CS346 Software Engineering Lab

~Overall Plan

SELECTION SORT

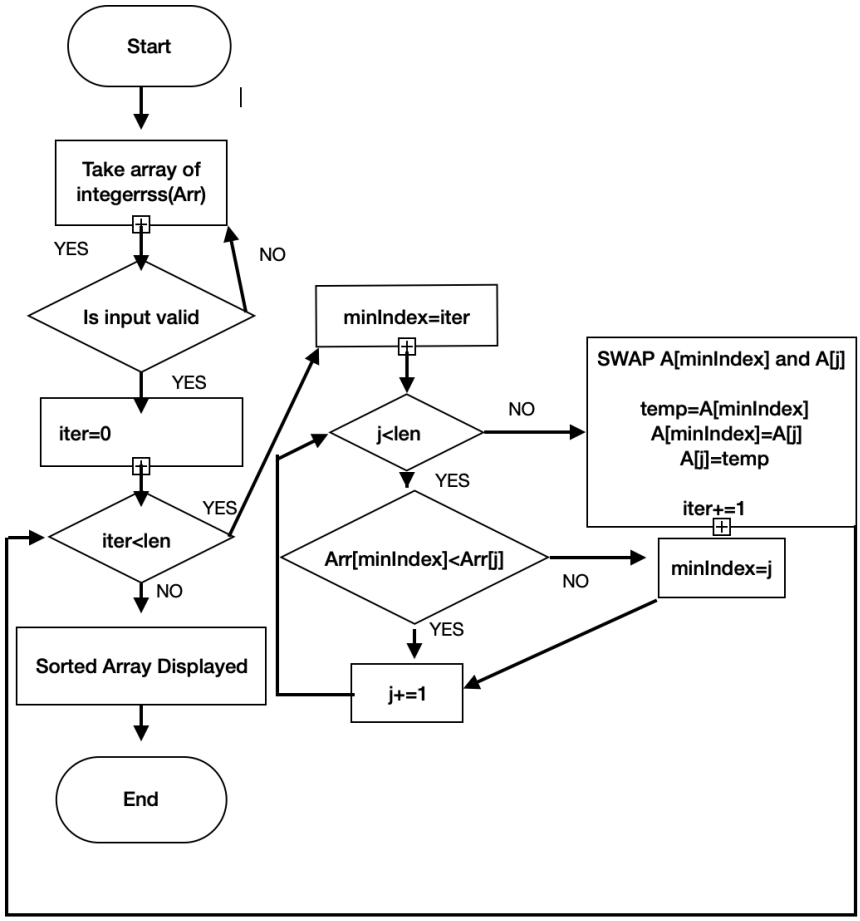
- **NAME:HIMANGSHU DEKA**
- **ROLL NO:210101050**

- OBJECTIVE AND DESIGN GOALS:

To implement Selection Sort and visualise the intermediate steps cretively (aka comparison and swapping)

- STAKEHOLDERS:

School Children,anyone unfamiliar with the ABCs of sorting and more specifically Selection Sort,and people enrolled into their first course on coding and/or CompSci.



- FLOW CHART DESCRIPTION:

1. Initially Start by asking the user to enter firstly, the length of the array and then to fill in the bubbles/buttons that appear on the screen,some integers.
2. Then check for validity of the inputs (Covered below in the section title Invalid Inputs and Corner Cases).
3. Run len-1 iterations of selection sort when user enters «SORT».
4. In (k)th iteration of the algorithm,traverse the input array from (k)th index to the last element.
5. For this initilaise in every such iteration of Pt:4; minIndex as k and then upon traversing the array as mentioned in Pt.4 ,set minIndex to the index of the minimum element(or lexicographically smallest element)
6. Upon completion of the inner for loop(or point 4):Swap Arr[iter] and Arr[minIndex]
7. Display the numbers being compared at each iteration.
8. At the end of each iteration display the partially sorted ouptut and highlight it
9. Continue till the whole array is sorted.

- PSUEDO CODE OF THE ALGORITHM:

The idea is to think of the given array of elements as two parts – Sorted and Unsorted. Initially, the whole array is to be considered unsorted. The algorithm repeatedly selects the smallest (for ascending order) element from the unsorted portion of the list and swaps it with the first element of the unsorted part and adds that index to the sorted part of the array. This process is repeated for the remaining unsorted portion until the entire list is sorted.

```
for (i=0 to len-1):
    minIndex=i
    for(j=i to len-1):
      if(Arr[minIndex]>Arr[j]):
        minIndex=j
```

```
temp=Arr[minIndex];
Arr[minIndex]=Arr[j]
 Arr[j]=temp
```

Time Complexity:

Worst Case:O(N^2)
Best Case: O(N^2)

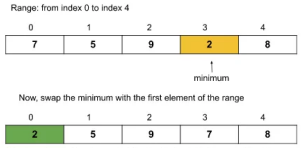
Dry run:

The following dry run will clarify the concepts:

Assume the given array is: {7, 5, 9, 2, 8}

Outer loop iteration 1:

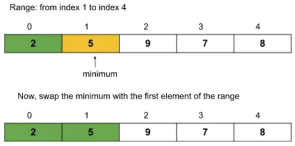
The range will be the whole array starting from the 1st index as this is the first iteration. The minimum element of this range is 2(**found using the inner loop**).



