Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Computer Programming

Laboratory work #2

One-Dimensional Array Operations and Processing

Author:
Alexandru Rudoi
std. gr. FAF-231

Verified:
Alexandru Furdui

Chișinău 2023

Theory Background

Before implementing any sorting algorithms, it's crucial to have a solid understanding of how each algorithm works. So, here's a brief overview of the research and concepts which I have explored:

Bubble Sort: Bubble Sort repeatedly compares adjacent elements and swaps them if they are in the wrong order. This process continues until the entire array is sorted. *Time complexity:* $O(n^2)$ *in the worst case.*

Selection Sort: Selection Sort divides the array into two parts: a sorted part and an unsorted part. It repeatedly selects the smallest element from the unsorted part and moves it to the end of the sorted part. This process continues until the entire array is sorted. *Time complexity:* $O(n^2)$ *in the worst case.*

Insertion Sort: Insertion Sort divides the array into two parts: a sorted part and an unsorted part. It iterates through the unsorted part, taking one element at a time and inserting it into its correct position within the sorted part. This process continues until the entire array is sorted. *Time complexity:* $O(n^2)$ *in the worst case.*

Quick Sort: Quick Sort selects a pivot element from the array and partitions the array into two sub-arrays: one with elements less than the pivot and one with elements greater than the pivot. Quick Sort recursively applies this partitioning process to the sub-arrays until the entire array is sorted. *Time complexity:* $O(n^2)$ *in the worst case, but typically* $O(n \log n)$ *on average.*

0.1 Key Concepts and Notions

Time Complexity: Measure of how the runtime of an algorithm grows as the input size increases.

Comparison-Based Sorting: Sorting algorithms that rely on comparing elements to determine their order.

Stable Sorting: Sorting that maintains the relative order of equal elements.

In-Place Sorting: Sorting that rearranges elements within the original array without using additional memory.

Divide and Conquer: Algorithmic strategy that breaks a problem into smaller sub-problems and combines their solutions.

Pivot: A chosen element used to partition data in Quick Sort.

Efficiency: Evaluation of how well an algorithm performs in terms of time and space usage.

The Task

You will have to read from console an array of numbers(must be int) of length n(also from console), and implement these sorting algorithms, and explain them:

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

Technical implementation

Listing of the program: Github
Pseudo-code:

```
1. Including necessary C libraries (stdio.h, time.h, bool.h)

2. Declare function prototypes for sorting algorithms:
   - bubbleSort(arr[], n)
   - selectionSort(arr[], n)
   - insertionSort(arr[], n)
   - quickSort(arr[], low, high)

3. Declare function prototypes for helper functions:
   - partition(arr[], low, high)
   - copyArray(arr[], arr2[], n)
   - printArray(arr[], n)

4. Define the main function:
   - Initialize variables:
     - start (clock_t)
     - end (clock_t)
     - cpu_time_used (double)
     - n (integer)
     - i (integer)

   - Prompt the user to enter the number of elements (n)
   - Read the value of n from the user
```

```
   - Create arrays arr[n] and arr2[n]

   - Prompt the user to enter n integers and store them in
     arr[]

   - For each sorting algorithm (Bubble Sort, Selection Sort,
       Insertion Sort, Quick Sort):
      - Record the start time (start = clock())
      - Copy the original array (arr) to a temporary array (
        arr2)
      - Print the algorithm's␣name␣and␣sort␣the␣array␣using␣
        the␣algorithm
␣␣␣␣␣␣-␣Print␣the␣sorted␣array
␣␣␣␣␣␣-␣Record␣the␣end␣time␣(end␣=␣clock())
␣␣␣␣␣␣-␣Calculate␣the␣CPU␣time␣used␣for␣sorting
␣␣␣␣␣␣-␣Print␣the␣time␣taken

5.␣End␣of␣the␣main␣function

6.␣Define␣the␣Bubble␣Sort␣algorithm:
␣␣␣␣-␣Initialize␣variables:␣i,␣j,␣swap,␣swapped␣(boolean)
␣␣␣␣-␣Iterate␣through␣the␣array:
␣␣␣␣␣␣-␣Initialize␣swapped␣to␣false
␣␣␣␣␣␣-␣Iterate␣through␣the␣unsorted␣part␣of␣the␣array:
␣␣␣␣␣␣␣␣-␣If␣arr[j]␣>␣arr[j␣+␣1],␣swap␣arr[j]␣and␣arr[j␣+␣1]
␣␣␣␣␣␣␣␣-␣Set␣swapped␣to␣true␣if␣a␣swap␣occurred
␣␣␣␣␣␣-␣If␣swapped␣is␣false,␣break␣out␣of␣the␣loop␣(array␣is␣
   sorted)

7.␣Define␣the␣Selection␣Sort␣algorithm:
␣␣␣␣-␣Initialize␣variables:␣min_idx,␣swap
␣␣␣␣-␣Iterate␣through␣the␣array:
␣␣␣␣␣␣-␣Find␣the␣index␣of␣the␣minimum␣element␣in␣the␣unsorted␣
   part
␣␣␣␣␣␣-␣Swap␣the␣minimum␣element␣with␣the␣current␣element␣if␣
   necessary

8.␣Define␣the␣Insertion␣Sort␣algorithm:
```

```
   - Initialize variables: key, j
   - Iterate through the array:
     - Store the current element (key)
     - Move elements greater than key to create space for key
     - Place key in its correct position

9. Define the Quick Sort algorithm:
   - If low < high, do the following:
     - Partition the array and get the pivot index (pi)
     - Recursively quickSort the elements before and after 
       the pivot

10. Define the partition function:
    - Initialize variables: swap, pivot
    - Iterate through the array:
      - If an element is smaller than or equal to the pivot, 
        swap it
    - Swap the pivot element with the element at the correct 
      position
    - Return the index of the pivot

11. Define the copyArray function:
    - Copy elements from arr[] to arr2[]

12. Define the printArray function:
    - Print each element of the array
    - Print a newline character to separate the output
```

