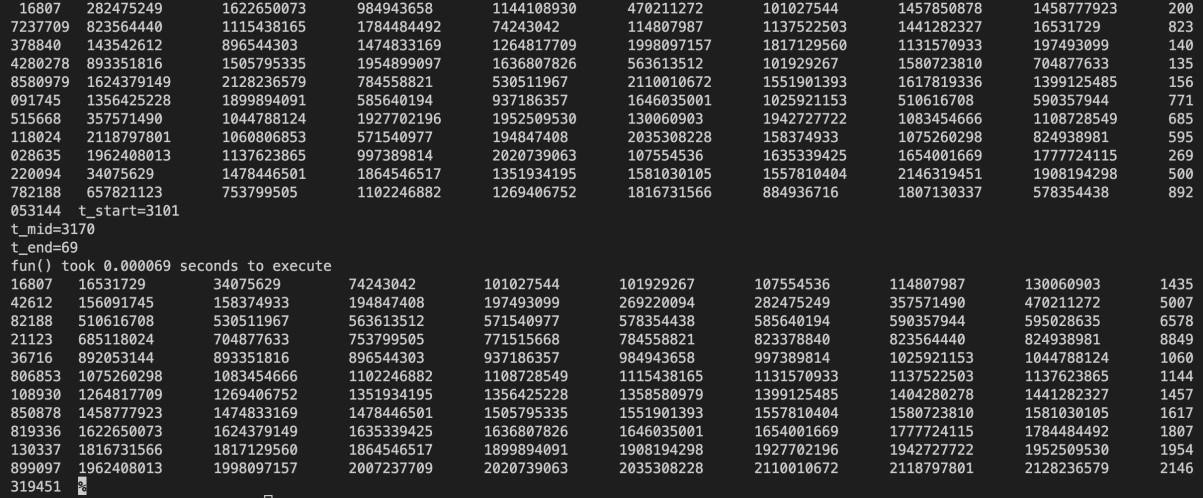
printf("%d\t", arr[i]);

}

Sorted data



Random data



5.

#include<stdio.h>

#include<time.h>

#include <stdlib.h>

void quick(int a[], int start, int end); void printArr(int arr[], int n);

int main(){

int n=100; int arr[n];

for(int i=0;i<n;i++){ arr[i]=i;

}

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

printf("%d\t",arr[i]);

}

clock\_t t\_start,t\_end,t\_mid; t\_start = clock(); //starting time after initializing data

printf("t\_start=%lu\n",t\_start);

quick(arr, 0, n - 1);

t\_mid=clock(); //mid time after function call printf("t\_mid=%lu\n",t\_mid);

t\_end = t\_mid - t\_start; //ending time after mid - start

printf("t\_end=%lu\n",t\_end);

double time\_taken = ((double)t\_end)/CLOCKS\_PER\_SEC; //clock per sec is a macro

and 1cps= 1 million micro sec

printf("fun() took %f seconds to execute \n", time\_taken);

printArr(arr, n); return 0;

}

int partition (int a[], int start, int end)

{

int pivot = a[end]; // pivot element

int i = (start - 1);

for (int j = start; j <= end - 1; j++)

{

// If current element is smaller than the pivot if (a[j] < pivot)

{

i++; // increment index of smaller element

int t = a[i]; a[i] = a[j]; a[j] = t; } } int t = a[i+1]; a[i+1] = a[end]; a[end] = t; return

(i + 1);

}

/\* function to implement quick sort \*/ void quick(int a[], int start, int end) /\* a[] = array to be sorted, start = Starting index, end = Ending index \*/

{

if (start < end)

{ int p = partition(a, start, end); //p is the partitioning index

quick(a, start, p - 1);

quick(a, p + 1, end);

}

}

void printArr(int arr[], int n) /\* function to print the array \*/

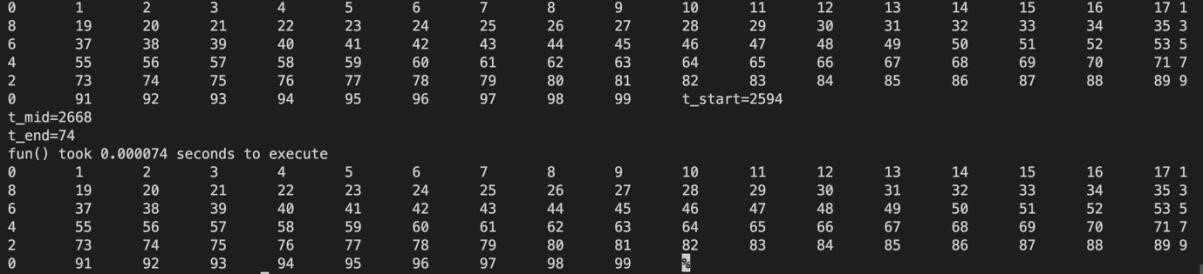
{ int i;

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

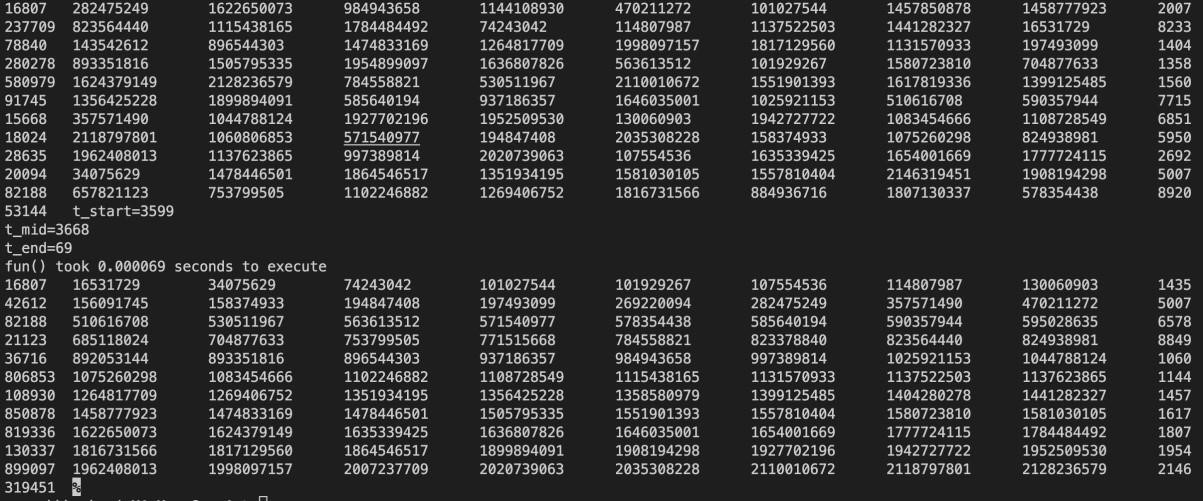
printf("%d\t", arr[i]);

}

Sorted data



Random data



GRAPH:

