Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester:(Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 02

Course Title: Data Structure Lab

Course Code: CSE 106 Section: 231-D1

Lab Experiment Name: **Implement Bubble Sort using Arrays**

Student Details

Name		ID
1.	Promod Chandra Das	231002005

Lab Date	**:**	
Submission Date	**:**	
Course Teacher's Name	**:**	**Md. Shihab Hossain**

Lab Report Status

Marks: ………………………………… **Signature:.....................**

Comments:... **Date:..............................**

➢ Objective(s):

- To be familiar with different type of sorting algorithms.
- To learn problem solving techniques using C.

➢ Problem Analysis

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst-case complexity are of (n^2) where n is the number of items.

❖ Step-by-step example:

Take an array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort. In each step, elements written in **bold** are being compared. Three passes will be required;

❖ First Pass:

(**5 1** 4 2 8) → (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
(1 **5 4** 2 8) → (1 **4 5** 2 8), Swap since 5 > 4
(1 4 **5 2** 8) → (1 4 **2 5** 8), Swap since 5 > 2
(1 4 2 **5 8**) → (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

❖ Second Pass:

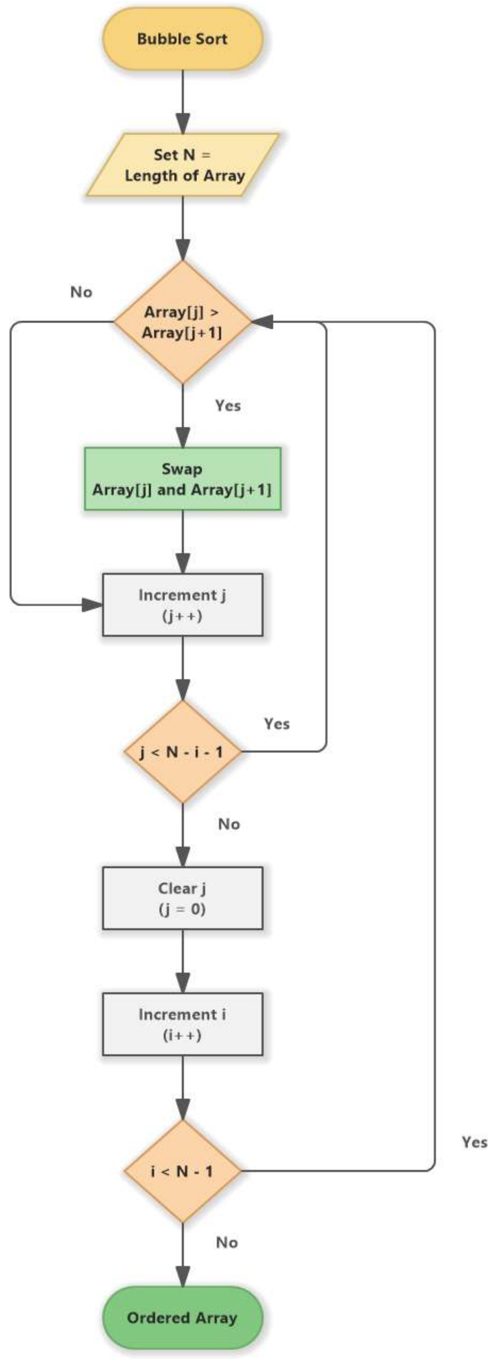
(**1 4** 2 5 8) → (**1 4** 2 5 8)
(1 **4 2** 5 8) → (1 **2 4** 5 8), Swap since 4 > 2
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

Now, the array is already sorted, but the algorithm does not know if it is completed. The algorithm needs one additional whole pass without any swap to know it is sorted.

❖ Third Pass:

(**1 2** 4 5 8) → (**1 2** 4 5 8)
(1 **2 4** 5 8) → (1 **2 4** 5 8)
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

➢ <u>**Flowchart**</u>



➢ **Bubble Sort on Linked List**

Given a singly linked list, sort it using bubble sort.

