SUBJECT: SOFTWARE DESIGN 2	ANNEXURE(S) Formulas	PAGES 3	TIME 3 HRS
CODE: SDN260S	DATE 03 October 2022	MARKS 76	

Cape Peninsula University of Technology

FACULTY OF ENGINEERING

ASSESSMENT II: SEMESTER TWO: MEMO

COURSE: BET: COMPUTER ENGINEERING

EXAMINER	:	Haltor Mataifa
MODERATOR (INTERNAL)	:	Mr. V. Moyo
MODERATOR (EXTERNAL)	:	N/A

SPECIAL INSTRUCTIONS

1. Answer **ALL** questions
2. Write your name and student number on each page of everything you submit
3. Write answers to theoretical questions in an MS Word document
4. Create a new project for each programming problem
5. Zip all files and folders together and submit as one zipped file on Blackboard
6. Be sure to add comments to your program code to make it understandable

QUESTION 1 **[16]**

For each of the following statements, state whether it is true or false. In each case, give a motivation for your answer.

1.1. Binary search is a better algorithm than linear search because it is easier to implement [**2**]

Answer:

- ***False***. *Binary search is actually more complex than linear search, but is also more efficient for large data sets*

1.2. Linear search executes in less time than binary search because, unlike binary search, it does not require the array to be sorted [2]

Answer:

- ***False***. *Although binary search requires the array to be sorted, it is far more efficient than linear search, and so requires less execution time for data sets of arbitrary size*

1.3. Selection sort is slower than merge sort because it is more complex [2]

Answer:

- ***False***. *Selection sort is slower than merge sort, but it is much less complex*

1.4. An **O(n)** algorithm is more efficient than an **O(nlog n)** algorithm [2]

Answer:

- ***True***. *An **O(n)** algorithm executes in less time than an **O(nlog n)** algorithm for arbitrary data sets*

1.5. Auto-unboxing occurs when a value is assigned to a primitive variable using an element from a Collection [2]

Answer:

- ***True***. *Assigning a value to a primitive variable using an element from a Collection causes implicit data type conversion from a reference type to a primitive type, which is referred to as auto-unboxing*

1.6. A Vector is more efficient than an ArrayList because it is synchronized [2]

Answer:

- ***False***. *The fact that a Vector is synchronized makes it relatively less efficient, because of the synchronization overhead*

1.7. A Set differs from a List in the sense that a Set can contain duplicate elements [2]

Answer:

- ***False***. *A Set cannot contain duplicate elements, a List can*

1.8. Generic methods enable us to call the same (generic) method with a different number of inputs [2]

Answer:

- ***False***. *Generic methods enable us to operate on inputs of different data types (overloaded methods are the ones that enable us to call the same method with a different number of inputs)*

QUESTION 2: **[30]**

Design a Java application to implement the bubble sort algorithm as a generic method to sort a given array of any valid data type in ascending order. The bubble sort algorithm works as follows:

- The algorithm makes several passes through an array. Each pass compares successive pairs of elements
- If a pair is in increasing order (or the values are equal), the pair is left as it is. If a pair is in decreasing order, the positions of the elements are swapped
- The first pass compares the first two elements, swapping the positions if they are in decreasing order (leaving them unchanged otherwise), then it compares the second and third elements, and continues doing so until it compares the last two elements of the array
- After *i* passes through the array, the largest *i* elements occupy the last *i* positions in the array

Generate two 10-element random-number arrays, one of integers, the other of doubles, each in the **range 1 to 49**, and use them to test the generic bubble sort method. Output both the unsorted and the sorted arrays in each case.

Answer:

```
import java.util.ArrayList;
import java.util.Random;

public class GenericBubbleSort {
   public static void main(String[] args) {
      Random gen=new Random();

      // instantiate two ArrayLists, one for integers, another for doubles:
      ArrayList<Integer> integerList=new ArrayList<>();
      ArrayList<Double> doubleList=new ArrayList<>();

     // generate the random integer and double elements as specified:
      for (int i=0; i<10; i++)
      {
         integerList.add(1+gen.nextInt(49));
         doubleList.add(1+48*(double)Math.round(gen.nextDouble()*100)/100);
      }

      // print the unsorted and sorted arrays:
      System.out.println("\nUnsorted integer array: "+integerList.toString());
      bubbleSort(integerList);
      System.out.println("Sorted integer array: "+integerList.toString());

      System.out.println("\nUnsorted double array: "+doubleList.toString());
      bubbleSort(doubleList);
      System.out.println("Sorted double array: "+doubleList.toString());

   } // end main method
```

```
    // define generic bubble sort algorithm
    public static <T extends Comparable<T>> void bubbleSort(ArrayList<T> array)
    {
       for (int i=0; i<array.size()-1; i++)
       {
          for (int j=1; j<array.size()-i; j++)
          {
             if (array.get(j).compareTo(array.get(j-1))<0)
             {
                T temp=array.get(j-1);
                array.set(j-1, array.get(j));
                array.set(j, temp);
             }
          }
       }
    }  // end method bubbleSort

} // end class GenericBubbleSort
```

