COMP6008 Data Structures and Algorithms

Assessment 1

Submitted By

Ramon Pacheco

Student ID

24575333

Module One Programming Tasks

Task 1: Lexicographical order

1. Solution code (compulsory)

|  |
| --- |
| import java.util.\*;  class SortedTupleList {      public static class Tuple {          int x, y;          public Tuple(int x, int y) {              this.x = x;              this.y = y;          }          boolean equal(Tuple other) {              if (this.x == other.x && this.y == other.y)                  return true;              else                  return false;          }          public boolean greaterThan(Tuple other) {              if (this.x > other.x)                  return true;              if (this.x == other.x && this.y > other.y)                  return true;              return false;          }      }      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);          List<Tuple> list = new ArrayList<>();          while (true) {              System.out.print("Enter a tuple (a, b): ");              int a = scanner.nextInt();              int b = scanner.nextInt();              if (a == -1 && b == -1)                  break;              Tuple newTuple = new Tuple(a, b);              insert(list, newTuple);          }          System.out.println("Sorted tuples:");          for (Tuple t : list) {              System.out.println("\t(" + t.x + ", " + t.y + ")");          }          scanner.close();      }      public static void insert(List<Tuple> list, Tuple newTuple) {          int position = list.size(); // Start from the end of the list          while (position > 0 && !newTuple.greaterThan(list.get(position - 1))) {              position--;          }          list.add(position, newTuple);      }  } |

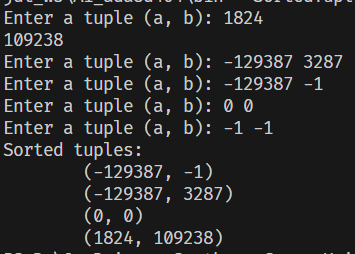
2. Testing inputs and outputs (compulsory)

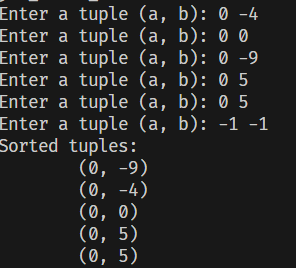
|  |  |
| --- | --- |
| Input | Output |
| Enter a tuple (a, b): 1 5  Enter a tuple (a, b): 1 4  Enter a tuple (a, b): 0 0  Enter a tuple (a, b): -1 3  Enter a tuple (a, b): -1 -1 | Sorted tuples:  (-1, 3)  (0, 0)  (1, 4)  (1, 5) |
| Enter a tuple (a, b): 1824  109238  Enter a tuple (a, b): -129387 3287  Enter a tuple (a, b): -129387 -1  Enter a tuple (a, b): 0 0  Enter a tuple (a, b): -1 -1 | Sorted tuples:  (-129387, -1)  (-129387, 3287)  (0, 0)  (1824, 109238) |
| Enter a tuple (a, b): 0 -4  Enter a tuple (a, b): 0 0  Enter a tuple (a, b): 0 -9  Enter a tuple (a, b): 0 5  Enter a tuple (a, b): 0 5  Enter a tuple (a, b): -1 -1 | Sorted tuples:  (0, -9)  (0, -4)  (0, 0)  (0, 5)  (0, 5) |

3. Screenshots of tests (compulsory)

A screen shot of a computer

Description automatically generated





4. Analysis of experiment results(if required)

5. Reflection on the use of online resources including GenAI (if any)

* What resource? (e.g., ChatGPT, Stack Overflow)
* How did you use it?
* How did it assist you in completing this task?

