COMP605 - ASSIGNMENT

LEARNING OUTCOMES

- Analyse and implement arrays, lists, binary search trees, balanced trees, and hash table.
- Write and analyse common algorithms including sorting and searching.
- Apply data structures and algorithms to software applications.

COURSE CONTENT

The course work tasks aim to practice the following topics:

- Big O analysis
- Arrays and simple algorithms
- Singly and doubly linked lists
  - Basic operations
  - Advanced functions
- Stack and Queues abstract data types
  - Implementation using arrays
  - Implementation using linked lists
  - Examples of using stack and queues in solving problems
- The use of data structures in modular programming
- Binary search trees
  - Standard operations: create, insert, delete and search operations
  - Useful applications of binary search trees
  - Traversing and advanced functions
- Sorting and searching algorithms e.g. binary search, bubble sort, quick sort or heap sort
- Hash tables
  - Insert, delete and search operations
  - The use and rationale of hash tables in ubiquitous applications
- Examples of exponential algorithms
- Balanced trees (e.g. 2-3 trees, B-trees and AVL trees)

MARKING PROCESS

STEP 1 Copy all your ***.cs*** files into the provided upload textbox on Moodle.

STEP 2 Demonstrate your solution to your tutor during class. You may have to explain to your tutor how your program works and be prepared to answer questions with regards to your solution. Note, your solutions must follow best programming practices.

MARKING CRITERIA AND MARK ALLOCATION

To achieve the allocated mark on each question, your solution must be complete and function correctly according to the following:

- The question requirement description.
- The sample run outputs (if any).
- Your program must follow the Programming Best Practices – you can find the document on Moodle.

TASKS TO COMPLETE

Note, there is a marking guide at the end of this document.

TASK 1

- Learning outcomes: 1, 3
- Relevant topics: random numbers, unique numbers, arrays operations
- Resources: self-learning tutorials videos, lecture tutorials, reference material on Moodle, recommended textbooks
- Moodle section reference: week 1

- In the sample output below are five unique random numbers. Use them to create a text file named "data.txt". Store the numbers in the file, **one number per line**.
- Complete the function in the code template given below that
  - Reads the numbers from the text file, saves them into an integer array, and
  - Displays the numbers to the screen.
  - Calculates and displays the minimum number and the **index** of the minimum number.

Sample run output:

The numbers in the file are:
19 21 13 14 15

The minimum number is 13
The minimum number index is 2

Code template:

```
class Program
{
    static void Main()
    {
        Task1();
    }
    static void Task1()
    {
      //complete function here
    }
}
```

TASK 2

- Learning outcomes: 1, 3
- Relevant topics: array segregation, list operations, pseudo code, searching algorithms
- Resources: self-learning tutorials videos, lecture tutorials, reference material on Moodle, recommended textbooks
- Moodle section reference: week 2

- The code template below calls GenerateUniqueNumbers (covered in previous week) to create an integer array. Note, copy the function from last week and put it in your solution.
- Complete the function CountPrime(int[] a) that counts and displays the number of prime numbers in an array. Hint:
  - Call the function GenerateUniqueNumber to get an array of 5 unique random numbers and pass the array to the function CountPrime as parameter.
  - Make use of the IsPrime function from the lecture, i.e., copy the function and put it in your solution.
- Complete the function SegregateEvenOdd(int[] a, int left, int right) that segregates odd and even numbers in an integer array, even numbers first, followed by odd ones.
- Complete the pseudo code function below that describes using linear search to find if the target x exists in the array a. Print "Found" or "Not found" respectively.

```
function LinearSearch (int array a, target x)
{
        //complete pseudo code here
}
```

Sample run output:

67 19 68 43 54
Count of prime numbers: 3

Array elements after segregation
54 68 19 43 67

Code template:

```
class Program
{
    static void Main()
    {
        Task2();
    }
    static void Task2()
    {
        /*
                Task 2 pseudo code
                function (int array a, target x)
                {

                }
        */
        int[] a = GenerateUniqueNumbers(5);
        //code to display array original content
        CountPrime(a);
```

```
            Segregate(a, 0, a.Length - 1);
            WriteLine($"Array elements after segregation");
            //code to display array a after segregation
        }
        static void SegregateEvenOdd(int[] arr, int left, int right)
        {
          //complete function here
        }
        static void CountPrime(int[] a)
        {
          //complete function here
        }
}
```

