Multi-threaded sorting application

Team members:

UTKARSH TAWAKLEY - 200905071
LIKHITH MALLAVARAPU - 200905079
PRATHIK JAYARAJA SHETTY - 200905045
CALVIN JOHN MACHADO - 200905013

Department of Computer Science, Manipal Institute of Technology, Manipal, Karnataka, India

Abstract

The project is a sorting application that incorporates selection sort, bubble sort, insertion sort and merge sort whilst also displaying the array at each iteration of the sort. The selection sort, bubble sort and insertion sort is implemented using 2 threads; one handling the sorting in ascending order and the other handling the sorting in descending order. To achieve thread synchronisation, we use mutex locks to lock the functioning of one thread until the other is done and unlocks. For the merge sort, threads handle the sorting and merging of each individual sub array. The user is allowed to select a particular sort with the help of a menu.

Problem statement and description

Build a multi-threaded sorting application (menu driven program) that allows the user to select from four different sorts. Each sort is implemented using threads; displaying the array at each iteration of the sort.

Topics it is related to

1) Multi-threading
2) Mutex locks
3) Thread synchronisation

Algorithm

//To implement selection sort

Step 1: Initialize minimum value(min_idx) to location 0.

Step 2: Traverse the array to find the minimum element in the array.

Step 3: While traversing if any element smaller than min_idx is found then swap both the values.

Step 4: Then, increment min_idx to point to the next element.

Step 5: Repeat until the array is sorted.

//To implement bubble sort

Step 1: Run a nested for loop to traverse the input array using two variables i and j, such that $0 \leq i < n-1$ and $0 \leq j < n-i-1$.

Step 2: If arr[j] is greater than arr[j+1] then swap these adjacent elements, else move on.

//To implement insertion sort

Step 1: Iterate from arr[1] to arr[N] over the array.

Step 2: Compare the current element (key) to its predecessor.

Step 3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

//To implement merge sort

Step 1: Check if there is one element in the list then return the element. Else, go to step 2.

Step 2: Divide the data using threads into two halves until it can’t be divided further.

Step 3: Finally, merge the smaller lists into new lists in sorted order.

//To implement main code

Step 1: Display the menu and prompt user to pick an option.

Step 2: Once the user selects desired option, the appropriate functions are called using threads to perform sorts in ascending and descending order.
Step 3: Lock one thread until the other has finished executing, displaying the array at each iteration.

Methodology

The code is built entirely in C language. Threads are used to sort an array in ascending and descending order which speeds up the process of sorting. But since each iteration has to be displayed, thread synchronization was required which was achieved by using mutex locks. Posix threads are being used for thread implementation.

