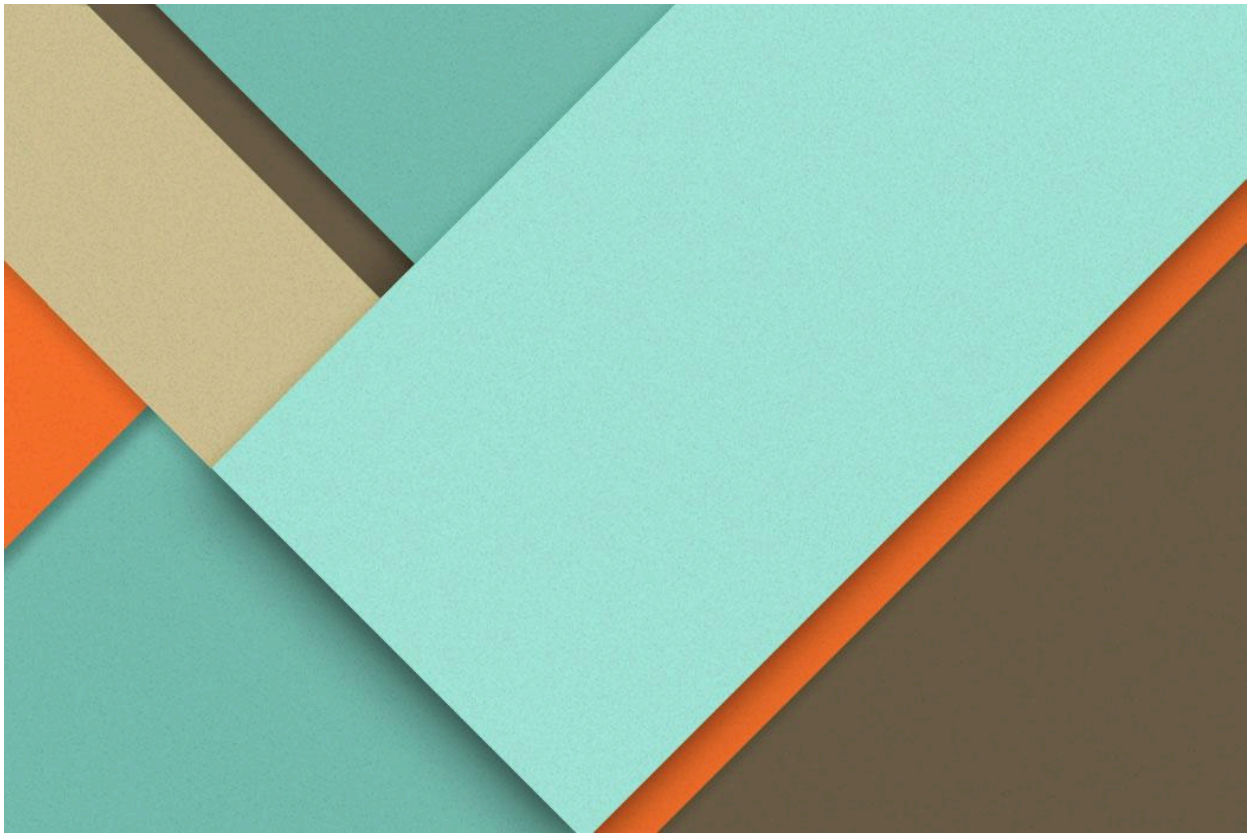


Selection Sort

CS346: Software Engineering Lab

Assignment 1 :: Milestone 1 Overall Planning of the Software

—

M Devi Naga Sai Srinivas (210101070)
B.Tech 3rd Year,
Dept of CSE, IITG

Objective

To create a software tool for visualizing the Selection Sort algorithm to guide beginners through the step-by-step execution of this simple sorting algorithm. The application aims to provide a user-friendly interface to enhance understanding by representing each stage of the algorithm, making the sorting process more comprehensible for users.

