

**Data Structures & Algorithms (DSA)**

Year 2/3 (2020/21), Semester 4/6

## SCHOOL OF INFOCOMM TECHNOLOGY

Diploma in Cybersecurity & Digital Forensics

Diploma in Information Technology

**TEST 1 Sample – SOLUTION DOCUMENT**

INSTRUCTIONS TO CANDIDATES:

1. Write your Student Number, Name and Module Group CLEARLY in the boxes provided below.
2. Provide your answers to the questions in the Test 1 paper in this document.
3. Save this file as "Test1 – s1234567 Solution.docx" where s1234567 is your student number.
4. Map to network drive: [**\\ictspace.ict.np.edu.sg\DSATest1\**](file:///\\ictspace.ict.np.edu.sg\DSATest1\)
5. Copy this solution file into the network drive.

**ictspace.ict.np.edu.sg > DSATest1 > group > studentID**

|  |  |
| --- | --- |
| **Student Number:** | **Seat Number:** |
| **Student Name:** | **Module Group:** |

**GRADE**

There are 3 questions. Answer ALL questions (100 marks).

Write your solutions to the questions in the space allocated for each question.

Question 1 – Solution (40 marks)

|  |  |
| --- | --- |
| (a) | void List::sortedInsert(ItemType newItem)  {  // create a new node to store the data item (2 marks)  Node\* newNode = new Node;  newNode->item = newItem;  newNode->next = NULL;  // consider if List is empty  if (size == 0 || firstNode->item >= newItem) (3 marks)  // isEmpty() 🡨 look in the file whether this function is given  newNode->next = firstNode; // to handle case where list not empty  firstNode = newNode; (2 marks)  // consider if List got something  else  { // find location to do insert  Node\* temp = firstNode; (5 marks)  while (temp->next ->item < newItem && temp->next != NULL)  // traverse to the correct position  temp = temp->next;  // insert (3 marks)  newNode->next = temp->next; //1  temp->next = newNode; //2  }  size++;  }  // any other function that you write, you need to copy here. |
|  | (15 marks) |
| (b) | // this function must be given  // it makes use of the helper function sortedMerge that has the firstNodes // as the parameters.  void List::sortedMerge(List list1, List list2)  {  firstNode = sortedMerge(list1.firstNode, list2.firstNode);  }  // you are asked to write this function  List::Node\* List::sortedMerge(Node\*& a, Node\*& b)  { // a and b are also Lists.  Node\* result = NULL; // result is a List  } |
|  | (15 marks) |
| (c) | For SortedInsert(), the best case is when the list is originally empty or item is inserted as the first element, in this case it takes constant time O(1).  Worst case is when the item is to be inserted at the end of all existing elements and traverse down the entire list, in this case it takes O(n).  In general, it takes O(n). |
|  | (10 marks) |

Question 2 – Solution (35 marks)

|  |  |
| --- | --- |
| (a) | void registerCustomer(Queue& serviceQueue, int& queueNumber)  {  string name;  cout << "Enter name : ";  cin >> name;  Customer c(queueNumber, name); (4 marks)  serviceQueue.enqueue(c); (3 marks)  queueNumber++; (3 marks)  } |
|  | (10 marks) |
| (b) | void nextCustomer(Queue& serviceQueue)  {  Customer c;  serviceQueue.dequeue(c); (3 marks)  cout << c.getQueueNumber() << endl; (2 marks)  } |
|  | (5 marks) |
| (c) | void displayCount(Queue& serviceQueue)  {  int count = 0;  Queue tempQ;  Customer c;  while (!serviceQueue.isEmpty())  {  count++;  serviceQueue.dequeue(c);  tempQ.enqueue(c);  }  serviceQueue = tempQ; // restore the serviceQueue  cout << "Length of the queue is " << count << endl;  }  void displayCount(Queue& serviceQueue)  {  int count = 0;  if (!serviceQueue.isEmpty())  {  count++;  Customer c;  serviceQueue.dequeue(c);  int currentQueueNumber = c.getQueueNumber();  // add back to the queue  serviceQueue.enqueue(c);  serviceQueue.getFront(c);  while (c.getQueueNumber() != currentQueueNumber)  {  count++;  serviceQueue.dequeue(c); // remove from queue  serviceQueue.enqueue(c); // add to back of queue  serviceQueue.getFront(c); // get the next customer  }  }  cout << "Length of the queue is " << count << endl;  } |
|  | (15 marks) |
| (d) | To allow removal of later customers near the end of the queue, we add an operation to dequeuer from the back.  For those customers who need to be processed more quickly than others due to special conditions, they can be added to the front of the queue.  This results in a double ended queue structure where additions/deletions can be done at both ends.  This is called a deque queue. |
|  | (5 marks) |

Question 3 – Solution (25 marks)

|  |  |
| --- | --- |
| (a) | Iteration of GCD(24, 54) is 6  1st iteration x = 24 y = 54 x%y = 24  2nd iteration x = 54 y = 24 x%y = 6  3rd iteration x = 24 y = 6 x%y = 0 (found) |
|  | (5 marks) |
| (b) | int gcd(int x, int y)  {  // base case  if ( x% y == 0 )  return y;  // recursive step  else  return gcd(y, x%y);  } |
|  | (10 marks) |
| (c) | int gcd(int x, int y)  {  int result = x%y ;  while (result != 0)  {  x = y;  y = result ; // cannot have y = x%y because this is y= y%y  result = x%y ;  }  return y;  } |
|  | (10 marks) |

**– End of Document –**