Results

In the next picture, it can be seen that the program successfully sorted a list of 50 integers in ascending order using four different sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, and Quick Sort. All four algorithms produced identical sorted sequences. The program's execution took approximately 36.202 seconds.

```
Admin@DESKTOP-3BO5FF3 MINGW64 ~/Desktop/University/PC/Lab-2
$ ./a.exe
Enter number of elements: 50
Enter 50 integers:
48 7 68 82 50 76 32 79 99 33 35 18 25 84 36 69 9 86 66 25 49 11 51 49 79 4 32 24 10 58 75 22 28 89 87 77 44 22 7 21 50 53 43 12 71 54 84 93 12 38

Sorted array using Bubble Sort algorithm:
4 7 7 9 10 11 12 12 18 21 22 22 24 25 25 28 32 32 33 35 36 38 43 44 48 49 49 50 50 51 53 54 58 66 68 69 71 75 76 77 79 79 82 84 84 86 87 89 93 99
Array was sorted in 0.032000 seconds

Sorted array using Selection Sort algorithm:
4 7 7 9 10 11 12 12 18 21 22 22 24 25 25 28 32 32 33 35 36 38 43 44 48 49 49 50 50 51 53 54 58 66 68 69 71 75 76 77 79 79 82 84 84 86 87 89 93 99
Array was sorted in 0.032000 seconds

Sorted array using Insertion Sort algorithm:
4 7 7 9 10 11 12 12 18 21 22 22 24 25 25 28 32 32 33 35 36 38 43 44 48 49 49 50 50 51 53 54 58 66 68 69 71 75 76 77 79 79 82 84 84 86 87 89 93 99
Array was sorted in 0.028000 seconds

Sorted array using Quick Sort algorithm:
4 7 7 9 10 11 12 12 18 21 22 22 24 25 25 28 32 32 33 35 36 38 43 44 48 49 49 50 50 51 53 54 58 66 68 69 71 75 76 77 79 79 82 84 84 86 87 89 93 99
Array was sorted in 0.025000 seconds
```

Conclusion

In summary, working on these sorting algorithms was a significant step in my journey toward becoming a future software engineer. It deepened my understanding of efficient code and expanded my knowledge in algorithmic thinking. This experience has equipped me with essential skills and insights that will undoubtedly benefit my future in software engineering.

Bibliography

1. The overview of Computer Programming course lessons for students (lecturer: associate professor M. Kulev). Chisinau, UTM, FCIM, 2023.

2. https://www.programiz.com/dsa/bubble-sort

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

4. https://www.javatpoint.com/quick-sort

5. https://www.khanacademy.org/computing/computer-science/algorithms/insertion-sort/a/insertion-sort