Implementation

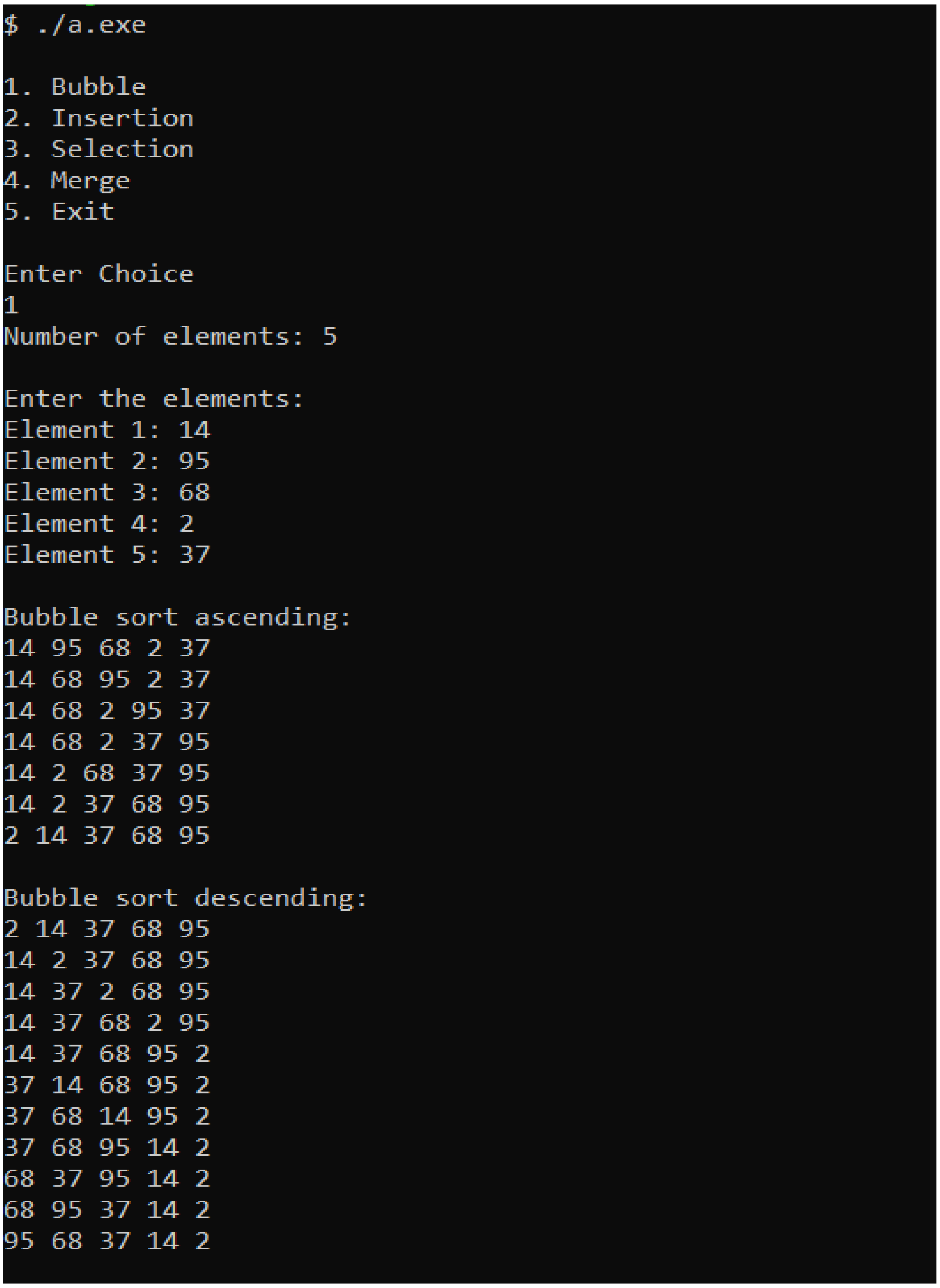
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

pthread_mutex_t mutex;

// structure for arguments
typedef struct
{
    int *arr; // array
    int n;    // size of array
} arguments;

int array_ptr[10];
int b[10];
int c[5], d[5];
int low = 0, high = 9, high1 = 4;
void *run(void *array_ptr);
void *run1(void *array_ptr);

void merge1(int low, int mid, int high1)
{
    int l1, l2, i;

    for (l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high1; i++)
    {
        if (c[l1] <= c[l2])
            d[i] = c[l1++];
        else
            d[i] = c[l2++];
    }

    while (l1 <= mid)
        d[i++] = c[l1++];

    while (l2 <= high1)
        d[i++] = c[l2++];

    for (i = low; i <= high1; i++)
    {
        c[i] = d[i];
    }
}

void mergesort1(int low, int high1)
{
    int mid;
    int *elements = (int *)c;

    if (low < high1)
    {
        mid = (low + high1) / 2;
        mergesort1(low, mid);
        mergesort1(mid + 1, high1);
        merge1(low, mid, high1);
    }
    else
    {
        return;
    }
}
int c1[5], d1[5];

void *merging(void *array_ptr)
{
    int i = 0;

    int j = 0;
    int k = 0;
    int m = 5, n = 5;

    while (i < m && j < n)

    {

        if (c1[i] < d1[j])

        {

            b[k] = c1[i];

            i++;
        }
        else

        {

            b[k] = d1[j];

            j++;
        }

        k++;
    }

    if (i >= m)

