

**Bahir Dar University**

**Bahir Dar Institute of Technology**

**Faculty of Computing**

**Computer Science Department**

**Data Structures and Algorithms Assignment**

**Name: ID No**

1. Mengistu Getie --------------- *BDU1305912*
2. Hailemichael Adane ------ *BDU1305942*
3. Tsehaynesh Alemnew ------ *BDU1306062*
4. Hafiz Temam ---------------- *BDU1306058*
5. Muhamed Seid -------------- *BDU1305982*
6. Seid Issa ----------------------BDU1306072
7. Kirubel

**Submitted to: Melkamu**

**Mar 2023**

Acknowledgement

# List of Acronyms

SMS: School Management System

# Table of Contents

Table of Contents

[List of Acronyms 1](#_Toc130911796)

[Table of Contents 1](#_Toc130911797)

[Summary 2](#_Toc130911798)

[**1.** **Problem Description** 4](#_Toc130911799)

[2. Problem Description 4](#_Toc130911800)

[2.1. Solution 4](#_Toc130911801)

[**4.** **References** 7](#_Toc130911802)

[**5.** **Appendices** 7](#_Toc130911803)

**List of Figures**

[Figure 1 System Admin User Case Diagram 10](#_Toc130623136)

[Figure 2 Student User Case Diagram 11](#_Toc130623137)

[Figure 3 Teacher User Case Diagram 12](#_Toc130623138)

[Figure 4 parent User Case Diagram 13](#_Toc130623139)

[Figure 5 Prototype system interface 15](#_Toc130623140)

[Figure 6 Activity diagram of the SMS 17](#_Toc130623141)

[Figure 7 Sequence diagram 18](#_Toc130623142)

[Figure 8 System Login Page 19](#_Toc130623143)

[Figure 9 Class diagram 21](#_Toc130623144)

[Figure 10 System Login Page 24](#_Toc130623145)

List of Tables

[Table 1 Actors of the system 7](#_Toc130623146)

# Summary

School Management System helps headmasters to get the most accurate information to make more effective decisions. Teachers and headmasters gain time saving administrative tools, parents gain immediate access to their children’s grades and students can track their own progress. School Management System equipped features makes it possible to generate schedules and reports in minutes and to retrieve attendance records, grade checks, report cards, transcripts, and form letters in just a few clicks.

School Management Systems helps Teachers to complete grade book, track student’s attendance, input class notes, create lesson plans and detailed reports, and communicate with other staff members, students, and parents all via e-mail. It also helps Students to access assignments and tests, and view attendance records, grades, report cards, and progress reports all online. They also can communicate through mail and forums with teachers and other students online.

School Management System (SMS) is a web enabled application model. To implement the system, schools do not need expensive hardware and software, they just need an internet connection and desktops. Our system works as a centralized database and application that schools can easily access the system from anywhere based on the login credentials. The system is a platform independent system that virtually any user can access from anywhere through a standard internet accessible system. All the backends and full functionality will be tested in the future.

1. **Problem Description**

which sorting algorithm is best to sort 1,2,3,4,5,6,8,7,9,20,30 and why?.

* 1. **Solution**

The solution to this problem depends on the direction of sort in case of simple sort algorithm. If the order of sorting is to be in ascending order, linear (simple) sort, insertion sort and selection sort all are suitable. The time complexity for traversing every element will be O(n2). For this case there will be no swapping (interchanging) of elements which makes algorithmic complexity for swapping elements become 0.

When we wish to sort in descending order, all sorting algorithms will take the worst case (i.e. O(n2)) number of comparisons. Based on the number of interchanges (swaps) the selection sort algorithm will work better than others.

# Problem Description

Show the necessary steps how to sort the following Array list using Bubble and Insertion Sort Algorithm

## Solution

Let’s take an assumption first that the order of sorting is ascending order

* + 1. **Using Bubble Sort**
* Compare the neighboring items starting from the first item to the last item and interchange
* Compare the neighboring items starting from the first to the second last item and interchange
* Repeat the above procedure until last comparison reaches the first index

Using this procedure let’s compare neighboring items from left to right. Steps can be simply visualized in Table 1 below.

