Experiment 1 [A]

Problem Statement: Given a sequence of n numbers <a1, a2, ...,an>, find a permutation (reordering) <a1', a2', ...,an'> of the input sequence such that a1'<= a2'<=, ...,<=an', using insertion sort algorithm.

Theory

Insertion sort is a simple sorting algorithm that allows for efficient, in-place sorting of the array, one element at a time. By in-place sorting, we mean that the original array is modified and no temporary structures are needed.

Algorithm

- ➔ Accept n, the number of elements
- ➔ Accept the data (numbers) into info
- ➔ Loop pos from 2 to n in steps of 1
  - ◆ Y info(pos)
  - ◆ Loop i from pos -1 to 1st element steps of -1
    - if y < info(i)
      - x(i+1) x(i)
    - else Break out of loop (goto 3.3)
  - ◆ x(i+1) y
- ➔ Display the sorted array, info

Example

Consider an array of size 5

| 5 | 4 | 3 | 2 | 1 |

Pass 1: | 4 | 5 | 3 | 2 | 1 |
Pass 2: | 3 | 4 | 5 | 2 | 1 |
Pass 3: | 2 | 3 | 4 | 5 | 1 |
Pass 4: | 1 | 2 | 3 | 4 | 5 |

Sorted array: | 1 | 2 | 3 | 4 | 5 |

Complexity Analysis

Time Complexity

In general, the time complexity depends on the size and arrangement of values within an array. Therefore, the standard terms like “best-case” and “worst-case” complexities are a measure of time-complexities for algorithms.

Worst-Case: O(n^2) – The scenario when the array is in descending order. Meanwhile, the algorithm tries to sort in the ascending order.

Best-Case: O(n) – The scenario when the array is already in a sorted order.

Overall, the time complexity for Insertion Sort is O(n^2).

Space Complexity

In simpler terms, space complexity refers to the amount of excess space or memory used during the running of the algorithm. Since we are only using extra variables like value, i and j, the space complexity is O(1).

Dheeraj Lalwani
1902085

Code

```
#include <stdio.h>
#define MAX 100
int data[MAX], n, i;

void printArray(int array[])
{
   for (i = 0; i < n; i++)
   {
      printf("| %d |", array[i]);
   }
   printf("\n");
}

void insertionSort(int array[])
{
   int i, j, temp;
   for (i = 1; i < n; i++)
   {
      temp = array[i];
      j = i - 1;
      while (j >= 0 && array[j] > temp)
      {
         array[j + 1] = array[j];
         j--;
      }
      array[j + 1] = temp;
      printf("Pass %d: ", i);
      printArray(array);
      printf("\n");
   }
}

int main()
{
   printf("\n");
   printf("Enter number of elements: ");
   scanf("%d", &n);
```

```
    printf("Start entering %d integers: \n", n);
    for (i = 0; i < n; i++)
    {
        printf("data[%d] = ", i);
        scanf("%d", &data[i]);
    }

    printf("\n------- Array before sorting -------\n");
    printArray(data);
    printf("------- x ------- x ------- x -------\n\n");

    insertionSort(data);

    printf("------- Array after sorting -------\n");
    printArray(data);
    printf("------- x ------- x ------- x -------\n\n");

    return 0;
}
```

Output

```
Enter number of elements: 7
Start entering 7 integers:
data[0] = 7
data[1] = 3
data[2] = 2
data[3] = 5
data[4] = 4
data[5] = 1
data[6] = 6
```

```
------- Array before sorting -------
| 7 || 3 || 2 || 5 || 4 || 1 || 6 |
------- x ------- x ------- x -------

Pass 1: | 3 || 7 || 2 || 5 || 4 || 1 || 6 |

Pass 2: | 2 || 3 || 7 || 5 || 4 || 1 || 6 |

Pass 3: | 2 || 3 || 5 || 7 || 4 || 1 || 6 |

Pass 4: | 2 || 3 || 4 || 5 || 7 || 1 || 6 |

Pass 5: | 1 || 2 || 3 || 4 || 5 || 7 || 6 |

Pass 6: | 1 || 2 || 3 || 4 || 5 || 6 || 7 |

------- Array after sorting -------
| 1 || 2 || 3 || 4 || 5 || 6 || 7 |
------- x ------- x ------- x -------
```