    {

        while (j < n)

        {

            b[k] = d1[j];

            j++;

            k++;
        }
    }
    if (j >= n)

    {

        while (i < m)

        {

            b[k] = c1[i];

            i++;

            k++;
        }
    }
    printf("\n After merging: \n");
```

```
    for (i = 0; i < m + n; i++)
    {
        printf("%d\t", b[i]);
    }
            printf("\n");
}

void *run(void *c)
{
    mergesort1(low, high1);
}

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

void *bubble_asc(void *arg)
{
    arguments *array = (arguments *)arg;
    pthread_mutex_lock(&mutex);
    printf("\nBubble sort ascending:\n");
    printArray(array);
    sleep(1);
    // loop to access each array element
    for (int step = 0; step < array->n - 1; ++step)
    {

        // loop to compare array elements
        for (int i = 0; i < array->n - step - 1; ++i)
        {

            // compare two adjacent elements
            // change > to < to sort in descending order
            if (array->arr[i] > array->arr[i + 1])
            {

                // swapping occurs if elements
                // are not in the intended order
                int temp = array->arr[i];
                array->arr[i] = array->arr[i + 1];
                array->arr[i + 1] = temp;
                sleep(1);
                printArray(array);
            }
        }
    }
    pthread_mutex_unlock(&mutex);
}

void *bubble_dsc(void *arg)
{

    arguments *array = (arguments *)arg;
    pthread_mutex_lock(&mutex);
    printf("\nBubble sort descending:\n");
    printArray(array);
    sleep(1);
    // loop to access each array element
    for (int step = 0; step < array->n - 1; ++step)
    {

        // loop to compare array elements
        for (int i = 0; i < array->n - step - 1; ++i)
        {

            // compare two adjacent elements
            // change > to < to sort in descending order
            if (array->arr[i] < array->arr[i + 1])
            {