TASK 3

- Learning outcomes: 1, 3
- Relevant topics: singly linked list
- Resources: self-learning tutorials videos, lecture tutorials, reference material on Moodle, recommended textbooks
- Moodle section reference: week 3

- The code template below calls GenerateUniqueNumbers (covered in previous weeks) and inserts them into an SLL.
- Complete the function that displays all nodes of an SLL.
- Complete the function that calculates and **returns** the length of an SLL. Hint, traverse the entire SLL and use a counter variable to count how many nodes have been iterated.
- Complete the function that displays the middle node of an SLL. Hint, get the SLL length by calling the previous function, then travel to half-way of the SLL to find the middle node.
- Describe (using either your own words or pseudo code) the algorithm you have used for finding the middle node of an SLL.

Sample run output:

28 25 49 95 47 79 72 20 12 44
SLL length: 10
Middle node: 79

The algorithm to find the middle node of an SLL begins with <write your description here>.

Code template:

```
class Program
{
    static void Main()
    {
      Task3();
    }
    static void Task3()
    {
        SLL sll = new SLL();
        int[] a = GenerateUniqueNumbers(10);
        foreach (int i in a)
        {
            sll.AddFirst(i);
        }
        WriteLine("SLL nodes");
        PrintSLL(sll.Head);
        WriteLine($"SLL length: {GetSLLSize(sll.Head)}");
        FindSLLMiddle(sll.Head);
    }
    static void PrintSLL(SLLNode node)
    {
        // complete function code
    }
    static int GetSLLSize(SLLNode node)
    {
       // complete function code
    }
```

```
    static void FindSLLMiddle(SLLNode node)
    {
        // complete function code
    }

}
class SLL
{
    // get the class code from lecture

}
class SLLNode
{
    // get the class code from lecture
}
```

TASK 4

- Learning outcomes: 1, 3
- Relevant topics: Doubly linked list
- Resources: self-learning tutorials videos, lecture tutorials, reference material on Moodle, recommended textbooks
- Moodle section reference: Array and List

- The code template below calls GenerateUniqueNumbers (covered in previous weeks) and inserts them into a DLL.
- Complete the function that prints the numbers in reverse order (from tail to head).
- Complete the function that prints the middle node of a DLL.
- (Optional challenge) You can keep using the half-length approach from last week to find the middle node, but can you think of another strategy of doing it instead, given the list is DLL (which allows you to travel backwards)?

Sample run output:

DLL nodes in reverse order
72 20 12 44 79 28 25 49 95 47
Middle node: 79

Code template:

```
class Program
{
    static void Main()
    {
        Task4();
    }
    static void Task4()
    {
        DLL dll = new DLL();
        int[] a = GenerateUniqueNumbers(10);
        foreach (int i in a)
        {
            dll.AddLast(i);
        }
        WriteLine("DLL nodes in reverse order");
        PrintBackwards(dll);
        FindMiddle(dll);
    }

    static void PrintBackwards(DLL dll)
    {
      // complete function code
    }
    static void FindMiddle(DLL dll)
    {
      // complete function code
    }
```

```
}
class DLL
{
    // get the class code from lecture
}
class DLLNode
{
    // get the class code from lecture
}
```

TASK 5

- Learning outcomes: 1, 3
- Relevant topics: Stack, queue
- Resources: self-learning tutorials videos, lecture tutorials, reference material on Moodle, recommended textbooks
- Moodle section reference: week 5

- Complete the function in the code template that reads three integers from the user. The function then displays the user entered numbers in the reverse order in which they were read. Hint,
    - Save the user entered numbers into a stack (Note, use either C# built-in Stack class, or the MyStack class from the lecture), then
    - Print the content of the stack. To print stack content, use a while loop controlled by the stack Count property.
- In your own words, provide definitions for the following operations:
    - Push
    - Pop
    - Enqueue
    - Dequeue

Sample run output:

Enter 3 numbers:
Enter a number: 3
Enter a number: 4
Enter a number: 5
The entered numbers in reverse order:
5 4 3

