

**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 – 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: S10198298** | **Seat Number:** |
| **Student Name: Lee Quan Sheng** | **Module Group: P04** |

**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 (50 marks)

|  |  |
| --- | --- |
| (a) | ItemType List::getMiddle()  {  Node\* currentNode = firstNode;  int count = size / 2;  for (int i = 0; i < count; i++)  {  currentNode = currentNode->next;  }  return currentNode->item;  } |
|  | (10 marks) |
| (b) | ItemType List::getMiddleR()  {  int count = size / 2;  return getMiddleR2(firstNode, count);  }  ItemType List::getMiddleR2(Node\* currNode, int count) {  if (count == 0) {  return currNode->item;  }  else {  return getMiddleR2(currNode->next, count - 1 );  }  } |
|  | (30 marks) |
| (c) | The time complexity for 1(a) is O(n). The time performance for 1(a) will take the number of **n** loops to reach to the middle. In this case the time take is O(n)  The time complexity for 1(b) is O(n). The time performance for 1(b) will take the number of n recursion to reach the base case. In this case time complexity is O(n) |
|  | (10 marks) |

Question 2 – Solution (25 marks)

|  |  |
| --- | --- |
| (a) | The diagram from the right is the TOP  Stack – 10  Stack – 10, 4  Stack – 10, 6, 4  Stack – 10, 6, 4, 3  Stack – 10, 6, 5, 4, 3 |
|  | (10 marks) |
| (b) | bool Stack::push(ItemType item)  {  bool success = top < MAX\_SIZE-1;  if(success) {  for (int i = top; i >= 0; i--)  {  if (items[i] < item) {  for (int a = top; a >= i + 1 ; a--)  {  items[a + 1] = items[a];  }  top++;  items[i + 1] = item;  return true;  //break  }  }  top++;  items[top] = item;  return true;  }  else {  return false;  }  } |
|  | (15 marks) |

Question 3 – Solution (25 marks)

|  |  |
| --- | --- |
| (a) | Hash table  Each of it refers to - **Key(position)**  0 refers to NULL  Index  0,  0,  0,  the (1) - > fox (4) -> the(7) -> dog (9)  over (6) -> lazy(8)  quick(2) -> brown(3) -> jumps(5)  0,… |
|  | (10 marks) |
| (b) | The suitable has function would be string.count() power by its position % MAX\_SIZE. This is to prevent any size to go beyond MAX\_SIZE |
|  | (5 marks) |
| (c) | int Dictionary::hash(KeyType key)  {  return key.size() % MAX\_SIZE;  } |
|  | (5 marks) |
| (d) | // prints the length of each chain  void Dictionary::printChainLength()  {  for (int i = 0; i < MAX\_SIZE; i++)  {  if (items[i] == NULL) {  cout << "Pos " << i << "Size : " << "0" << endl;  }  else {  int count = 0;  Node\* firstNode = items[i];  while (firstNode != NULL) {  count++;  firstNode = firstNode->next;  }  cout << "Pos " << i << "Size : " << count << endl;  }  }  } |
|  | (5 marks) |

**– End of Document –**