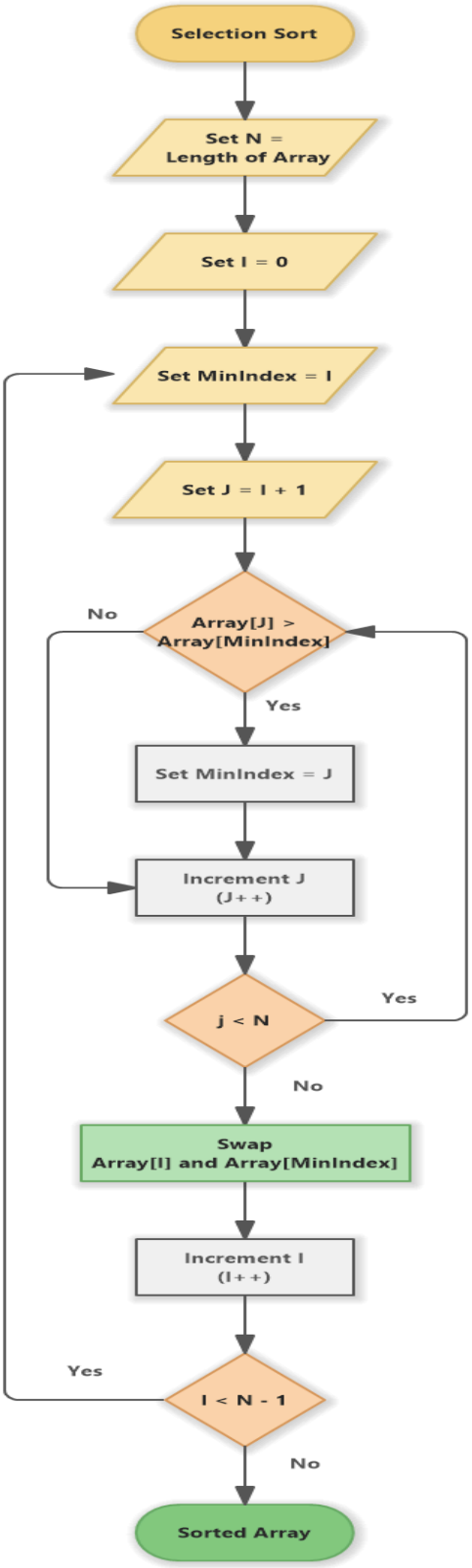
Project Scope

The project involves developing a Visual Basic application that takes an array of integers as input and visually represents the step-by-step execution of the Selection Sort Algorithm. The application aims to offer a user-friendly interface, displaying the array elements and visually indicating the selection and swapping of elements during each step. The output will showcase the sorted array.

Target Audience

The intended users include school children, individuals unfamiliar with the basics of sorting, particularly Selection Sort, and those enrolled in their initial coding or computer science courses.

Algorithm Flow & Description



Initialization:

- Prompt the user to enter the length of the array.
- Allow users to input numbers into an array.

Input Validation:

- Check for the validity of inputs.
- Provide user-friendly error messages for invalid inputs.
  (check the section titled **Limitations and Edge Cases**)

Selection Sort Algorithm:

- Run **n-1** iterations of Selection Sort when the user enters the **SORT** button.
- In the **k-th** iteration, traverse the input array from the k-th index to the last element.
- Find the minimum element (lexicographically smallest) and swap it with the element at index k.

Visualization:

- Display the numbers being compared at each iteration.
- Show the partially sorted output and highlight it at the end of each iteration.

Pseudo code of Selection Sort

Intuition:

The idea is to divide the given array into two segments: **Sorted** and **Unsorted**. Initially, the entire array is treated as unsorted. The algorithm iteratively identifies the **smallest** element (for ascending order) from the unsorted section of the list, **swaps** it with the first element in the unsorted segment, and incorporates that index into the sorted part of the array. This procedure continues for the remaining unsorted portion until the entire list is arranged in ascending order.