                // swapping occurs if elements
                // are not in the intended order
                int temp = array->arr[i];
                array->arr[i] = array->arr[i + 1];
                array->arr[i + 1] = temp;
                sleep(1);
                printArray(array);
            }
        }
    }
    pthread_mutex_unlock(&mutex);
}
```

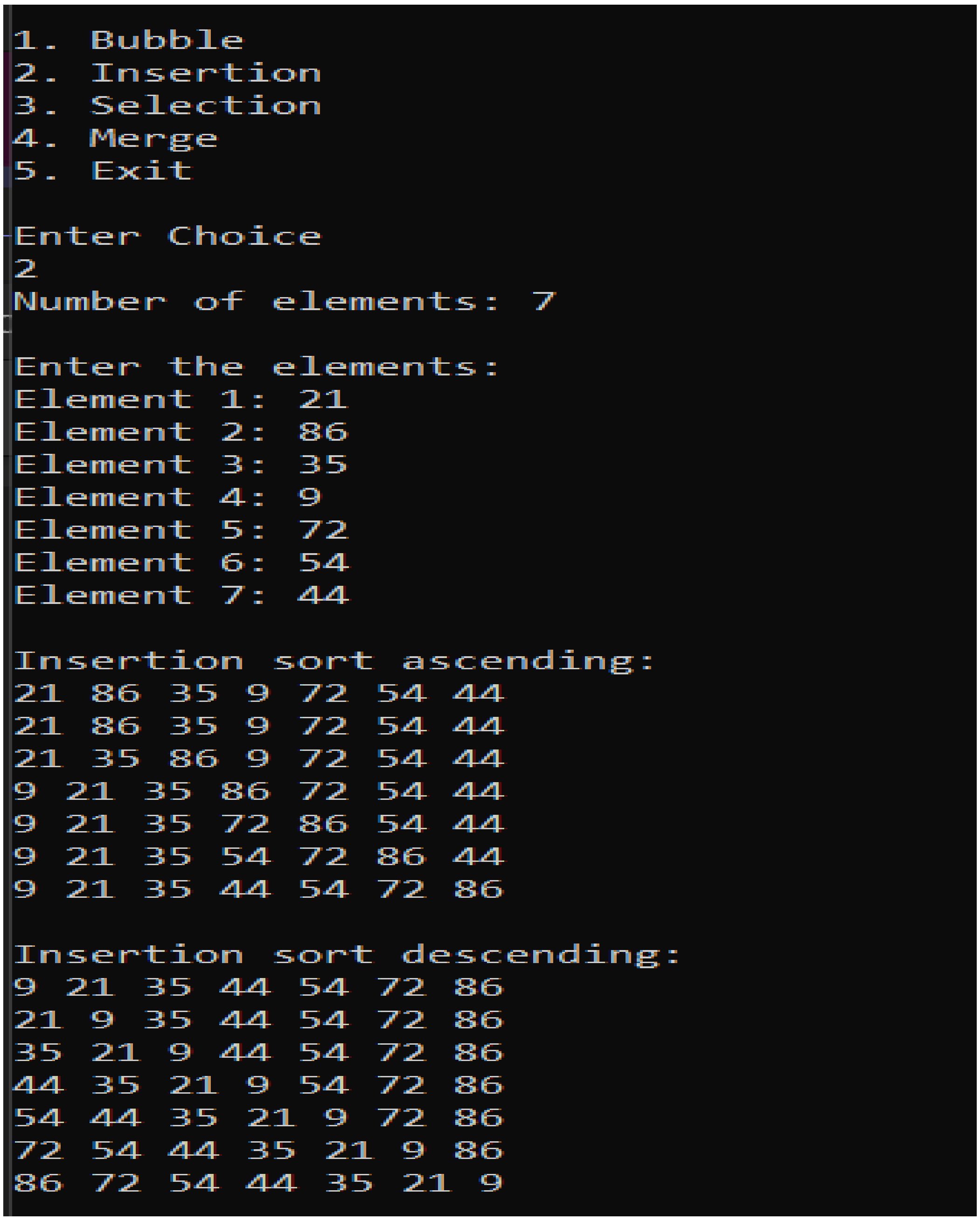
```
// func to swap 2 variables
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// sorts array in ascending using insertion sort
void *insertion_asc(void *arg)
{

    pthread_mutex_lock(&mutex);
    printf("\nInsertion sort ascending:\n");
    arguments *a = (arguments *)arg;
    sleep(1);
    printArray(a);
    int i, key, j;
    for (i = 1; i < (a->n); i++)
    {
        key = a->arr[i];
        j = i - 1;
        /* Move elements of arr[0..i-1], that are
       greater than key, to one position ahead
       of their current position */
        while (j >= 0 && a->arr[j] > key)
        {
            a->arr[j + 1] = a->arr[j];
            j = j - 1;
        }
        a->arr[j + 1] = key;
        sleep(1);
        printArray(a);
    }
    pthread_mutex_unlock(&mutex);
}

// sorts array in descending using insertion sort
void *insertion_dsc(void *arg)
{

    pthread_mutex_lock(&mutex);
    printf("\nInsertion sort descending:\n");
    arguments *a = (arguments *)arg;
    sleep(1);
    printArray(a);

    int i, key, j;
    for (i = 1; i < (a->n); i++)
    {
        key = a->arr[i];
        j = i - 1;
        /* Move elements of arr[0..i-1], that are
       greater than key, to one position ahead
       of their current position */
        while (j >= 0 && a->arr[j] < key)
        {
            a->arr[j + 1] = a->arr[j];
            j = j - 1;
        }
        a->arr[j + 1] = key;
        sleep(1);
        printArray(a);
    }

    pthread_mutex_unlock(&mutex);
}

// sorts array in ascending using selection sort
void *sel_asc(void *arg)
{

    pthread_mutex_lock(&mutex);
    printf("\nSelection sort ascending:\n");
    arguments *a = (arguments *)arg;
    sleep(1);
    printArray(a);

    for (int step = 0; step < (a->n) - 1; step++)
    {
        int min = step;
        for (int i = step + 1; i < a->n; i++)
        {
            if (a->arr[i] < a->arr[min])
                min = i;
        }