QUESTION 3: **[30]**

Design and implement a Java application with a generic method that searches for an element entered by a user in a given array. The generic search method takes as parameters the element to search for, and the array in which to search the element for. It returns the index of the search element if the array contains the element, otherwise an indication that the search element was not found. You can make use of appropriate **Collections** methods in your implementation of the generic search method.

Test the generic method using the following arrays:

- int array1=[8, 21, 6, 1, 24, 9, 47, 36, 18, 38]
- double array2=[14.7, 7.84, 20.58, 26.46, 35.28, 10.78, 35.28, 6.86, 30.87, 17.15]
- String array3=[blue, red, yellow, green, white, cyan, magenta, grey, black, brown]

In each case that you test the method, you will prompt the user to enter the element to search for. Specify to the user the element type (weather integer, double or String), and perform the search in the relevant array. Output the results of the search. The following snapshot shows an example of running the program with various user inputs.

```
Enter an integer to search for:
24
Enter a floating point number to search for:
35.28
Enter a string to search for:
red

24 was found at index 6 in array:
[1, 6, 8, 9, 18, 21, 24, 36, 38, 47]

35,28 was found at index 8 in array:
[6.86, 7.84, 10.78, 14.7, 17.15, 20.58, 26.46, 30.87, 35.28, 35.28]

red was found at index 7 in array:
[black, blue, brown, cyan, green, grey, magenta, red, white, yellow]
```

Answer:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Scanner;

public class GenericSearch {
	public static void main(String[] args) {

	// Instantiate integer, double and string arrays as specified:
	Integer[] integerArray={8, 21, 6, 1, 24, 9, 47, 36, 18, 38};
	Double[] doubleArray={14.7,  7.84,  20.58,  26.46,  35.28,
	   10.78,  35.28,  6.86,  30.87,  17.15};
	String[] stringArray={"blue", "red", "yellow", "green", "white", "cyan",
	"magenta", "grey", "black", "brown"};

	// Obtain a list view of each of the arrays, to enable using Collections methods:
	ArrayList<String> stringList=new ArrayList<>(Arrays.asList(stringArray));
	ArrayList<Integer> integerList=new ArrayList<>(Arrays.asList(integerArray));
	ArrayList<Double> doubleList=new ArrayList<>(Arrays.asList(doubleArray));

	// Instantiate a Scanner object to accept user input
	Scanner input = new Scanner(System.in);

	// Prompt user to enter an integer to search for:
	System.out.println("\nEnter an integer to search for: ");
	int intSearchElement=input.nextInt();
	input.nextLine();

	// Prompt user to enter a double to search for:
	System.out.println("Enter a floating point number to search for: ");
	String doubleString=input.nextLine();
	double doubleSearchElement=Double.parseDouble(doubleString);

	// Prompt user to enter a string to search for:
```

```
        System.out.println("Enter a string to search for: ");
        String stringSearchElement=input.nextLine();

        // Search for the integer input and print the result of the search:
        int intIndex=search(intSearchElement, integerList);
        if (intIndex<0)
           System.out.printf("\n%d was not found in array:\n%s\n",intSearchElement, integerList);
        else
           System.out.printf("\n%d was found at index %d in array:\n%s\n",intSearchElement, intIndex,
integerList);

        // Search for the double input and print the result of the search:
        int doubleIndex=search(doubleSearchElement, doubleList);
        if (doubleIndex<0)
           System.out.printf("\n%.2f was not found in array:\n%s\n",doubleSearchElement, doubleList);
        else
           System.out.printf("\n%.2f was found at index %d in array:\n%s\n",
                 doubleSearchElement, doubleIndex, doubleList);

        // Search for the string input and print the result of the search:
        int stringIndex=search(stringSearchElement, stringList);
        if (stringIndex<0)
           System.out.printf("\n%s was not found in array:\n%s\n",stringSearchElement, stringList);
        else
           System.out.printf("\n%s was found at index %d in array:\n%s\n",
                 stringSearchElement, stringIndex, stringList);

    }

    // Define a generic method to search of an element entered by user:
    public static <T extends Comparable<T>> int search(T searchElement, ArrayList<T> array)
    {
       Collections.sort(array);
       return Collections.binarySearch(array, searchElement);
    }  // end method search
}   // end class GenericSearch
```
