DSA LAB

Assignment 10

Name: Madhav Jha
Roll no.: E3-48
Branch: CSE (AI & ML)

Sorting Algorithm

Write a generalized function that takes a parameter to indicate the mode (say 1 for decreasing order, 2 for increasing order, 3 for increasing order with the Nth element out of order, 4 for a randomly generated element values), to create a list of elements. The parameter indicating the number of elements (the maximum size is large enough to run possible iterations to test the time complexity, say 1000000) should be a multiple of 10. Also write appropriate functions to create a copy of the list and to display the list contents. Using above functions, write a menu-driven C program to order the list in ascending sequence using - Insertion Sort, Selection Sort, Shell Sort and Merge Sort and Quick sort Create a table which display time complexity of each sort in all the cases and also indicate which sort executes in minimum time as compare to other.

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

#define MAXNUM 1000000

struct timeval start, end;
long sec, ms;

// arrays to check for
// best -> increasing array
// average -> random array
// worst -> decreasing array
int incrArr[MAXNUM], decArr[MAXNUM], randArr[MAXNUM];

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

void makeIncrArr() {
    int start = rand();
    for (int i = 0;i < MAXNUM;i++) {
```

```
        incrArr[i] = start + i;
    }
}
void makeDecArr() {
    int start = rand() + MAXNUM;
    for (int i = MAXNUM - 1;i >= 0;i--) {
        decArr[i] = start - i;
    }
}
void makeRandArr() {
    for (int i = 0;i < MAXNUM;i++) {
        randArr[i] = rand();
    }
}

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

    // creating copy to sort
    int arr[MAXNUM];
    for (int i = 0;i < MAXNUM;i++) {
        arr[i] = inpArr[i];
    }

    for (int i = 0;i < n;i++) {
        for (int j = i;j < n;j++) {
            if (arr[i] < arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }

    // printf("\nBubble sort: ");
    // printArr(arr, n);
}

void insertionSort(int n, int inpArr[]) {

    // creating copy to sort
```

```
    int arr[MAXNUM];
    for (int i = 0;i < MAXNUM;i++) {
        arr[i] = inpArr[i];
    }

    int key = 0, j = 0;
    for (int i = 1;i < MAXNUM;i++) {
        j = i - 1;
        key = arr[i];

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }

    // printf("\nInsertion sort: ");
    // printArr(arr, n);
}

void selectionSort(int n, int inpArr[]) {
    // creating copy to sort
    int arr[MAXNUM];
    for (int i = 0;i < MAXNUM;i++) {
        arr[i] = inpArr[i];
    }

    int k = 0;
    for (int i = 0;i < n - 1;i++) {
        k = i;
        for (int j = i;j < n;j++) {
            k = (arr[j] < arr[k]) ? j : k;
        }
        //swap
        int temp = arr[k];
        arr[k] = arr[i];
        arr[i] = temp;
    }
```

```
    // printf("\nSelection sort: ");
    // printArr(arr, n);
}

void shellSort(int n, int inpArr[]) {
    // creating copy to sort
    int arr[MAXNUM];
    for (int i = 0;i < MAXNUM;i++) {
        arr[i] = inpArr[i];
    }

    for (int gap = n / 2;gap > 0;gap /= 2) {

        for (int i = gap;i < n;i++) {
            int temp = arr[i], j;
            for (j = i;j >= gap && arr[j - gap] > temp;j -= gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }

    // printf("\nShell sort: ");
    // printArr(arr, n);
}

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

    int L[n1], R[n2];