Pseudo Code:

```
procedure selection sort
   array  : array of items
   n      : size of array
   for i in range(len-1):
   /* set current element as minimum*/
    minIndex = i
    /* check the element to be minimum */
    for j in range(i, len-1):
        if Arr[minIndex] > Arr[j]:
            minIndex = j
        end if
    end for
    /* swap the minimum element with the current element*/
    Swap(Arr[minIndex],Arr[i])
   end for
end procedure
```

Time Complexity:

Best Case: O(n^2)

Worst Case: O(n^2)

Limitations and Edge Cases

Limitations:

- **Data format : Integer**
    - For improving user experience by providing a simple UI, we restrained ourselves with integers.
- **Array Size : 1<=n<=10 elements**
    - Considering the screen size we choose to make up to 10 elements.
- **Array element range : -99 <=A[i]<=99**
    - For The Same reason as above, we choose to make 2 digits.
- **Algorithm :** Proceeds one step at a time as per flowchart of selection sorting.
    - This could enable the user for easy understanding of the steps involved in the selection sort algorithm.

Invalid Inputs:

- No special chars are allowed in between (12@#3)
- Mixed-type inputs are not allowed. (123asd)
- No non-integer input is allowed. (abc **or** 1.23)
- Blank inputs are not allowed. ( )

Edge/Corner Cases:

Error dialogue will be displayed if the above constraints are not satisfied.

System Architecture

The application will have a simple architecture with a user interface and the core algorithm implementation. Using Visual Basic 2010 for the development.

Design Ideation for Visualization of Comparisons and Swapping

- I intend to use Visual Basic for creating the interface.
- Implementing effective coloring to highlight the minIndices collected during iterations.
- Utilizing effective coloring to visually distinguish the two integers currently being swapped.
- Incorporating buttons for Clear, Enter, Sort, and Next Step functionalities.

BUTTON FUNCTIONALITIES:

- **Enter:** Entering the input size and array elements to sort.
- **Clear:** Resets all logs and returns to the initial menu.
- **Start:** Initiates the Selection Sort algorithm.
- **Next Step:** Proceeds to the next iteration of sorting if the array remains unsorted.

TEXT FIELDS:

- **Input Field:** Accepts input size and integers to sort from the user.
- **Current Comparison:** Displays the comparison (greater than or less than) between the element to be inserted and the element under consideration.
- **Output Field:** Dynamic set of boxes illustrating partial output at each step of the Selection Sort.

ERROR PREVENTION:

- The input text field becomes read-only after the sorting process begins, and comparison and output fields remain read-only throughout.
- All fields are designed to be multi-line, accommodating large inputs without text box overflow.
- The "Clear" and "Next Step" buttons disappear during an iteration of Selection Sort to prevent additional errors.

Dry Run of the application:

(Designed in canva)

Currently, I am showcasing a generic appearance and structure, which I aim to align with the final project in terms of visualizing the iterations.

Input Array:

Initially:

Array is unsorted, so the algorithm assigns **minIndex = 0, i = 0**.

(current element is indicated with **red** font)

After traversal, **minIndex = 4** (min element 2 is in index 4).

Swaps Array[0] and Array[4].

After 1st iteration:

After traversal from **i=1**, **minIndex = 2**.

Swaps Array[1] and Array[2].

After 2nd iteration:

After traversal from **i=2**, **minIndex** doesn't change. So, no swapping.

After 3rd iteration:

After traversal from **i=3**, still **minIndex** doesn't change so, no swapping.

After 4th iteration:

Finally, the largest value within the array is automatically positioned at the last index. The resulting array represents the sorted arrangement.

Sorted Array:

(In the application, I'm thinking of visualizing the swapping through arrows if possible.)

NOTE:

Additional documentation for the software will be incorporated into the final project. The outlined plan presented here serves as the fundamental structure envisioned to ensure alignment with the goals of the intended stakeholders.