Task 2: Checking unbalanced opening and closing tags in HTML

1. Solution code (compulsory)

|  |
| --- |
| import java.util.\*;  import java.io.\*;  public class StackTesting {      String path = "";      public StackTesting(String path) {          this.path = path;      }      public void tagCheck() {          Stack<String> symStack = new Stack<>();          try {              String line = "";              BufferedReader br = new BufferedReader(new FileReader(path));              while ((line = br.readLine()) != null) {                  String[] tokens = line.split(" ");                  for (String e : tokens) {                      if (e.charAt(0) == '<' && e.charAt(1) != '/')                          symStack.push(e);                        else if (e.charAt(0) == '<' && e.charAt(1) == '/') {                          if (symStack.empty())                              System.err.println("Tags being closed before they are being opened.");                          else                              symStack.pop();                      }                  }              }              br.close();          } catch (FileNotFoundException e) {              System.err.println("File could not be found.\nPlease make sure the path to the file is correct.");              e.printStackTrace();          } catch (IOException e) {              e.printStackTrace();          }          System.err.println(symStack.isEmpty());      }      public static void main(String[] args) {          String path = "D:\\OneDrive - Southern Cross University\\SCU\\Term 4\\COMP6008 Data Structure and Algorithms\\A1\\Module 1\\html\_file.txt";          StackTesting std = new StackTesting(path);          std.tagCheck();      }  } |

2. Testing inputs and outputs (compulsory)

|  |  |
| --- | --- |
| Input | Output |
| <html> <head> <title> Example </title> </head> <body> <h1> Hello, world </h1> </body> </html> | True |
| <html> <head> <title> Example <title> </head> <body> <h1> Hello, world </h1> </body> </html> | False |
| <html> <head> <title> Example <title> </head> <body> <h1> Hello, world </h1> </body> <html> | False |

3. Screenshots of tests (compulsory)

4. Analysis of experiment results(if required)

5. Reflection on the use of online resources including GenAI (if any)

* What resource? (e.g., ChatGPT, Stack Overflow)
* How did you use it?
* How did it assist you in completing this task?

Task 3: Benchmark two queue implementations

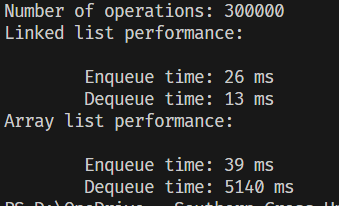
1. Solution code (compulsory)

|  |
| --- |
| import java.util.\*;  public class QueueTest {      private static final int numOperations = 300\_000;      public static void main(String[] args) {          List<Integer> qLinked = new LinkedList<>();          List<Integer> qArray = new ArrayList<>();          System.err.println("Number of operations: "+numOperations);          System.out.println("Linked list performance: \n");          benchmark(qLinked);          System.out.println("Array list performance: \n");          benchmark(qArray);      }      private static void benchmark(List<Integer> queue) {          // Enqueue benchmark          long startTime = System.currentTimeMillis();          for (int i = 0; i < numOperations; i++) {              queue.add(i);          }          long endTime = System.currentTimeMillis();          long enqueueTime = endTime - startTime;          // Dequeue benchmark          startTime = System.currentTimeMillis();          for (int i = 0; i < numOperations; i++) {              queue.remove(0);          }          endTime = System.currentTimeMillis();          long dequeueTime = endTime - startTime;          System.out.printf("\tEnqueue time: %d ms\n", enqueueTime);          System.out.printf("\tDequeue time: %d ms\n", dequeueTime);      }  } |

2. Testing inputs and outputs (compulsory)

|  |  |
| --- | --- |
| Input | Output |
| Case 1 | Number of operations: 300000  Linked list performance:  Enqueue time: 26 ms  Dequeue time: 13 ms  Array list performance:  Enqueue time: 39 ms  Dequeue time: 5140 ms |
| Case 2 | Number of operations: 300000  Linked list performance:  Enqueue time: 23 ms  Dequeue time: 13 ms  Array list performance:  Enqueue time: 52 ms  Dequeue time: 5112 ms |
| Case 3 | Number of operations: 300000  Linked list performance:  Enqueue time: 27 ms  Dequeue time: 12 ms  Array list performance:  Enqueue time: 50 ms  Dequeue time: 5183 ms |

3. Screenshots of tests (compulsory)

 A screen shot of a computer

Description automatically generated

4. Analysis of experiment results(if required)

5. Reflection on the use of online resources including GenAI (if any)

* What resource? (e.g., ChatGPT, Stack Overflow)
* How did you use it?
* How did it assist you in completing this task?