        swap(&a->arr[min], &a->arr[step]);
        sleep(1);
        printArray(a);
    }

    pthread_mutex_unlock(&mutex);
}

// sorts array in descending using selection sort
void *sel_dsc(void *arg)
{

    pthread_mutex_lock(&mutex);
    printf("\nSelection sort descending:\n");
    arguments *a = (arguments *)arg;
    sleep(1);
    printArray(a);

    for (int step = 0; step < (a->n) - 1; step++)
    {
        int min = step;
        for (int i = step + 1; i < a->n; i++)
        {
            if (a->arr[i] > a->arr[min])
                min = i;
        }

        swap(&a->arr[min], &a->arr[step]);
        sleep(1);
        printArray(a);
    }

    pthread_mutex_unlock(&mutex);
}

int main()
{
    int ch;
    while (1)
    {
        printf("\n1. Bubble\n2. Insertion\n3. Selection\n4. Merge\n5. Exit\n\nEnter Choice\n");
        scanf("%d", &ch);
        if (ch == 5)
        {
            printf("Exiting\n");
            break;
        }
        if (ch == 1)
        {
            arguments arg;
            pthread_t thread[2];
            pthread_mutex_init(&mutex, 0);

            printf("Number of elements: ");
            scanf("%d", &arg.n);

            arg.arr = (int *)malloc(arg.n * sizeof(int));

            printf("\nEnter the elements: \n");

            for (int i = 0; i < arg.n; i++)
            {
                printf("Element %d: ", i + 1);
                scanf("%d", &arg.arr[i]);
            }
            pthread_create(&thread[0], NULL, &bubble_asc, (void *)&arg);
            pthread_create(&thread[1], NULL, &bubble_dsc, (void *)&arg);
            pthread_join(thread[0], NULL);
            pthread_join(thread[1], NULL);
            pthread_mutex_destroy(&mutex);
        }
        else if (ch == 2)
        {
            arguments arg;
            pthread_t thread[2];
            pthread_mutex_init(&mutex, 0);

            printf("Number of elements: ");
            scanf("%d", &arg.n);

            arg.arr = (int *)malloc(arg.n * sizeof(int));

            printf("\nEnter the elements: \n");

            for (int i = 0; i < arg.n; i++)
            {
                printf("Element %d: ", i + 1);
                scanf("%d", &arg.arr[i]);
            }
            pthread_create(&thread[0], NULL, &insertion_asc, (void *)&arg);
            pthread_create(&thread[1], NULL, &insertion_dsc, (void *)&arg);
            pthread_join(thread[0], NULL);
            pthread_join(thread[1], NULL);
            pthread_mutex_destroy(&mutex);
        }
        else if (ch == 3)
        {
            arguments arg;
            pthread_t thread[2];
            pthread_mutex_init(&mutex, 0);

            printf("Number of elements: ");
            scanf("%d", &arg.n);

            arg.arr = (int *)malloc(arg.n * sizeof(int));

            printf("\nEnter the elements: \n");

            for (int i = 0; i < arg.n; i++)
            {
                printf("Element %d: ", i + 1);
                scanf("%d", &arg.arr[i]);
            }
            pthread_create(&thread[0], NULL, &sel_asc, (void *)&arg);
            pthread_create(&thread[1], NULL, &sel_dsc, (void *)&arg);
            pthread_join(thread[0], NULL);
            pthread_join(thread[1], NULL);
            pthread_mutex_destroy(&mutex);
        }
        else if (ch == 4)
        {
            int n = 10;
            printf(" 10 elements you want to enter\n");
            for (int i = 0; i < n; i++)
            {
                scanf("%d", &array_ptr[i]);
            }
            for (int i = 0; i < 5; i++)
            {
                c[i] = array_ptr[i];
            }

            printf("\n 2 sorted subarray are:\n ");
```

```
            pthread_t t1, t2, t3;
            pthread_create(&t1, NULL, run, (void *)c);
            pthread_join(t1, NULL);