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

```
    int 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 mergeSortRec(int arr[], int l, int r)
{
    if (l < r) {
        int m = l + (r - l) / 2;

        mergeSortRec(arr, l, m);
        mergeSortRec(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

void mergeSort(int n, int inpArr[]) {
    // creating copy to sort
```

```
    int arr[MAXNUM];
    for (int i = 0;i < MAXNUM;i++) {
        arr[i] = inpArr[i];
    }
    mergeSortRec(arr, 0, n - 1);

    // printf("\nMerge sort: ");
    // printArr(arr, n);
}

int partition(int arr[], int l, int h)
{
    int pivot = arr[h];
    int i = (l - 1);

    for (int j = l; j <= h - 1; j++)
    {
        // If current element is smaller than the pivot
        if (arr[j] < pivot)
        {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[h];
    arr[h] = temp;
    return (i + 1);
}

void quickSortRec(int arr[], int l, int h)
{
    if (l < h)
    {
        int p = partition(arr, l, h);

        quickSortRec(arr, l, p - 1);
        quickSortRec(arr, p + 1, h);
```

```
    }
}

void quickSort(int n, int inpArr[]) {
    // creating copy to sort
    int arr[MAXNUM];
    for (int i = 0;i < MAXNUM;i++) {
        arr[i] = inpArr[i];
    }
    quickSortRec(arr, 0, n - 1);

    printf("\nQuick Sort: ");
    printArr(arr, n);
}

int main() {
    srand(time(0));

    printf("\n\n::::: Sorting Algorithms :::::\n\n");

    // generating arrays
    makeIncrArr();
    // printArr(incrArr, MAXNUM);
    makeDecArr();
    // printArr(decArr, MAXNUM);
    makeRandArr();
    // printArr(randArr, MAXNUM);

    printf("  Sorting algo  |\tBest Case\tWorst Case\tAverage
Case\n");
    printf("----------------|----------------------------------------
--------\n");

    {
        // bubble sort
        printf("Bubble Sort     |\t");

        // Best case
        gettimeofday(&start, NULL); // start time
        bubbleSort(MAXNUM, incrArr);
```

```
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\t", sec, ms);

        // Worst case
        gettimeofday(&start, NULL); // start time
        bubbleSort(MAXNUM, decArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\t", sec, ms);

        // average case
        gettimeofday(&start, NULL); // start time
        bubbleSort(MAXNUM, randArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\n", sec, ms);
    }

    {
        // Insertion sort
        printf("Insertion Sort  |\t");

        // Best case
        gettimeofday(&start, NULL); // start time
        insertionSort(MAXNUM, incrArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\t", sec, ms);

        // Worst case
        gettimeofday(&start, NULL); // start time
```

```
        insertionSort(MAXNUM, decArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\t", sec, ms);

        // average case
        gettimeofday(&start, NULL); // start time
        insertionSort(MAXNUM, randArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\n", sec, ms);
    }

    {
        // Selection sort
        printf("Selection Sort  |\t");

        // Best case
        gettimeofday(&start, NULL); // start time
        selectionSort(MAXNUM, incrArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\t", sec, ms);

        // Worst case
        gettimeofday(&start, NULL); // start time
        selectionSort(MAXNUM, decArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\t", sec, ms);

        // average case
```

```
        gettimeofday(&start, NULL); // start time
        selectionSort(MAXNUM, randArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\n", sec, ms);
    }

    {
        // Shell sort
        printf("Shell Sort      |\t");

        // Best case
        gettimeofday(&start, NULL); // start time
        shellSort(MAXNUM, incrArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\t", sec, ms);

        // Worst case
        gettimeofday(&start, NULL); // start time
        shellSort(MAXNUM, decArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\t", sec, ms);

        // average case
        gettimeofday(&start, NULL); // start time
        shellSort(MAXNUM, randArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\n", sec, ms);
    }
```

