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:
Mihai Caraman
std. gr. FAF-233

Verified:
Alexandru Furdui

Chișinău 2023

Theory Background

Sorting algorithms are fundamental procedures in computer science used to arrange elements of a data structure (e.g., an array, list, or set) in a specific order, typically in an ascending or descending fashion.The order can be based on numerical, lexicographical, or other criteria.

1. Bubble Sort: Bubble Sort is a simple comparison-based sorting algorithm that repeatedly steps through the list, compares each pair of adjacent items and swaps them if they are in the wrong order. The process is repeated until the list is sorted.

2. Selection Sort: Selection Sort is also a simple comparison-based sorting algorithm. It works by dividing the array into a sorted and an unsorted region. In each iteration, it selects the smallest (or largest, depending on the order) element from the unsorted region and swaps it with the leftmost unsorted element, thereby expanding the sorted region.

3. Insertion Sort: Insertion Sort is an efficient comparison-based sorting algorithm. It builds the sorted array one item at a time by repeatedly picking the next item from the unsorted part of the array and inserting it into its correct position within the sorted part.

4. Quick Sort: Quick Sort is a popular efficient, in-place, and comparison-based sorting algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The process is then applied recursively to the sub-arrays.

5. Cocktail Sort (Bidirectional Bubble Sort): Cocktail Sort is a variation of Bubble Sort where the list is sorted by comparing and swapping adjacent elements, first from left to right (like in Bubble Sort), and then from right to left in subsequent passes. This approach works by bringing the smallest item to the beginning and the largest to the end with each pass.

The Task

Describe your task, and enumerate the task/tasks you have implemented:

1. Task 2.3 (Hard) - I implemented in a program all 4 sorting algorithms and as a bonus i included a fancy algorithm(Cocktail sort)

Technical implementation

Pseudocode

```
procedure BubbleSort(array, length)
```

```
    for i from 0 to length - 1
        for j from 0 to length - 1 - i
            if array[j] > array[j+1]
                swap(array[j], array[j+1])
    end for
end procedure

procedure SelectionSort(array, length)
    for i from 0 to length - 2
        min = i
        for j from i + 1 to length - 1
            if array[j] < array[min]
                min = j
        end for
        if min     i
            swap(array[i], array[min])
    end for
end procedure

procedure InsertionSort(array, length)
    for i from 1 to length - 1
        key = array[i]
        j = i - 1
        while j >= 0 and array[j] > key
            array[j + 1] = array[j]
            j = j - 1
        end while
        array[j + 1] = key
    end for
end procedure

procedure Swap(x, y)
    temp = x
    x = y
    y = temp
end procedure

function Partition(array, low, high)
```

```
    pivot_index = low + (random() mod (high - low))
    if pivot_index     high
        Swap(array[pivot_index], array[high])
    pivot_value = array[high]
    i = low
    for j from low to high - 1
        if array[j] <= pivot_value
            Swap(array[i], array[j])
            i = i + 1
        end if
    end for
    Swap(array[i], array[high])
    return i
end function

procedure QuickSortRecursion(array, low, high)
    if low < high
        pivot_index = Partition(array, low, high)
        QuickSortRecursion(array, low, pivot_index - 1)
        QuickSortRecursion(array, pivot_index + 1, high)
    end if
end procedure

procedure QuickSort(array, length)
    QuickSortRecursion(array, 0, length - 1)
end procedure

procedure CocktailSort(array, length)
    swap = 1
    beg = 0
    end = length - 1
    while swap
        swap = 0
        for i from 0 to end - 1
            if array[i] > array[i + 1]
                Swap(array[i], array[i + 1])
                swap = 1
            end if
```

```
        end for
        if not swap
            break
        end if
        swap = 0
        for i from end - 1 to beg step -1
            if array[i] > array[i + 1]
                Swap(array[i], array[i + 1])
                swap = 1
            end if
        end for
        beg = beg + 1
    end while
end procedure

procedure Main()
    length = inputLength()  // Get array length from user
    array = inputArray(length)  // Get array elements from
       user

    BubbleSort(array, length)
    SelectionSort(array, length)
    InsertionSort(array, length)
    QuickSort(array, length)
    CocktailSort(array, length)
end procedure
```

Results

Describe your results.

The provided C program implements several sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, and Cocktail Sort. The program takes an array size and elements as input from the user, sorts the array using each of these algorithms, and prints the sorted array and the time taken for each sorting method. The program provides the sorted arrays and the time taken by each sorting algorithm. These times represent how long each sorting algorithm took to sort the original array. Looking at the results(i will put some text files with results from the console), we can see that bubble sort and selection sort aren't that efficient with larger arrays, however, the other three , insertion,

cocktail and quick sort seem all to work pretty well with medium and larger arrays, but the best remains quick sort.

Conclusion

Quick sort stands out among the five for its average and best-case time complexity of O(n log n) and good performance in practice.

Insertion sort is efficient for small datasets and nearly sorted data.

Bubble sort and selection sort are simple and easy to implement but highly inefficient for large datasets.

Cocktail sort, although a variation of bubble sort, does not offer significant advantages in terms of efficiency, however, after numerous testings , it seems to work really efficient on small and medium arrays, like insertion sort.

Bibliography

1. https://www.geeksforgeeks.org/
2. https://www.programiz.com/
3. https://www.tutorialspoint.com/
4. https://github.com/caramisca/Lab-2 (github link with all results and the source code)