Code template:

```
class Program
{
    static void Main()
    {
        Task5();
    }
    static void Task5()
    {
      //complete function here
    }
}
```

TASK 6

- Learning outcomes: 1, 3
- Relevant topics: using Big-O notation to analyse algorithm efficiency
- Resources: self-learning tutorials videos, lecture tutorials, reference material on Moodle, recommended textbooks
- Moodle section reference: week 6

- Use Big-O notation to estimate the time complexity of each of the following algorithms (presented as code snippet). Note, variable n is a large positive number.
- Your answer should include
    - Big-O estimate
    - Analysis to justify the above estimate

1.

```
for (int k = n; k > 0; k--)
{
    for (int i = 30; i > 0; i--)
    {
        for (int j = n; j > 0; j--)
        {
            WriteLine(“*”);
        }
    }
}
```

2.

```
int count = n;
while (count > 1)
{
    count = count / 2;
}
```

3.

```
for (int i = 10; i > 0; i--)
{
    WriteLine(“*”);
}
for (int i = n; i > 0; i--)
{
     for (int j = n; j > 0; j--)
     {
         WriteLine(“*”);
     }
}
```

TASK 7

- Learning outcomes: 1, 3
- Relevant topics: Binary search tree
- Resources: self-learning tutorials videos, lecture tutorials, reference material on Moodle, recommended textbooks
- Moodle section reference: week 7

- The code template below includes the class Program, BSTNode, and BST. The function Task7() first creates a BST instance, then inserts the numbers 3,2,5,1,7,9,6 into the BST tree. It then calls the functions.
- Complete the function PrintDescending that displays the keys in order from the largest to the smallest.
- Complete the function PrintAscending that displays the keys in the order from the smallest to the largest.
- Complete the function DisplayPrime that uses recursion to traverse and display all prime numbers in the BST. Hint,
    - Instead of displaying every node (like the Preorder() function does), this function should put an if condition on the Write statement and only print a node if its Key is prime.
    - Make use of the IsPrime function from previous weeks.
- (Optional challenge) Complete the function GetBSTSize that that uses recursion to calculate and return the size of BST (number of nodes). Hint, use the CountPrimeNodes function from the lecture as guide.

Sample run output:

BST in ascending order: 1 2 3 5 6 7 9
BST in descending order: 9 7 6 5 3 2 1
Prime numbers in BST: 3 2 5 7
BST size: 7

Code template:

```
class Program
{
   static void Main()
   {
        Task7();
   }
    static void Task7()
    {
       BST bst = new BST();
       new int[] {3, 2, 5, 1, 7, 9, 6}.ToList().ForEach(i => bst.Insert(i));
       if (bst.Root != null)
       {
          Write("\nBST in ascending order: ");
          PrintAscending(bst.Root);
          Write("\nBST in descending order: ");
          PrintDescending(bst.Root);
          Write("\nPrime numbers in BST: ");
          DisplayPrime(bst.Root);
          WriteLine($"\nBST size: {GetBSTSize(bst.Root)}");
       }
    }
```

```
    static void PrintDescending(BSTNode root)
    {
       //Complete function here
    }
    static void PrintAscending(BSTNode root)
    {
       //Complete function here
    }
    static void DisplayPrime(BSTNode root)
    {
       //Complete function here
    }
    static int GetBSTSize(BSTNode root)
    {
       //Complete function here
    }
    static bool IsPrime(int n)
    {
       // get code for this function from lecture
    }

}
class BST
{
    // get code for this class from lecture
}
class BSTNode
{
    // get code for this class from lecture
}
```

TASK 8

- Learning outcomes: 1, 3
- Relevant topics: BST, AVL, B-tree
- Resources: self-learning tutorials videos, lecture tutorials, reference material on Moodle, recommended textbooks
- Moodle section reference: week 8

- In your own words, define BST balance factor.
- In your own words, describe the rules of 3 order B-tree.
- Insert the numbers 10 12 8 6 5 into an AVL tree.
    - Draw the resulting tree before rebalancing
    - Draw the tree after rebalancing
    - What rotation was performed in rebalancing the tree?