After iteration 1, the element at the first index is in its correct position.

Outer loop iteration 2:

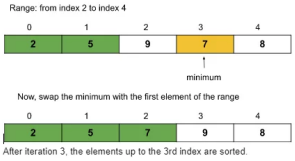
The range will be from the **[2nd index to the last index]** as the array is sorted up to the first index. The minimum element of this range is 5(**found using the inner loop**).



After iteration 2, the elements up to the 2nd index are sorted.

Outer loop iteration 3:

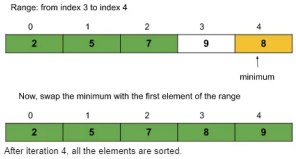
The range will be from the **[3rd index to the last index]**. The minimum element of this range is 7(**found using the inner loop**).



After iteration 3, the elements up to the 3rd index are sorted.

Outer loop iteration 4:

The range will be from the **[4th index to the last index]**. The minimum element of this range is 8(**found using the inner loop**).



After iteration 4, all the elements are sorted.

So, after 4 iterations(i.e. n-1 iterations where n = size of the array), the given array is sorted.

- LIMITATIONS,CORNER CASES AND INVALID INPUTS:

Corner/ Edge Cases:

1) Invalid Inputs:

   a. No non integer allowed
   b. No blank space allowed(eg 1 2)
   c. No Number greater than 100 allowed(only 2 digits) (while enterring as string)
   d. No number less than -99 is allowed(only1 digit negative numbers are allowed)
   e. Mixed Type inputs of the form abc123 not allowed
   f. No floating point/non integeal no.s are allowed
   g. Blank input (eg) not allowed
   h. No special chars allowed in between(eg: 1&2,1+2 etc.)

2) Behaviour in coner cases :

   a. When only numbers (+ve /-ve) behaviour is as expected.

- **DESIGN IDEA FOR VISUALISING COMPARISONS AND SORTING:**

~I plan to use Visual Basic to add interface .

~Effective coloring that will demarcate the minIndices that ahave been collected so far in the iterations.

~Effective colouring to visualise/demarcate the two integers being swapped curently.

~Buttons for Clear,Enter,Exit,Sort,Next Step.

I.BUTTONS LIKELY TO BE USED

- Clear : Clears all logs and brings back the starting menu.
- Start : Starts Selection Sort .
- Next Step : Iterates the next round of sorting if array is still unsorted
- Exit : Close the application
- Help : Redirects user to user documentation

II: TEXT FIELDS

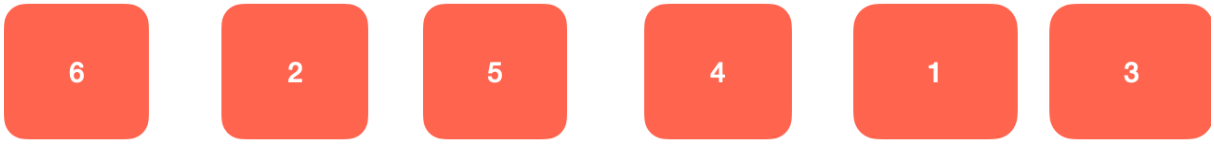
- Input Field: Takes input string/ decimals from user
- Current Comparison: Shows greater than, less than comparison between element to be inserted and element being compared against
- Output Field: Dynamic set of boxes showing partial output at each step of the insertion sort

III: ERROR PREVENTION:

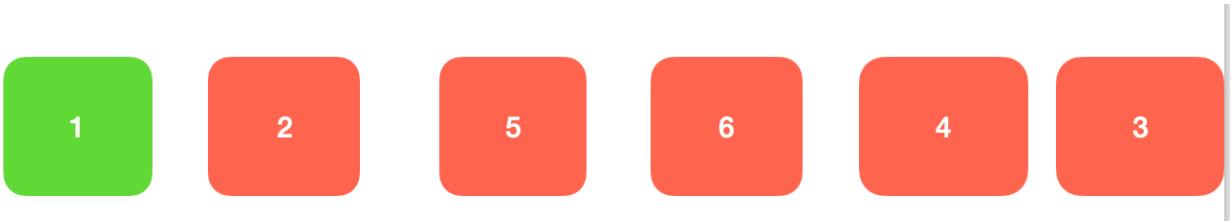
- Making the input text field read-only post sorting start, and keeping comparison and output read only throughout
- Making all fields multi-line to accommodate large inputs without text box overflow
- Clear and Next step buttons disappear during an iteration of insertion sort to prevent extra errors.