Module Two Programming Tasks

Task 4: Longest consecutive sequence

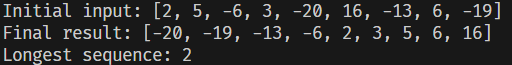
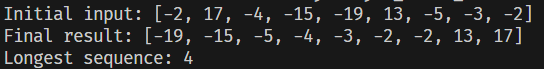
1. Solution code (compulsory)

|  |
| --- |
| import java.util.Arrays;  public class ConsecutiveSequence {      static int longestSeq = 0;      public static void main(String[] args) {          int[] X = ArrayUtils.generateRandomArray(9, 8);          System.out.println("Initial input: " + Arrays.toString(X));          QuickSort(X, 0, X.length-1);          countSeq(X);          System.out.println("Final result: " + Arrays.toString(X));          System.out.println("Longest sequence: " + longestSeq);      }      static void QuickSort(int[] list, int startIndex, int endIndex) {          if (startIndex < endIndex) {              int pivotIndex = partition(list, startIndex, endIndex);              QuickSort(list, startIndex, pivotIndex - 1);              QuickSort(list, pivotIndex + 1, endIndex);          }      }        static int partition(int[] list, int startIndex, int endIndex) {          int pivotVal = list[startIndex];          int leftMark = startIndex + 1;          int rightMark = endIndex;          while (leftMark <= rightMark) {              while (leftMark <= rightMark && list[leftMark] <= pivotVal)                  leftMark++;              while (leftMark <= rightMark && list[rightMark] >= pivotVal)                  rightMark--;              if (leftMark <= rightMark)                  swap(list, leftMark, rightMark);          }          swap(list, startIndex, rightMark);          return rightMark;      }      static void swap(int[] list, int i, int j) {          int temp = list[i];          list[i] = list[j];          list[j] = temp;      }      static void countSeq(int[] list) {          int currentSeq = 1;          for (int i = 0; i < list.length - 1; i++) {              if (list[i] + 1 == list[i + 1]) {                  currentSeq++;              } else {                  currentSeq = 1;              }              longestSeq = currentSeq > longestSeq ? currentSeq : longestSeq;          }      }  } |

2. Testing inputs and outputs (compulsory)

|  |  |
| --- | --- |
| Input | Output |
| Initial input: [2, 5, -6, 3, -20, 16, -13, 6, -19] | Longest sequence: 2 |
| Initial input: [-2, 17, -4, -15, -19, 13, -5, -3, -2] | Longest sequence: 4 |
| Initial input: [-5, 16, 8, 21, -13, -15, 14, -21, -15] | Longest sequence: 1 |

3. Screenshots of tests (compulsory)

  A number on a black background

Description automatically generated

4. Analysis of experiment results(if required)

5. Reflection on the use of online resources including GenAI (if any)

* What resource? (e.g., ChatGPT, Stack Overflow)
* How did you use it?
* How did it assist you in completing this task?