            for (int i = 0; i < 5; i++)
            {
                printf("%d\t", c[i]);
                c1[i] = c[i];
            }
            printf("\n");

            for (int i = 0; i < 5; i++)
            {
                c[i] = array_ptr[i + 5];
            }
            pthread_create(&t2, NULL, run, (void *)c);
            pthread_join(t2, NULL);
            for (int i = 0; i < 5; i++)
            {
                printf("%d\t", c[i]);
                d1[i] = c[i];
            }

            pthread_create(&t3, NULL, merging, (void *)array_ptr);
            pthread_join(t3, NULL);
        }
        else
            printf("Invalid Choice\n");
    }
}
```

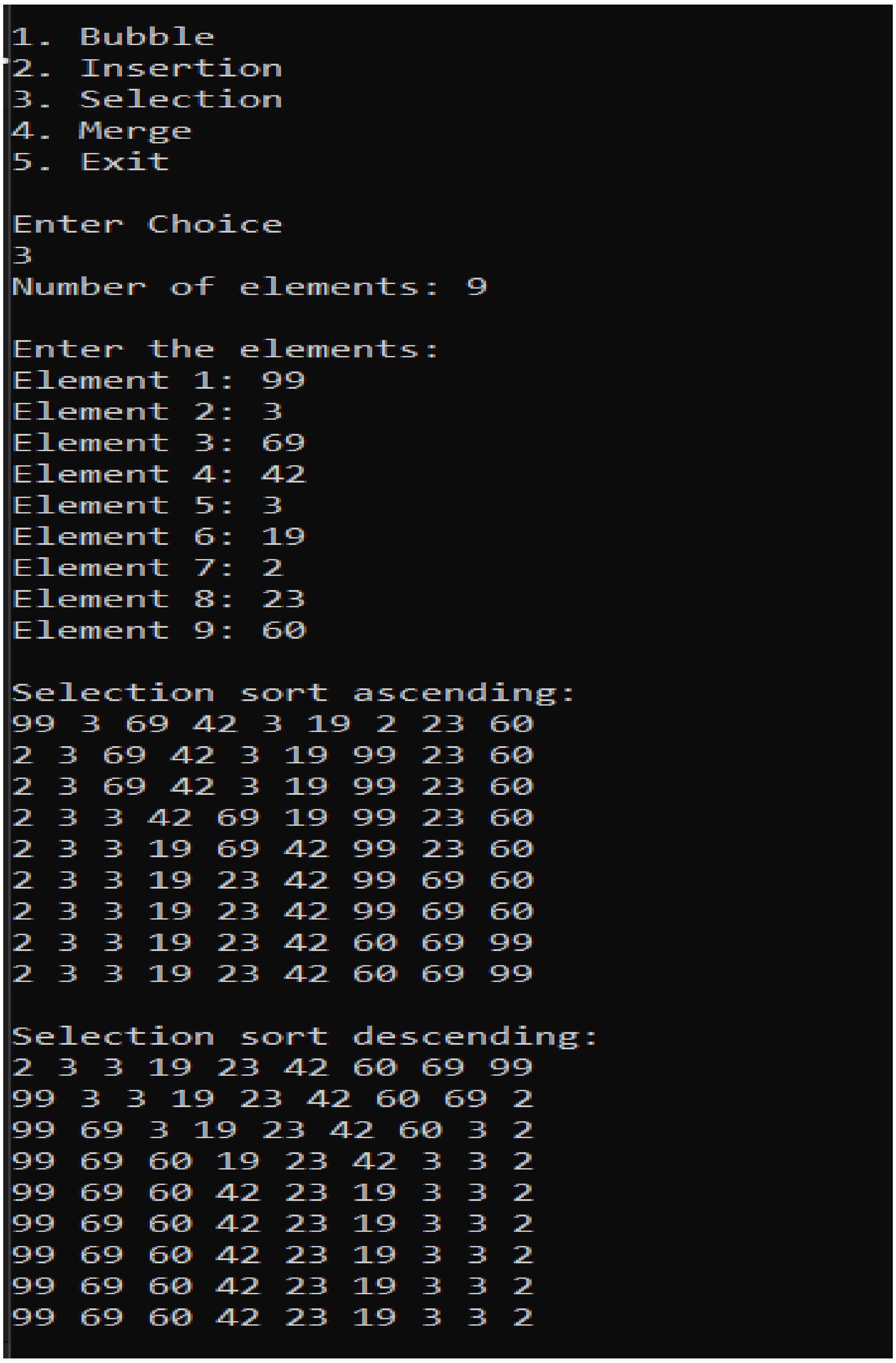
Output examples



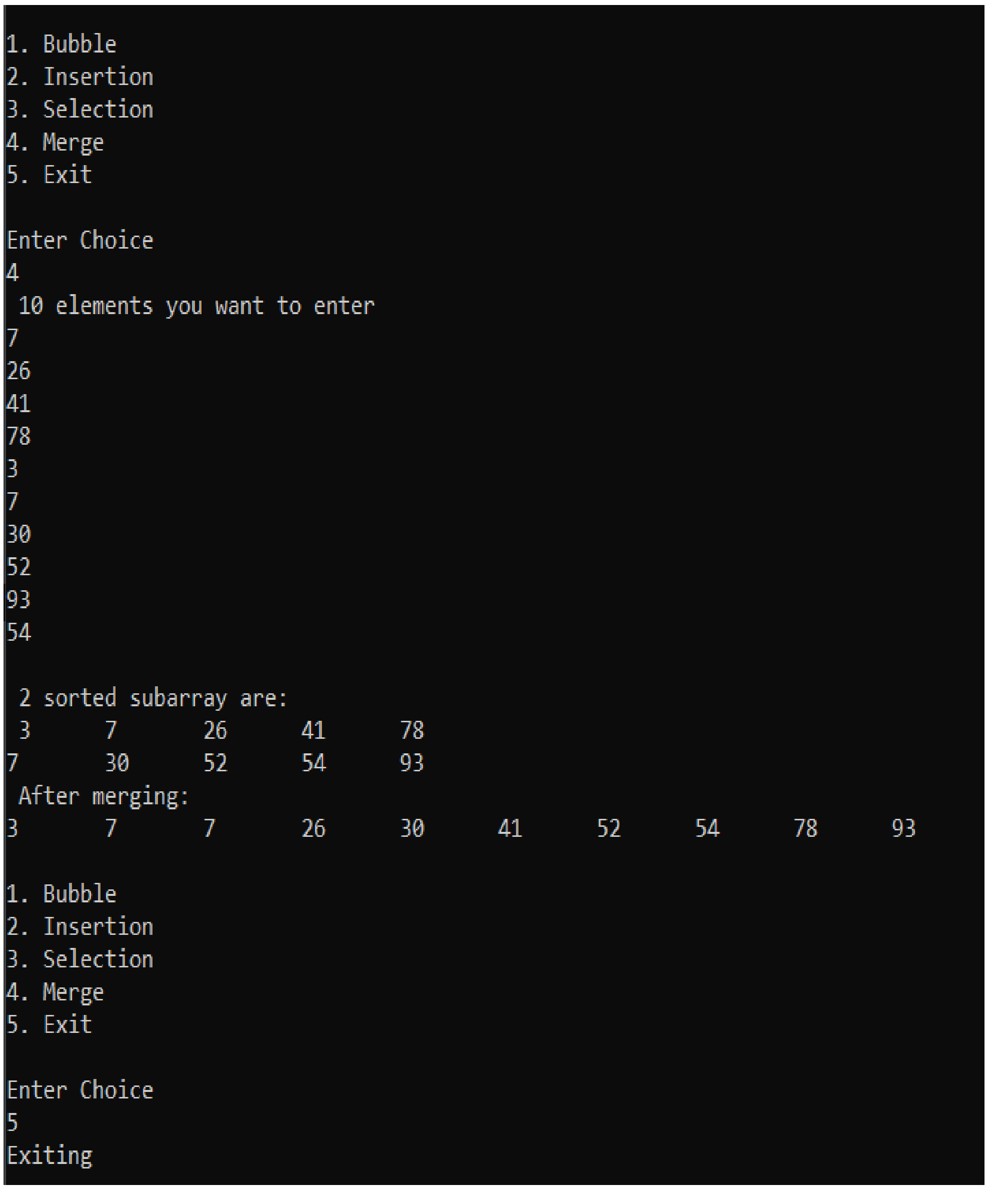
```
$ ./a.exe