AS OF NOW,I AM PRESENTING THEGENERIC LOOK AND FIELD ,WHICH I SHALL TRY TO MATCH WITH THE FINAL PROJECT,AS TO HOW THE ITERATIONS WILL BE VISUALISED::

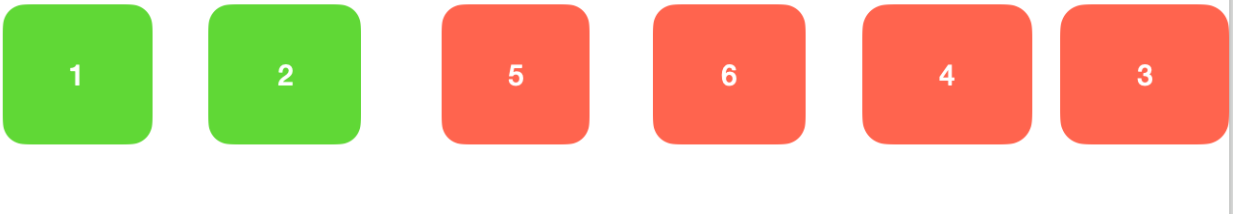
~Done using Canva



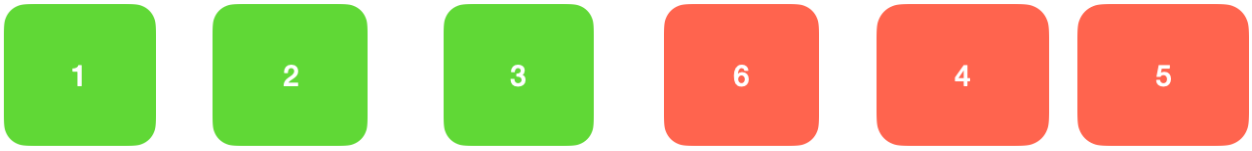
INITIALLY Situation is as shown as above



First pass: Initially the whole array is unsorted,the algorithm assigns iter=0 and fixes minIndex=0 initially.Thereafer upon traversing the whole length of the array (with j) it obtains minIndex=4(ie 1 is the smallest) and then swaps Arr[4] with Arr[0]



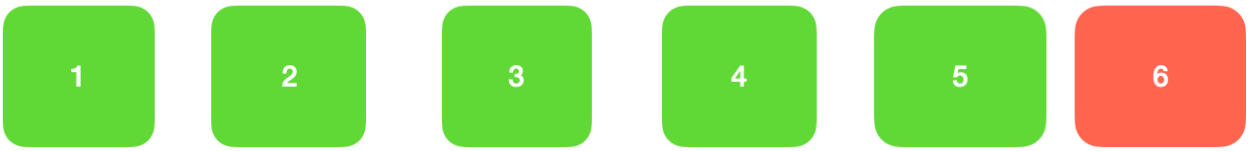
Second pass: For the second position, where 6 is present, fix iter=1 and minIndex=1 and again traverse the rest of the array in a sequential manner. After traversing,we find that minIndex doesn't change and therefore we



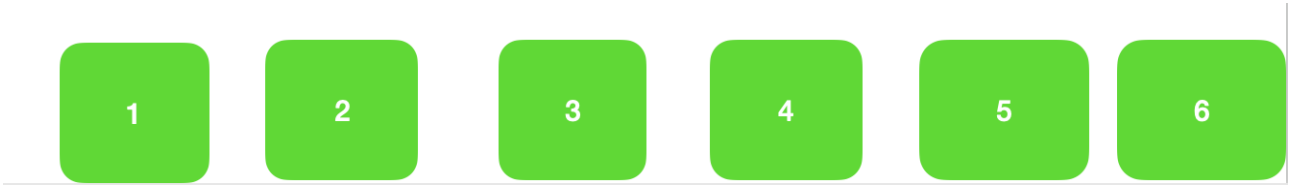
make no swaps in this case.

Third pass: Now, for third place, where 5 is present,fix iter=2 and again traverse the rest of the array and find the least value present in the remainder of the array. While traversing,5 is the position of the least value(~=minIndex) in Unsorted part.swap Arr[5] with Arr[2]

Fourth pass:Now iter is fixed as 3 and then in the remaining unsorted array the minIndex comes out to be 3 itself(no change) so do no swaps here and simply increment the iter.



Fifth pass: Now for the 5th element,initialise iter=4 and then traverse the remaining array,as 5 comes out to be the least element (<6) so swap 5 and 6.



Sixth pass: At last the largest value present in the array automatically get placed at the last position in the array .The resulted array is the sorted array.

NOTE:
Further Documentation about the software shall be included in the final project ;This plan presented here is the basic outline of the project envisioned to make it meets the goals of the intended stakeholders.