Task 5: Alternative pivot value for quicksort

1. Solution code (compulsory)

|  |
| --- |
| public class PivotPosition {      public static void main(String[] args) {          int[] testList = ArrayUtils.generateRandomArray(200000, 8);          long startTime = System.currentTimeMillis();          QuickSort(testList, 0, testList.length - 1);            long endTime = System.currentTimeMillis();          System.out.printf("Quick sort execution time: %d ms.", endTime-startTime);            startTime = System.currentTimeMillis();          QuickSortSwapping(testList, 0, testList.length - 1);            endTime = System.currentTimeMillis();          System.out.printf("\nQuick sort swapping pivot index execution time: %d ms.", endTime-startTime);      }      static void QuickSort(int[] list, int startIndex, int endIndex) {          if (startIndex < endIndex) {              int pivotIndex = partition(list, startIndex, endIndex);              QuickSort(list, startIndex, pivotIndex - 1);              QuickSort(list, pivotIndex + 1, endIndex);          }      }        static void QuickSortSwapping(int[] list, int startIndex, int endIndex) {          if (startIndex < endIndex) {              swapPivot(list, startIndex, endIndex);              int pivotIndex = partition(list, startIndex, endIndex);              QuickSort(list, startIndex, pivotIndex - 1);              QuickSort(list, pivotIndex + 1, endIndex);          }      }      static void swapPivot(int[] list, int startIndex, int endIndex) {          int midIndex = (startIndex + endIndex) / 2 + 1;          if (list[startIndex] <= list[midIndex] && list[midIndex] <= list[endIndex]) {              swap(list, startIndex, midIndex);          } else if (list[startIndex] <= list[endIndex] && list[midIndex] <= list[endIndex]) {              swap(list, startIndex, endIndex);          }      }      static int partition(int[] list, int startIndex, int endIndex) {          int pivotVal = list[startIndex];          int leftMark = startIndex + 1;          int rightMark = endIndex;          while (leftMark <= rightMark) {              while (leftMark <= rightMark && list[leftMark] <= pivotVal)                  leftMark++;              while (leftMark <= rightMark && list[rightMark] >= pivotVal)                  rightMark--;              if (leftMark <= rightMark)                  swap(list, leftMark, rightMark);          }          swap(list, startIndex, rightMark);          return rightMark;      }      static void swap(int[] list, int i, int j) {          int temp = list[i];          list[i] = list[j];          list[j] = temp;      }  } |

2. Testing inputs and outputs (compulsory)

|  |  |
| --- | --- |
| Input | Output |
| Case 1 |  |
| Case 2 |  |
| Case 3 |  |

3. Analysis of experiment results(if required)

4. Reflection on the use of online resources including GenAI (if any)

* What resource? (e.g., ChatGPT, Stack Overflow)
* How did you use it?
* How did it assist you in completing this task?

Task 6: Intersection of sets

1. Solution code (compulsory)

|  |
| --- |
| import java.util.\*;  public class ArrayIntersection {      public static void findIntersection(int[] X, int[] Y) {          long startTime = System.currentTimeMillis();          HashSet<Integer> X\_set = new HashSet<>();          for (int num : X) {              if (!X\_set.contains(num)) // Remove repetitive items                  X\_set.add(num);          }          // Find intersection elements          HashSet<Integer> intersectionSet = new HashSet<>();          for (int num : Y) {              if (X\_set.contains(num) && !intersectionSet.contains(num))                  intersectionSet.add(num);          }          System.out.print("Intersection of X and Y: ");          System.out.println(intersectionSet);          long endTime = System.currentTimeMillis();          long deltaHS = endTime - startTime;          System.out.println("Hash set execution time: " + deltaHS + " ms.");      }      public static void bruteForce(int[] X, int[] Y) {          long startTime = System.currentTimeMillis();          ArrayList<Integer> intersectionList = new ArrayList<Integer>();          for (int i : X) {              for (int j : Y) {                  if (i == j && !intersectionList.contains(i)) {                      intersectionList.add(i);                  }              }          }          System.out.println(intersectionList.size());          long endTime = System.currentTimeMillis();          long deltaBF = endTime - startTime;          System.out.println("Brute force execution time: " + deltaBF + " ms.");      }      public static void main(String[] args) {          int[] X = new int[50000];          int[] Y = new int[X.length];          X = ArrayUtils.generateRandomArray(X.length, 7);          Y = ArrayUtils.generateRandomArray(Y.length, 7);          findIntersection(X, Y);          bruteForce(X, Y);      }  } |

