Department of
Computer Science and Engineering

Title: Implement Implementation of bBubble sort, insertion sort, selection sort

Data Structure Lab

CSE 106

Green University of Bangladesh

1 Objective(s)

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

2 Sorting

Sorting refers to ordering data in an increasing or decreasing manner according to some linear relationship among the data items. List of some sorting algorithm.

1. Bubble Sort
2. Insertion Sort
3. Selection Sort

3 Bubble Sort

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**)

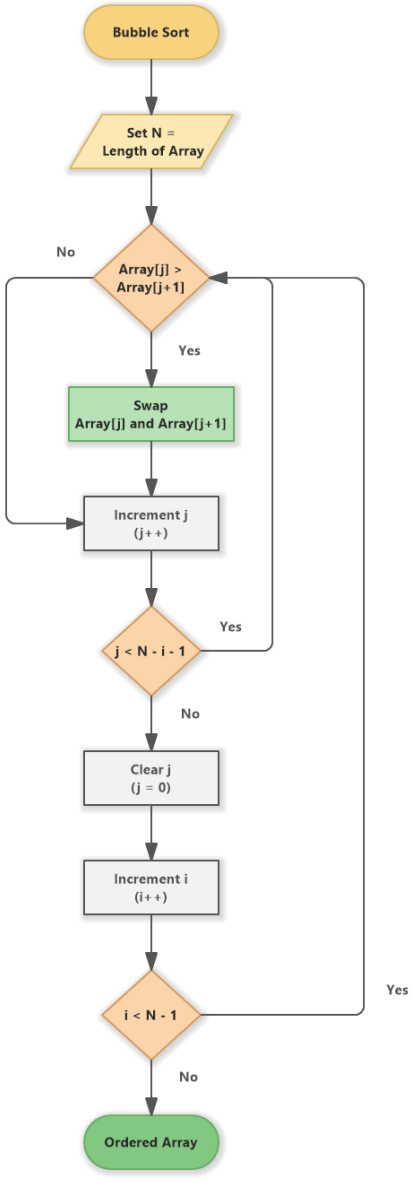
3.1 Algorithm

Algorithm 1: Bubble Sort Algorithm

Input: Array *A[ ]*
Output: Sorted Array *A[ ]*

```
/* Algorithm for Bubble Sort                                    */
int i, j;
N=length(A)
if j=1 to N then
    if i=0 to N-1 then
        if A[i] > A[i+1] then
            temp=A[i]
            A[i]=A[i+1]
            A[i+1]=temp
        end
    end
end
```

3.2 Flowchart



3.3 Implementation in C

```
1
2  /* Bubble sort code */
3  #include <stdio.h>
4  int main()
5  {
6    int array[100], n, c, d, swap;
7    printf("Enter number of elements:\n");
8    scanf("%d", &n);
9    printf("Enter %d integers:\n", n);
10   for (c = 0; c < n; c++)
11     scanf("%d", &array[c]);
12   for (c = 0 ; c < n - 1; c++)
13   {
14     for (d = 0 ; d < n - c - 1; d++)
15     {
16       if (array[d] > array[d+1]) /* For decreasing order use '<' instead of '>'
             */
17       {
18         swap       = array[d];
19         array[d]   = array[d+1];
20         array[d+1] = swap;
21       }
22     }
23   }
24   printf("Sorted list in ascending order:\n");
25   for (c = 0; c < n; c++)
26      printf("%d\n", array[c]);
27   return 0;
28 }
```

3.4 Sample Input/Output (Compilation, Debugging & Testing)

Input:
Enter number of elements:
5
Enter 5 integers:
99
10
56
45
7
Output:
Sorted list in ascending order:
7
10
45
56
99