TASK 9

- Learning outcomes: 1, 3
- Relevant topics: Hash table
- Resources: self-learning tutorials videos, lecture tutorials, reference material on Moodle, recommended textbooks
- Moodle section reference: week 9

- In your own words, describe how linear probing resolves collision.
- In the code template below, complete the function Task9() with the requirements below.
  - Create a string variable and use this text as data input: *data structures and algorithms is about data structures and algorithms. data structures is about data structures. algorithms is the about algorithms. there are good algorithms and not so good algorithms. good algorithms are good good, not so good ones are not so good.*
  - Split the string variable into an array of words.
  - Use the array of words to populate a dictionary, word as key and word frequency as value. Insert each word into a dictionary as key. If the dictionary doesn't contain the word, add the word as key and set the value as 1. If the dictionary already contains the word, increment the value by 1.
- Complete the function ProcessWords takes a dictionary as parameter (passed from function Task9).
  - Iterate the dictionary and display the words along with their corresponding frequencies.
  - Calculate the most and least frequent words in the dictionary. Display the words along with their frequencies. Hint, first find out the most and least frequency values, in the case of the sample text, 7 and 1. Then use the frequency values to search the words that have the two frequency values and collect them using either string concatenation or a list. Caution, there may be more than one word that share the same most and least word counts.
- (Optional challenge) Complete the function FindLongShortWords that finds and displays the longest and shortest words with their frequencies. Again, there may be duplicates. Hint, use the ProcessWords function as a guide.

Sample run output:

```
data 4
structures 4
and 3
algorithms 7
is 3
about 3
the 1
there 1
are 3
good 7
not 3
so 3
ones 1

Most and least frequent words with frequency:
algorithms good  7
the there ones  1

Longest words with frequency:
```

structures 4
algorithms 7

Shortest words frequency:
is 3
so 3

Code template:

```
class Program
{
    static void Main()
    {
        Task9();
    }
    static void Task9()
    {
      //complete code here to generate a dictionary “dict”
      ProcessWords(dict); //calls function ProcessWords and passes dict
    }
    static void ProcessWords(Dictionary<string,int> dict)
    {
      //complete function here
    }
    static void FindLongShortWords(Dictionary<string,int> dict)
    {
      //complete function here
    }

}
```

TASK 10

- Learning outcomes: 1, 3
- Relevant topics: Sorting algorithms
- Resources: self-learning tutorials videos, lecture tutorials, reference material on Moodle, recommended textbooks
- Moodle section reference: week 10

- Given an array 21, 30, 11, 34, 20, 56, use diagrams to illustrate the step-by-step sorting process using each of the following sorting algorithms.
    - Insertion sort (what the array looks like after each pass)
    - Merge sort (a diamond shape diagram showing each split and each merge)
    - Quick sort (use the first element as pivot, and show what the array looks like after each partition)

☐ Moodle Submission Name: Score:

COMP605 ASSIGNMENT MARKING GUIDE (TOTAL MARKS: 40)

TASK 1 (3 marks)	
Read file and save into an integer array	1
Display array	1
Calculate and display min and min index	1

TASK 2 (3 marks)	
CountPrime function	1
SegregateEvenOdd function	1
Linear search pseudo code	1

TASK 3 (4 marks)	
PrintAll funciton	1
GetSLLSize function	1
FindMiddle function	1
Describe FindMiddle algorithm	1

TASK 4 (2 marks)	
PrintBackwards function	1
FindMiddle function	1

TASK 5 (2 marks)	
Task5 function	1
Definitions of Push, Pop, Enqueue, Dequeue	1

TASK 6 (3 marks)	
Big-O notations for three code snippets	1.5
Analyses for the three notations	1.5

TASK 7 (4 marks)	
PrintAscending function	1
PrintDescending function	1
DisplayPrime function	2

TASK 8 (4 marks)	
Definition of balance factor	0.5
Rules of 3 order B-tree	1
AVL tree before rebalancing	1
AVL tree after rebalancing	1
Rotation type	0.5

TASK 9 (7 marks)	
Population dictionary of words and word counts (in function Task9)	2
Display words along with word counts (in function ProcessWords)	2
Display the most and least frequent words and word counts (in function ProcessWords)	3

TASK 10 (8 marks)	
Illustrate process of insertion sort	2
Illustrate process of merge sort	3
Illustrate process of quick sort	3