

**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& item)  {  // create a new node  Node\* newNode = new Node;  newNode->item = item;  newNode->next = NULL;  // consider for the case when head is empty  if (firstNode == NULL || firstNode->item >= newNode->item)  { // Best case for (c)  newNode->next = firstNode;  firstNode = newNode;  }  // otherwise, list is not empty  else  { // Worst case for (c)  // locate the node before the point of insertion  Node\* temp = firstNode;  while (temp->next != NULL &&  temp->next->item < newNode->item)  temp = temp->next;  // add the item  newNode->next = temp->next; // 1  temp->next = newNode; // 2  }  } |
|  | (15 marks) |
| (b) | **THIS WILL NOT COME OUT FOR TEST1**  Node\* List::sortedMerge(Node\* a, Node\* b)  {  // create new list  Node\* result = NULL;  Node\*\* lastPtrRef = &result;  // temp pointer to the last result pointer    // repeat until one of the list a or b is empty  while (1) { // infinite loop, and use break to get out of loop  // if a is empty, attach the rest of b to the new list  if (a == NULL) {  \*lastPtrRef = b;  break;  }  // if b is empty, attach the rest of a to the new list  else if (b == NULL) {  \*lstPtrRef = a;  break;  }  // if a->item <= b->item,  if (a->item <= b->item) {  // move a node to new list  moveNode(lastPtrRef, &a);  // else  else  // move b node to new list  moveNode(lastPrtRef, &b);  // advance to point to the next ".next"  lastPtrRef = &((\*lastPtrRef)->next);  }  return result;  }  void List::moveNode(Node\*\* dest, Node\*\* src)  {  /\* the front source node \*/  Node\* newNode = \*src;  if (newNode != NULL)  {  \*src = newNode->next;  /\* Link the old dest off the new node \*/  newNode->next = \*dest;  //dest->next = newNode;  /\* Move dest to point to the new node \*/  \*dest = newNode;  }  } |
|  | (15 marks) |
| (c) | For SortedInsert(), the best case time performance is when the list is originally empty, or the item to be inserted is less than the first element in the list (sorted in ascending order), in which case it only take fixed set of instructions that take constant time => O(1).  Worst case for SortedInsert() is when the item to be inserted is bigger than all existing elements in the list, in which case traversal down the list is needed  => O(n).  For SortedMerge(), when either of the list is empty; then only traversal down the non-empty list is needed, as new items are created in the resultant list. => O(n).  When both list are non-empty, then traversal down both lists are required to obtain the items of both list. O(n) + O(n) => 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)  }  ALTERNATIVE SOLUTION  serviceQueue.getFront(c); ß need to also supply this function if not there  serviceQueue.dequeue(); |
|  | (5 marks) |
| (c) | void displayCount(Queue& serviceQueue)  {  int count = 0;  Queue tempQ;  Customer c;  while (!serviceQueue.isEmpty())  {  count++;  serviceQueue.dequeue(c);  tempQ.enqueue(c);  }  while (!tempQ.isEmpty())  {  tempQ.dequeue(c);  serviceQueue.enqueue(c);  }  cout << "Length of the queue is " << count << endl;  }  ALTERNATIVE SOLUTION  void displayCount(Queue& serviceQueue)  {  int count = 0;  if (!serviceQueue.isEmpty())  {  count++;  Customer c;  serviceQueue.dequeue(c);  int firstQueueNumber = c.getQueueNumber();  // add back to the queue  serviceQueue.enqueue(c);  serviceQueue.getFront(c);  while (c.getQueueNumber() != firstQueueNumber)  {  count++;  serviceQueue.dequeue(c); // remove from queue  serviceQueue.enqueu(c); // add to back of queue  serviceQueue.getFront(c); // get 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, need an operation to remove customers from the back eg dequeueBack()  For those customers who need to be processed more quickly than others due to special condition, need an operation to add to the front of the queue eg. enqueueFront()  We need a double ended queue structure that can add/remove at both ends.  This is called a deque (pronounced deck) 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 (found) x%y = 0 (stop) |
|  | (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 x%y because x is already y  result = x%y ;  }  return y;  } |
|  | (10 marks) |

**– End of Document –**