2. Testing inputs and outputs (compulsory)

|  |  |
| --- | --- |
| Input | Output |
| Case 1 | Intersection of X and Y: [-1, 0, 1, -2, -3, 2, -4, 3, 4, -5, -6, 5, 6, -7, -8, 7, 8, -9, -10, 9, -11, 10, 11, -12, 12, -13, 13, -14, 14, -15, -16, 15, 16, -17, -18, 17, -19, 18, -20, 19, -21, 20, -22, 21, -23, 22, -24, 23, 24, -25, 25, -26, -27, 26, -28, 27, -29, 28, 29, -30, 30, -31, 31, -32, 32, -33, 33, -34, -35, 34, -36, 35, 36, -37, 37, -38, 38, -39, 39, -40, 40, -41, -42, 41, -43, 42, 43, -44, 44, -45, -46, 45, 46, -47, 47, -48, 48, -49, -50, 49, -51, 50, -52, 51, 52, -53, 53, -54, -55, 54, 55, -56, -57, 56, -58, 57, -59, 58, -60, 59, 60, -61, 61, -62, -63, 62, 63, -64, 64, -65, -66, 65, -67, 66, 67, -68, 68, -69, -70, 69, 70, -71, 71, -72, -73, 72, 73, -74, -75, 74, -76, 75, 76, -77, 77, -78, 78, -79, -80, 79, -81, 80, -82, 81, 82, -83, -84, 83, 84, -85, 85, -86, 86, -87, -88, 87, -89, 88, -90, 89, -91, 90, 91, -92, 92, -93, -94, 93, 94, -95, 95, -96, -97, 96, 97, -98, -99, 98, 99, -100, 100, -101, 101, -102, 102, -103, 103, -104, -105, 104, -106, 105, 106, -107, -108, 107, 108, -109, 109, -110, 110, -111, 111, -112, 112, -113, -114, 113, -115, 114, 115, -116, -117, 116, 117, -118, -119, 118, 119, -120, 120, -121, 121, -122, -123, 122, 123, -124, -125, 124, 125, -126, -127, 126, 127, -128, -129, 128, 129, -130, -131, 130, 131, -132, 132, -133, -134, 133, -135, 134, 135, -136, -137, 136, 137, -138, 138, -139, 139, -140, 140, -141, -142, 141, 142, -143, 143, -144, 144, -145, 145, -146, -147, 146, 147, -148, 148, -149, -150, 149, -151, 150, -152, 151, -153, 152, -154, 153, 154, -155, 155, -156, 156, -157, 157, -158, -159, 158, 159, -160, -161, 160, -162, 161, -163, 162, 163, -164, 164, -165, 165, -166, 166, -167, -168, 167, 168, -169, 169, -170, 170, -171, 171, -172, -173, 172, 173, -174, -175, 174, -176, 175, 176, -177, 177, -178, -179, 178, 179, -180, -181, 180, 181, -182, 182, -183, -184, 183, -185, 184, 185, -186, -187, 186, -188, 187, -189, 188, 189, -190, 190, -191, 191, -192, 192, -193, -194, 193, 194, -195, 195, -196, 196, -197, -198, 197, 198, -199, -200, 199, -201, 200, -202, 201, -203, 202, 203, -204, 204, -205, -206, 205, -207, 206, -208, 207, 208, -209, 209, -210, 210, -211, 211, -212, 212, -213, 213, -214, 214]  Hash set execution time: 34 ms.  429  Brute force execution time: 5688 ms. |
| Case 2 | Intersection of X and Y: [0, -1, 1, -2, -3, 2, -4, 3, -5, 4, 5, -6, 6, -7, -8, 7, -9, 8, -10, 9, -11, 10, 11, -12, 12, -13, 13, -14, 14, -15, 15, -16, 16, -17, 17, -18, 18, -19, -20, 19, -21, 20, 21, -22, 22, -23, 23, -24, 24, -25, -26, 25, 26, -27, -28, 27, -29, 28, 29, -30, -31, 30, 31, -32, 32, -33, -34, 33, 34, -35, 35, -36, -37, 36, 37, -38, 38, -39, -40, 39, 40, -41, 41, -42, 42, -43, 43, -44, 44, -45, -46, 45, 46, -47, 47, -48, 48, -49, 49, -50, 50, -51, 51, -52, 52, -53, -54, 53, -55, 54, 55, -56, -57, 56, -58, 57, -59, 58, 59, -60, -61, 60, 61, -62, 62, -63, -64, 63, -65, 64, -66, 65, -67, 66, -68, 67, 68, -69, -70, 69, -71, 70, 71, -72, -73, 72, -74, 73, -75, 74, -76, 75, 76, -77, -78, 77, -79, 78, -80, 79, -81, 80, -82, 81, -83, 82, -84, 83, -85, 84, -86, 85, 86, -87, 87, -88, 88, -89, 89, -90, 90, -91, -92, 91, -93, 92, 93, -94, -95, 94, -96, 95, -97, 96, -98, 97, -99, 98, -100, 99, -101, 100, 101, -102, -103, 102, -104, 103, -105, 104, -106, 105, -107, 106, -108, 107, 108, -109, 109, -110, 110, -111, -112, 111, 112, -113, -114, 113, -115, 114, 