1. Bubble
2. Insertion
3. Selection
4. Merge
5. Exit

Enter Choice
1
Number of elements: 5

Enter the elements:
Element 1: 14
Element 2: 95
Element 3: 68
Element 4: 2
Element 5: 37

Bubble sort ascending:
14 95 68 2 37
14 68 95 2 37
14 68 2 95 37
14 68 2 37 95
14 2 68 37 95
14 2 37 68 95
2 14 37 68 95

Bubble sort descending:
2 14 37 68 95
14 2 37 68 95
14 37 2 68 95
14 37 68 2 95
14 37 68 95 2
37 14 68 95 2
37 68 14 95 2
37 68 95 14 2
68 37 95 14 2
68 95 37 14 2
95 68 37 14 2
```



```
1.  Bubble
2.  Insertion
3.  Selection
4.  Merge
5.  Exit

Enter Choice
2
Number of elements: 7

Enter the elements:
Element 1: 21
Element 2: 86
Element 3: 35
Element 4: 9
Element 5: 72
Element 6: 54
Element 7: 44

Insertion sort ascending:
21 86 35 9 72 54 44
21 86 35 9 72 54 44
21 35 86 9 72 54 44
9 21 35 86 72 54 44
9 21 35 72 86 54 44
9 21 35 54 72 86 44
9 21 35 44 54 72 86

Insertion sort descending:
9 21 35 44 54 72 86
21 9 35 44 54 72 86
35 21 9 44 54 72 86
44 35 21 9 54 72 86
54 44 35 21 9 72 86
72 54 44 35 21 9 86
86 72 54 44 35 21 9
```



```
1. Bubble
2. Insertion
3. Selection
4. Merge
5. Exit

Enter Choice
3
Number of elements: 9

Enter the elements:
Element 1: 99
Element 2: 3
Element 3: 69
Element 4: 42
Element 5: 3
Element 6: 19
Element 7: 2
Element 8: 23
Element 9: 60

Selection sort ascending:
99 3 69 42 3 19 2 23 60
2 3 69 42 3 19 99 23 60
2 3 69 42 3 19 99 23 60
2 3 3 42 69 19 99 23 60
2 3 3 19 69 42 99 23 60
2 3 3 19 23 42 99 69 60
2 3 3 19 23 42 99 69 60
2 3 3 19 23 42 60 69 99
2 3 3 19 23 42 60 69 99

Selection sort descending:
2 3 3 19 23 42 60 69 99
99 3 3 19 23 42 60 69 2
99 69 3 19 23 42 60 3 2
99 69 60 19 23 42 3 3 2
99 69 60 42 23 19 3 3 2
99 69 60 42 23 19 3 3 2
99 69 60 42 23 19 3 3 2
99 69 60 42 23 19 3 3 2
99 69 60 42 23 19 3 3 2
```



```
1. Bubble
2. Insertion
3. Selection
4. Merge
5. Exit

Enter Choice
4
 10 elements you want to enter
7
26
41
78
3
7
30
52
93
54

 2 sorted subarray are:
 3      7       26      41      78
7       30      52      54      93
 After merging:
3       7       7       26      30      41      52      54      78      93

1. Bubble
2. Insertion
3. Selection
4. Merge
5. Exit

Enter Choice
5
Exiting
```

References

1) https://www.geeksforgeeks.org/sorting-algorithms/?ref=shm

2) https://www.geeksforgeeks.org/multithreading-c-2/

3) https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/