Table 1 simulating bubble sort

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iteration 1:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 7 | 2 | 8 | 5 | 4 | 6 |   7 > 2? true: interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 7 | 8 | 5 | 4 | 6 |   (7 > 8) ? false : continue   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 7 | 8 | 5 | 4 | 6 |   (8 > 5) ? true: interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 7 | 5 | 8 | 4 | 6 |   (8 > 4) ? true: interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 7 | 5 | 4 | 8 | 6 |   (8 > 6) ? true: interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 7 | 5 | 4 | 6 | 8 |   End of first Iteration | Iteration 2:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 7 | 5 | 4 | 6 | 8 |   (2 > 7) ? false: continue   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 7 | 5 | 4 | 6 | 8 |   (7 > 5) ? true: interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 5 | 7 | 4 | 6 | 8 |   (7 > 4) ? true: interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 5 | 4 | 7 | 6 | 8 |   (7 > 6) ? true: interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 5 | 4 | 6 | 7 | 8 |   //End of 2nd Iteration |
| Iteration 3:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 5 | 4 | 6 | 7 | 8 |   (2 > 5) ? false: continue   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 5 | 4 | 6 | 7 | 8 |   (5 > 4) ? true: interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 6 | 7 | 8 |   (5 > 6) ? false: continue   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 6 | 7 | 8 |   // End of 3rd Iteration | Iteration 4:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 6 | 7 | 8 |   2 > 4 ? false: continue   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 6 | 7 | 8 |   4 >5 ? false: continue   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 6 | 7 | 8 |   //End of 4th Iteration  Iteration 5:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 6 | 7 | 8 |   2 > 4 ? false: continue   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 6 | 7 | 8 |   // End of 5th Iteration  // End of Sorting algorithm |

* + 1. **Using Insertion Sort**

Insertion sort divides the data into sorted and unsorted. Then it takes an element from unsorted list and finds appropriate position in the sorted list.

Pseudo Code

**procedure** **insertionSort**(**A**: **list** **of** **sortable** items)

   n = **length**(A)

   for i = 1 to n - 1 do

       j = i

       while j > 0 and A[j-1] > A[j] do

**swap**(A[j], A[j-1])

           j = j - 1

       end while

   end for

end procedure

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Initial condition:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 7 | 2 | 8 | 5 | 4 | 6 |   Take left of 7: sorted  Take right of 7: unsorted   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 7 | 2 | 8 | 5 | 4 | 6 |  |   **First pass: take 2**  (2 < 7) ? true : interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 7 | 8 | 5 | 4 | 6 |   **Second Pass: take 8**  (8 < 7) ? false: break   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 7 | 8 | 5 | 4 | 6 |   **Third Pass: take 5**  (5 < 8) ? true: interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 7 | 5 | 8 | 4 | 6 |   5 < 7 ? true: interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 5 | 7 | 8 | 4 | 6 |   5 < 2 ? false : break   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 5 | 7 | 8 | 4 | 6 | | Fourth Pass: take 4  4 < 8 ? true : interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 5 | 7 | 4 | 8 | 6 |   4 < 7 ? true : interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 5 | 4 | 7 | 8 | 6 |   4 < 5? true : interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 7 | 8 | 6 |   4 < 2? false : break   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 7 | 8 | 6 |   Fifth Pass: take 6  6< 8? true : interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 7 | 6 | 8 |   6< 7? true : interchange   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 6 | 7 | 8 |   6< 5? false : break   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 2 | 4 | 5 | 6 | 7 | 8 |   // End Iteration  // End of sorting |

1. **Problem Description**

Write a C++ linked list program that implements student ADT with properties id, name, father name, sex, CGPA, etc. The ADT has also the following operation

1. Add operation that add a student wherever required
2. Delete operation that deletes an academic dismissal student ie. If his/her CGPA is less than 1.75;
3. Search operation that accepts student name then search and display his/her information.
4. Display operation that displays student information in the linked list.
5. Sort operation that sort students linked list by their name in ascending order (use either of searching technique
   1. **Solution**

The abstract data type described, can be implemented by both single and double linked list data structure. The linked list data structure is again implemented by using self-referential data structure. For better readability our source code is designed with header file

1. **References**
2. **Appendices**