115, -116, -117, 116, -118, 117, -119, 118, -120, 119, 120, -121, 121, -122, -123, 122, -124, 123, 124, -125, 125, -126, 126, -127, 127, -128, 128, -129, -130, 129, 130, -131, 131, -132, -133, 132, -134, 133, -135, 134, 135, -136, 136, -137, 137, -138, -139, 138, 139, -140, 140, -141, 141, -142, -143, 142, -144, 143, 144, -145, 145, -146, 146, -147, -148, 147, -149, 148, 149, -150, -151, 150, 151, -152, -153, 152, -154, 153, 154, -155, -156, 155, -157, 156, 157, -158, -159, 158, 159, -160, 160, -161, 161, -162, 162, -163, 163, -164, 164, -165, 165, -166, -167, 166, -168, 167, 168, -169, 169, -170, 170, -171, 171, -172, 172, -173, -174, 173, -175, 174, 175, -176, 176, -177, 177, -178, -179, 178, 179, -180, -181, 180, 181, -182, 182, -183, 183, -184, -185, 184, 185, -186, 186, -187, -188, 187, -189, 188, -190, 189, 190, -191, 191, -192, 192, -193, 193, -194, -195, 194, -196, 195, 196, -197, 197, -198, -199, 198, 199, -200, 200, -201, -202, 201, -203, 202, 203, -204, 204, -205, 205, -206, 206, -207, -208, 207, 208, -209, 209, -210, 210, -211, 211, -212, -213, 212, 213, -214, 214]  Hash set execution time: 26 ms.  429  Brute force execution time: 5858 ms. |
| Case 3 | Intersection of X and Y: [-1, 0, -2, 1, -3, 2, 3, -4, -5, 4, -6, 5, 6, -7, 7, -8, 8, -9, 9, -10, -11, 10, 11, -12, 12, -13, -14, 13, -15, 14, -16, 15, 16, -17, 17, -18, 18, -19, 19, -20, 20, -21, -22, 21, 22, -23, -24, 23, 24, -25, -26, 25, 26, -27, -28, 27, 28, -29, -30, 29, 30, -31, 31, -32, -33, 32, -34, 33, 34, -35, 35, -36, -37, 36, -38, 37, -39, 38, -40, 39, -41, 40, -42, 41, -43, 42, 43, -44, 44, -45, 45, -46, 46, -47, -48, 47, -49, 48, -50, 49, -51, 50, 51, -52, -53, 52, 53, -54, -55, 54, -56, 55, 56, -57, -58, 57, 58, -59, 59, -60, 60, -61, 61, -62, 62, -63, 63, -64, 64, -65, 65, -66, -67, 66, 67, -68, 68, -69, -70, 69, -71, 70, 71, -72, 72, -73, 73, -74, -75, 74, 75, -76, -77, 76, -78, 77, 78, -79, -80, 79, -81, 80, -82, 81, 82, -83, 83, -84, -85, 84, 85, -86, -87, 86, -88, 87, -89, 88, 89, -90, 90, -91, 91, -92, 92, -93, -94, 93, -95, 94, 95, -96, -97, 96, -98, 97, 98, -99, -100, 99, 100, -101, 101, -102, 102, -103, -104, 103, -105, 104, 105, -106, -107, 106, -108, 107, 108, -109, -110, 109, 110, -111, 111, -112, -113, 112, -114, 113, 114, -115, -116, 115, -117, 116, -118, 117, -119, 118, 119, -120, -121, 120, -122, 121, -123, 122, 123, -124, 124, -125, 125, -126, -127, 126, 127, -128, 128, -129, 129, -130, 130, -131, -132, 131, 132, -133, 133, -134, -135, 134, 135, -136, 136, -137, -138, 137, 138, -139, -140, 139, -141, 140, 141, -142, 142, -143, -144, 143, -145, 144, 145, -146, 146, -147, -148, 147, -149, 148, 149, -150, 150, -151, -152, 151, 152, -153, -154, 153, 154, -155, -156, 155, -157, 156, -158, 157, 158, -159, 159, -160, 160, -161, 161, -162, 162, -163, -164, 163, -165, 164, 165, -166, 166, -167, -168, 167, 168, -169, 169, -170, 170, -171, 171, -172, -173, 172, 173, -174, 174, -175, 175, -176, -177, 176, -178, 177, 178, -179, 179, -180, -181, 180, 181, -182, -183, 182, -184, 183, 184, -185, -186, 185, -187, 186, 187, -188, -189, 188, 189, -190, 190, -191, -192, 191, -193, 192, 193, -194, 194, -195, -196, 195, -197, 196, -198, 197, 198, -199, 199, -200, -201, 200, 201, -202, -203, 202, -204, 203, -205, 204, -206, 205, 206, -207, -208, 207, -209, 208, -210, 209, -211, 210, 211, -212, -213, 212, 213, -214, 214]  Hash set execution time: 64 ms.  429  Brute force execution time: 7449 ms. |

