CCC 204 Data Structures and Algorithms LABORATORY
REPORT :
LAB 10# - Sorting Algorithms

Samano, Rajel Johann
Information Technology Program, CABECS
Colegio San Agustin – Bacolod

I. INTRODUCTION

For Laboratory Activity Number Ten is to follow the objectives of the exercises and answer questions. The Three main objectives as follows:

- Identify and describe select sorting algorithms
- Modify and run sorting algorithms according to requirements
- Evaluate performance of sorting algorithms

II. IMPLEMENTATION / APPROACH

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

void swap(int* xp, int* yp){
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void bubbleSort(int arr[], int n){
    int i, j;
    bool swapped;
    for (i = 0; i < n - 1; i++) {
        swapped = false;
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(&arr[j], &arr[j + 1]);
                swapped = true;
            }
        }
        if (swapped == false)
            break;
    }
}

void printArray(int arr[], int size){
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
}

int main(int argc, char* argv[]){
    int n = argc - 1;
    int arr[n];

    for (int i = 1; i <= n; ++i) {
        arr[i - 1] = atoi(argv[i]);
    }

    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Figure.1 Bubblesort.c

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

void swap(int *xp, int *yp){
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n){
    int i, j, min_idx;
    for (i = 0; i < n-1; i++){
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;
        if(min_idx != i)
            swap(&arr[min_idx], &arr[i]);
    }
}

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

int main(int argc, char* argv[]){
    int n = argc - 1;
    int arr[n];
    for (int i = 1; i <= n; ++i) {
        arr[i - 1] = atoi(argv[i]);
    }
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Figure.2 Selectionsort.c

```c
#include <math.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

void insertionSort(int arr[], int n){
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

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

int main(int argc, char* argv[]){
    int n = argc - 1;
    int arr[n];
    for (int i = 1; i <= n; ++i) {
        arr[i - 1] = atoi(argv[i]);
    }
    insertionSort(arr, n);
    printArray(arr, n);
    return 0;
}
```

Figure.3 Insertionsort.c

```c
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>

void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
void printArray(int A[], int size) {
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

int main(int argc, char* argv[]) {
    int n = argc - 1;
    int arr[n];
    for (int i = 1; i <= n; ++i) {
        arr[i - 1] = atoi(argv[i]);
    }
    mergeSort(arr, 0, n);
    printf("\nSorted array is \n");
    printArray(arr, n);
    return 0;
}
```

Figure.4 Mergesort.c

```c
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>

void quicksortMiddle(int arr[], int low, int high) {
    if (low < high) {
        int pivot = arr[(low + high) / 2];
        int i = low;
        int j = high;
        int temp;
        while (i <= j) {
            while (arr[i] < pivot) i++;
            while (arr[j] > pivot) j--;

            if (i <= j) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
                i++;
                j--;
            }
        }
        if (low < j) quicksortMiddle(arr, low, j);
        if (i < high) quicksortMiddle(arr, i, high);
    }
}

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

int main(int argc, char* argv[]){
    int n = argc - 1;
    int arr[n];
    for (int i = 1; i <= n; ++i) {
        arr[i - 1] = atoi(argv[i]);
    }
    quicksortMiddle(arr, 0, n - 1);
    printf("Sorted with Middle Element as Pivot: \n");
    printArray(array1, n);
    return 0;
}
```

Figure.5 Quicksort.c

My approach to the problem was search google for each sorting methods and modify them so that they could be support using command line arguments as it is stated by our teacher that is would be fasster.

III. EXPERIMENTAL FINDINGS / DISCUSSIONS

Algorithm	5 Values (3 trials and average in seconds)				10 Values (3 trials and average in seconds)				20 Values (3 trials and average in seconds)			
Bubble	0.67	0.74	0.72	1.65	0.65	0.76	0.74	1.64	0.65	0.78	0.67	1.65
Selection	0.70	0.73	1.09	1.88	0.61	1.01	0.66	1.84	0.71	0.79	0.65	1.72
Insertion	0.74	0.85	0.88	1.88	0.80	0.70	0.77	1.76	0.75	0.82	0.70	1.80
Merge	0.71	0.82	0.68	1.76	0.76	0.75	0.73	1.75	0.66	0.67	0.70	1.56
Quicksort	0.85	0.81	0.73	1.90	2.00	0.71	0.75	2.96	0.79	0.62	0.82	1.68

IV.CONCLUSIONS

To conclude this laboratory activity, I believe the best and most time efficient sorting method would be the bubble sort as average for all three are in the ranges of one point 65 secs compared to other the second one would be merge method as it is in the ranges of one point seven. Third has to be insertion but I can't really base which is the fastest as when running the code for each sorting method a strong rain was occurring in my area it might have affected some of the execution time.

V. References:

https://hackr.io/blog/quick-sort-in-c

https://www.geeksforgeeks.org/c-program-for-merge-sort/

https://www.geeksforgeeks.org/insertion-sort/

https://www.geeksforgeeks.org/selection-sort/

https://www.geeksforgeeks.org/bubble-sort/