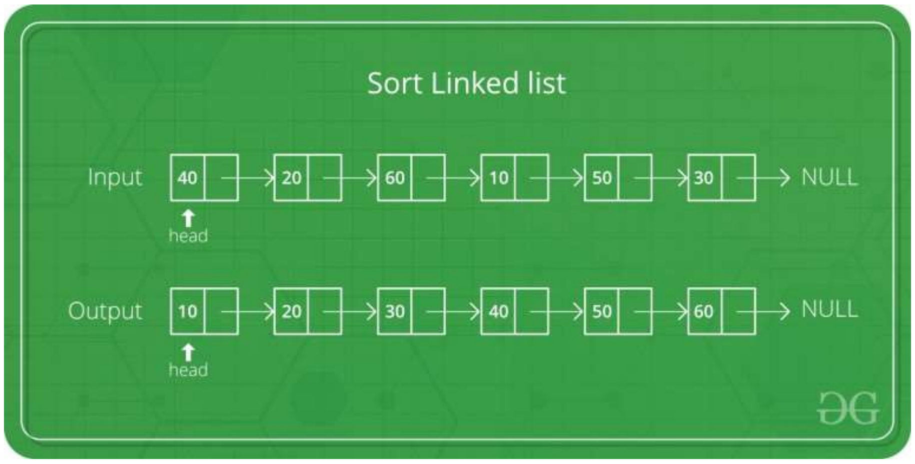


Figure 1: A Singly Sort Linked List

➢ <u>**Lab Exercise (Submit as a report)**</u>

- Implement Selection Sort algorithm using arrays.
- Implement Selection Sort algorithm using Linked List(Singly).

❖ <u>**Implement Selection Sort algorithm using arrays.**</u>

```
#include <stdio.h>


void selectionSort(int arr[], int n) {
  int i, j, minIndex, temp;


  // Move boundary of unsorted array one by one
  for (i = 0; i < n-1; i++) {
    // Find the minimum element in unsorted array
    minIndex = i;
    for (j = i+1; j < n; j++)
      if (arr[j] < arr[minIndex])
```

```
            minIndex = j;

        // Swap the found minimum element with the first element
        temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}

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

int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Original array: \n");
    printArray(arr, n);

    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);

    return 0;
}
```

❖ **Output:**

```
/tmp/bBzImDb1Uh.o
Original array:
64 25 12 22 11
Sorted array:
11 12 22 25 64


=== Code Execution Successful ===
```

❖ **Implement Selection Sort algorithm using Linked List(Singly).**

```
#include <stdio.h>

#include <stdlib.h>


// Node structure

struct Node {

   int data;

   struct Node* next;

};


// Function to create a new node

struct Node* createNode(int data) {

   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

   newNode->data = data;

   newNode->next = NULL;

   return newNode;

}


// Function to print the linked list

void printList(struct Node* head) {

   struct Node* temp = head;

   while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Function to perform selection sort on the linked list
void selectionSort(struct Node* head) {
    struct Node* temp1 = head;

    while (temp1 != NULL) {
        struct Node* min = temp1;
        struct Node* temp2 = temp1->next;

        while (temp2 != NULL) {
            if (temp2->data < min->data) {
                min = temp2;
            }
            temp2 = temp2->next;
        }

        // Swap data
        int temp = temp1->data;
        temp1->data = min->data;
        min->data = temp;

        temp1 = temp1->next;
    }
}

// Function to insert a node at the end of the list
```

```
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }

    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

int main() {
    struct Node* head = NULL;

    insertEnd(&head, 64);
    insertEnd(&head, 25);
    insertEnd(&head, 12);
    insertEnd(&head, 22);
    insertEnd(&head, 11);

    printf("Original list: \n");
    printList(head);

    selectionSort(head);

    printf("Sorted list: \n");
    printList(head);
    return 0;
}
```

❖ Output:

```
tmp/FnKousYHKR.o
Original list:
64 -> 25 -> 12 -> 22 -> 11 -> NULL
Sorted list:
11 -> 12 -> 22 -> 25 -> 64 -> NULL



=== Code Execution Successful ===
```

➢ Discussion & Conclusion

Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted. The algorithm is named for the way smaller elements "bubble" to the top of the list. Though easy to implement, Bubble Sort is inefficient for large datasets with a time complexity of O(n2)O(n^2)O(n2). It is more suitable for educational purposes and small datasets.