3. Screenshots of tests (compulsory)

A black background with many small squares

Description automatically generated with medium confidence A black background with many small squares

Description automatically generated A black background with many small squares

Description automatically generated

4. Analysis of experiment results(if required)

5. Reflection on the use of online resources including GenAI (if any)

* What resource? (e.g., ChatGPT, Stack Overflow)
* How did you use it?
* How did it assist you in completing this task?

Module Three Programming Tasks

Task 7: First occurrence of binary search

1. Solution code (compulsory)

|  |
| --- |
|  |

2. Testing inputs and outputs (compulsory)

|  |  |
| --- | --- |
| Input | Output |
| Case 1 |  |
| Case 2 |  |
| … |  |

3. Screenshots of tests (compulsory)

4. Analysis of experiment results(if required)

5. Reflection on the use of online resources including GenAI (if any)

* What resource? (e.g., ChatGPT, Stack Overflow)
* How did you use it?
* How did it assist you in completing this task?

Task 8: Stable sort insertion

1. Solution code (compulsory)

|  |
| --- |
|  |

2. Testing inputs and outputs (compulsory)

|  |  |
| --- | --- |
| Input | Output |
| Case 1 |  |
| Case 2 |  |
| … |  |

3. Screenshots of tests (compulsory)

4. Analysis of experiment results(if required)

5. Reflection on the use of online resources including GenAI (if any)

* What resource? (e.g., ChatGPT, Stack Overflow)
* How did you use it?
* How did it assist you in completing this task?

Appendix

A utility class was created to generate arrays of integers with varying degrees of magnitude.

Such arrays can be created by using the method ArraysUtils.generateRandomArray(size, order).

import java.util.\*;

import java.lang.Math;

public class ArrayUtils {

    // Method to generate an array of random integers

    public static int[] generateRandomArray(int size, double order) {

        //order will reduce the order/magnitude of the integers generated. eg order 0 will generate integers to the 10^8 power, so to reduce integers to the range 1 - 20, the order should be set to 8

        int[] randomNumbers = new int[size]; // Initialize an array with the specified size

        Random random = new Random();

        for (int i = 0; i < size; i++) {

            randomNumbers[i] = (int) (random.nextInt()/Math.pow(10,order)); // Generate a random integer and assign it to the array

        }

        return randomNumbers; // Return the generated array

    }

}