4 Bubble Sort on Linked List

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

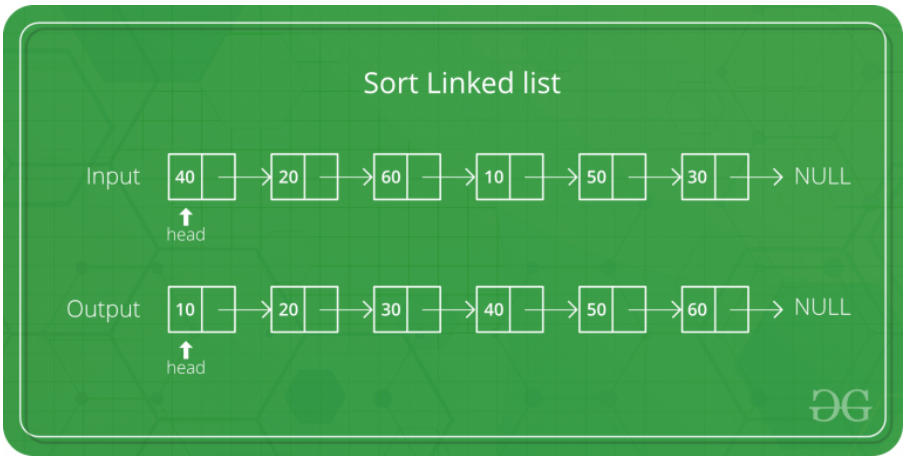


Figure 1: A Singly Sort Linked List

4.1 Implementation in C

```
1
2  // C program to sort Linked List
3  // using Bubble Sort
4  // by swapping nodes
5
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  /* structure for a node */
10 struct Node {
11     int data;
12     struct Node* next;
13 } Node;
14
15 /*Function to swap the nodes */
16 struct Node* swap(struct Node* ptr1, struct Node* ptr2)
17 {
18     struct Node* tmp = ptr2->next;
19     ptr2->next = ptr1;
20     ptr1->next = tmp;
21     return ptr2;
22 }
23
24 /* Function to sort the list */
25 int bubbleSort(struct Node** head, int count)
26 {
27     struct Node** h;
28     int i, j, swapped;
29
30     for (i = 0; i <= count; i++) {
31
32         h = head;
33         swapped = 0;
```

```
34
35          for (j = 0; j < count - i - 1; j++) {
36
37              struct Node* p1 = *h;
38              struct Node* p2 = p1->next;
39
40              if (p1->data > p2->data) {
41
42                  /* update the link after swapping */
43                  *h = swap(p1, p2);
44                  swapped = 1;
45              }
46
47              h = &(*h)->next;
48          }
49
50          /* break if the loop ended without any swap */
51          if (swapped == 0)
52              break;
53      }
54  }
55
56  /* Function to print the list */
57  void printList(struct Node* n)
58  {
59      while (n != NULL) {
60          printf("%d -> ", n->data);
61          n = n->next;
62      }
63      printf("\n");
64  }
65
66  /* Function to insert a struct Node
67     at the beginning of a linked list */
68  void insertAtTheBegin(struct Node** start_ref, int data)
69  {
70      struct Node* ptr1
71          = (struct Node*)malloc(sizeof(struct Node));
72
73      ptr1->data = data;
74      ptr1->next = *start_ref;
75      *start_ref = ptr1;
76  }
77
78  // Driver Code
79  int main()
80  {
81      int arr[] = { 78, 20, 10, 32, 1, 5 };
82      int list_size, i;
83
84      /* start with empty linked list */
85      struct Node* start = NULL;
86      list_size = sizeof(arr) / sizeof(arr[0]);
87
88      /* Create linked list from the array arr[] */
89      for (i = 0; i < list_size; i++)
90          insertAtTheBegin(&start, arr[i]);
91
```

```
92      /* print list before sorting */
93      printf("Linked list before sorting\n");
94      printList(start);
95
96      /* sort the linked list */
97      bubbleSort(&start, list_size);
98
99      /* print list after sorting */
100     printf("Linked list after sorting\n");
101     printList(start);
102
103     return 0;
104 }
```

4.2 Sample Input/Output (Compilation, Debugging & Testing)

Output:
Linked list before sorting
5 -> 1 -> 32 -> 10 -> 20 -> 78 ->
Linked list after sorting
1 -> 5 -> 10 -> 20 -> 32 -> 78 ->

5 Discussion & Conclusion

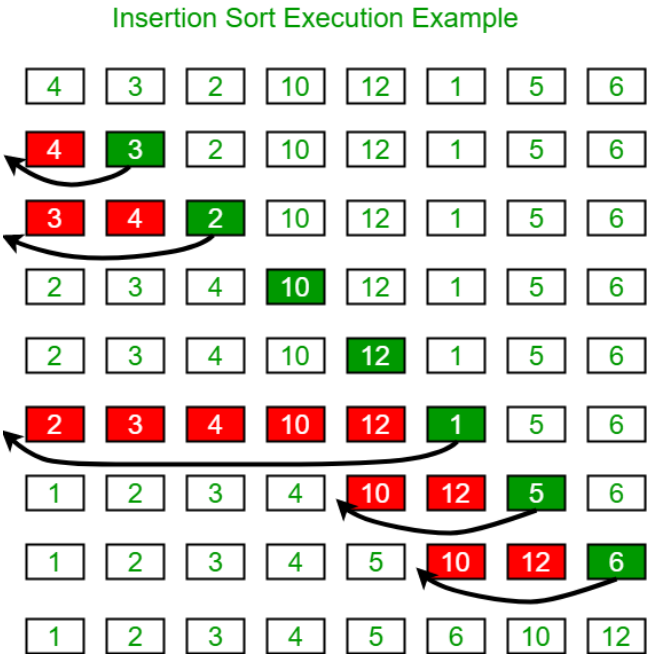
Based on the focused objective(s) to understand about the sorting algorithms, the additional lab exercise made me more confident towards the fulfilment of the objectives(s).

6 Lab Task (Please implement yourself and show the output to the instructor)

1. Implement Insertion Sort algorithm using Arrays

6.1 Problem analysis

Insertion sort is the sorting mechanism where the sorted array is built having one item at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order. The analogy can be understood from the style we arrange a deck of cards. This sort works on the principle of inserting an element at a particular position, hence the name Insertion Sort.



6.2 Algorithm

Algorithm 2: Insertion Sort Algorithm

Input: Array *A[ ]*
Output: Sorted Array *A[ ]*

```
/* Algorithm for Insertion Sort                                    */
1 i ← 1
2 while i < length(A) do
3 |   while (j > 0 and A[j-1] > A[j] ) do
4 |   |   swap A[j] and A[j-1]
5 |   |   j ← j - 1
6 |   end
7 |   i ← i + 1
8 end
```

7 Lab Exercise (Submit as a report)

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

8 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.