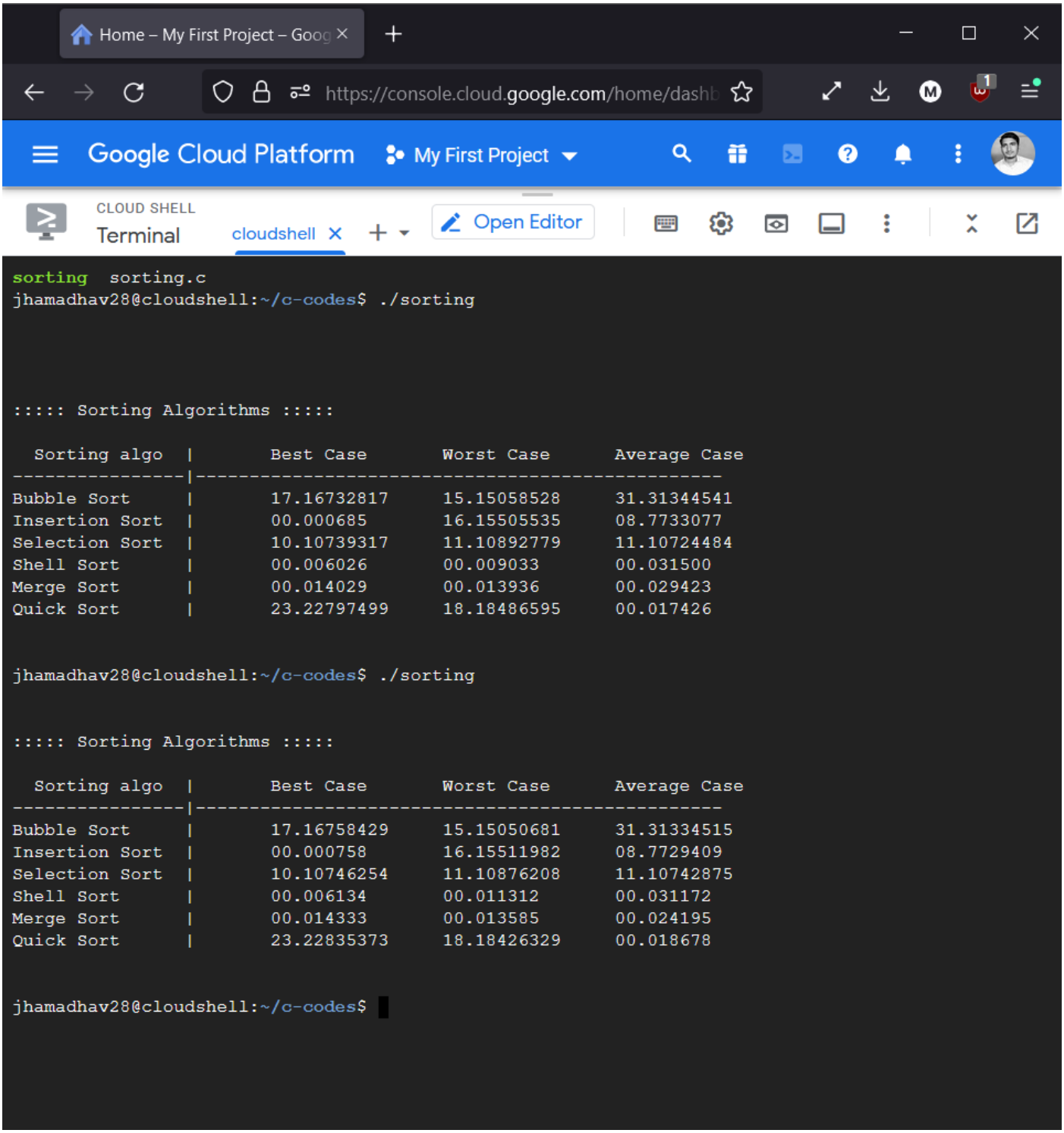
```
{
    // Merge sort
    printf("Merge Sort      |\t");

    // Best case
    gettimeofday(&start, NULL); // start time
    mergeSort(MAXNUM, incrArr);
    gettimeofday(&end, NULL); // end time

    sec = (end.tv_sec - start.tv_sec);
    ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
    printf("%.2lu.%.6lu\t", sec, ms);

    // Worst case
    gettimeofday(&start, NULL); // start time
    mergeSort(MAXNUM, decArr);
    gettimeofday(&end, NULL); // end time

    sec = (end.tv_sec - start.tv_sec);
    ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
    printf("%.2lu.%.6lu\t", sec, ms);

    // average case
    gettimeofday(&start, NULL); // start time
    mergeSort(MAXNUM, randArr);
    gettimeofday(&end, NULL); // end time

    sec = (end.tv_sec - start.tv_sec);
    ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
    printf("%.2lu.%.6lu\n", sec, ms);
}

{
    // Quick sort
    printf("Quick Sort      |\t");

    // Best case
    gettimeofday(&start, NULL); // start time
    quickSort(MAXNUM, incrArr);
```

```
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\t", sec, ms);

        // Worst case
        gettimeofday(&start, NULL); // start time
        quickSort(MAXNUM, decArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\t", sec, ms);

        // average case
        gettimeofday(&start, NULL); // start time
        quickSort(MAXNUM, randArr);
        gettimeofday(&end, NULL); // end time

        sec = (end.tv_sec - start.tv_sec);
        ms = ((sec * 1000000) + end.tv_usec) - (start.tv_usec);
        printf("%.2lu.%.6lu\n", sec, ms);
    }

    printf("\n\n");
    return 0;
}
```

Expected time complexity:

Sorting Algorithm	Best Case	Worst Case	Average Case
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Shell sort	$O(n\log(n))$	$O(n(\log(n))^2)$	$O(n(\log(n))^2)$
Merge sort	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$
Quick sort	$O(n\log(n))$	$O(n^2)$	$O(n\log(n))$

Output:


Home – My First Project – Goog
https://console.cloud.google.com/home/dashb
Google Cloud Platform
My First Project
CLOUD SHELL
Terminal
cloudshell
Open Editor
sorting sorting.c
jhamadhav28@cloudshell:~/c-codes$ ./sorting
::::: Sorting Algorithms :::::
Sorting algo | Best Case Worst Case Average Case
Bubble Sort | 17.16732817 15.15058528 31.31344541
Insertion Sort | 00.000685 16.15505535 08.7733077
Selection Sort | 10.10739317 11.10892779 11.10724484
Shell Sort | 00.006026 00.009033 00.031500
Merge Sort | 00.014029 00.013936 00.029423
Quick Sort | 23.22797499 18.18486595 00.017426
jhamadhav28@cloudshell:~/c-codes$ ./sorting
::::: Sorting Algorithms :::::
Sorting algo | Best Case Worst Case Average Case
Bubble Sort | 17.16758429 15.15050681 31.31334515
Insertion Sort | 00.000758 16.15511982 08.7729409
Selection Sort | 10.10746254 11.10876208 11.10742875
Shell Sort | 00.006134 00.011312 00.031172
Merge Sort | 00.014333 00.013585 00.024195
Quick Sort | 23.22835373 18.18426329 00.018678
jhamadhav28@cloudshell:~/c-codes$