Computer Science 2210 – Mid-Term Exam

November 2, 2011 **Grade:** 93 Name: Brandon Buchanan

1. For some of the methods in the ***List<T>*** class in ***C#*** to work properly, the data type ***T*** must implement certain interfaces. For methods such as ***Contains*** and ***IndexOf***, ***T*** must implement the ***interface*** named IEquatable and for other methods such as ***Sort***, ***T*** must implement the ***interface*** named IComparable. ***List<T>*** is one example of a group of specialized collection classes in ***C#*** that permit one or more parameterized types (such as ***T*** in ***List<T>***) to be specified so that an entire family of classes may be created by substituting appropriate actual data types for the parameterized types in a variety of programs. The formal term used to describe these specialized collection classes in ***C#*** and ***.NET*** is generic collection classes.

2. The ***pattern matching technology*** we discussed for use with ***Strings*** is called regular expressions

3. To implement the ability for a class named ***Gismo*** to use a ***"plus operator"*** on two objects of type ***Gismo*** (i.e., to enable the use of code such as ***gis3 = gis1 + gis2;*** to compile correctly for ***Gismo*** objects ***gis1***, ***gis2***, ***gis3***), one must create a method with the following header line: public static Gismo operator + (Gismo gis1, Gismo gis2)

4. Circle each of the following sorting methods that we studied in class.

(a.) quicksort	(e.) shell sort	(i.) sinking sort
b. heap sort	f. radix sort	j. merge sort
(c.) insertion sort	(g.) tag sort	k. all of these
d. bin sort	h. bucket sort	l. none of these

5. If a user-defined class ***C*** has a private attribute representing an indexable collection (***array***, ***List***, etc.) of objects of another class, ***C*** may have a specialized property called a(n) indexer that permits the user of the class to have controlled access to individual items in the collection, using a technique familiar to any ***C#*** programmer who has used ***arrays***, ***Lists***, and similar collections in ***C#***.

6. The type of ***search algorithm*** that allows one to locate a specific item in a sorted array of items in $O(\log_2 N)$ time is called the binary search algorithm. The common search algorithm that we discussed with ***O(N)*** performance is called the sequential search algorithm.

7. Consider the integer array values to be sorted into ascending order: ***53 72 16 95 39 63 24 88 44 71 57***

   a. After the first exchange of values in a ***Selection sort***, the values are: 53 57 16 95 39 63 24 88 44 71 72

   b. After the first pass of a ***Sinking sort***, the values are: 53 72 16 39 63 24 88 44 71 57 95

   c. After one pass with a ***gap of 3*** in a ***Shell sort***, the values are: 24 39 16 53 57 44 57 72 63 95 88

   d. After inserting the first 3 items in an ***Insertion Sort***, the array is: 16 24 39 95 53 63 72 88 44 71 57 -2

   e. Using the ***median-of-three*** technique to determine a pivot for the ***Quick Sort*** on this group of integers, the "three" are 53, 63, and 57, and the ***median*** of them is 57.

8. The number of operations required by a certain algorithm to process ***N*** data items has been computed to be at most $N \log_2 N + 2.15 N^3 - 47 N^2 + 2 N / 3 + 167$ operations. The ***Big-Oh*** notation used to describe this fact is $O(N^3)$.

9. To calculate **N!** in a "standard" way requires (how many) N-1 multiplications, 0 divisions, 0 additions, and 0 subtraction operations for a total of N-1 operations. Thus, the ***big-O*** notation for this is effort is O(N). You can verify your results by writing the calculation of, say, 5! and 10! on scrap paper. -4

10. ***Big-Oh*** notation is used to described a(n) upper ***bound*** on the performance of an algorithm in the (circle one) ***average / best / worst*** case based on the number of items (***N***) it must process for large values of ***N***. Similarly, ***Ω*** is used to describe a(n) lower bound on an algorithm's performance. -2

11. The class named Object is the ultimate base class for all classes in C#. From it, other classes inherit methods such as ***ToString***, ***Equals***, and ***GetHashCode***.

12. Write code to define a ***Title*** property in a ***Book*** class public Title title() {get; set;}

Points deducted: 8

Computer Science 2210 – Mid-Term Exam

November 2, 2011 **Grade:** _____ Name: _____

13. In doing a binary search for 73 on an ***List<int>*** named ***list*** containing ***0 – 99*** in its ***100*** positions using the following code, show the output produced by the code when called by this instruction:

```
int subscript = BinarySearch (theList, 73, 0, 99);
```

```
41      private static int BinarySearch (List<int> list, int SearchKey, int From, int To)
42      {
43          if (From > To)
44              return -1;
45
46          int Mid = (From + To) / 2;  // find midpoint
47          Console.Write (list[Mid] + " ");
48          if (SearchKey == list[Mid])
49          {
50              Console.WriteLine ( );
51              return Mid;
52          }
53          else
54              if (SearchKey < list[Mid])
55                  return BinarySearch (list, SearchKey, From, Mid - 1);   Recursively search lower half
56              else
57                  return BinarySearch (list, SearchKey, Mid + 1, To);     Recursively search upper half
58      }
```

0, 99
50, 99
50, 73
62, 73
68, 73
71, 73
73, 73

Answer: 49 74 61 67 70 72 73

14. Consider the following code in answering the questions below:

```
11      static void Main (string[ ] args)
12      {
13          String pat  = @"\b[0-9]{2,4}\b";
14          String str  = "Strings store Unicode characters.";
15          String test = "The 88 winners were selected from 888 contestants in contest number 8 in 2011.";
16
17          int col = str.IndexOfAny ("code".ToCharArray ( ));
18
19          Regex rgx = new Regex (pat);
20          MatchCollection collection = rgx.Matches (test);
21          Console.WriteLine ("The character found was '{0}' in column {1}", str[col], col);
22          foreach (Match match in collection)
23              Console.WriteLine (match.Value);
24      }
```

-2 a. What is output by the code on line 21? The character found was 'c' in column 17

b. What is output by the code on lines 22-23?

-2

88
88
88
20

01
11
2011

Answer in this box

15. Write ***T*** for ***True*** or ***F*** for ***False*** in the blanks below.

a. F Every algorithm with $O(N^3)$ performance also has $O(N^2)$ performance
b. T For large values of N, $(N \log_2 N) < 2^N$
c. F Algorithms with ***exponential performance*** are acceptable for values of ***N*** up to about 1000 usually
d. F For-loops nested 3 levels deep always have $O(N^3)$ performance or worse
e. F Typecasting in C# can be used to change the declared data type of a variable at run time
f. T C# implicitly convert an ***integer*** into a ***double***, but it will not implicitly convert a ***double*** into an ***integer***
g. F ***Event handlers*** in C# must be ***static*** methods
h. T A ***regular expression*** may be used to verify that a string matches a designated pattern
i. T The ***String*** class implements the comparison operators ***<, >, ==***, and so forth that allows its users to compare ***String*** objects in ***C#*** code using these operators
j. T The "*internal*" array of an object of ***List<Book>*** will be reallocated and copied whenever one tries to add a ***Book*** and the current ***capacity*** of the ***List*** is equal to its ***Count*** property
k. F Performing a ***selection sort*** on an array of ***N*** items is a $O(\log^2 N)$ operation
l. F A baseball player bats up to 600 times in a season. Calculating the batting average of all 25 players on all of the 30 major league baseball teams is a $O(N)$ operation

Points deducted: 4

16. Consider the following code in answering the question after it. The method is in the Utility class.

```
158         public static List<String> Tokenize (string original, string delimiters)
159         {
160             List<string> Tokens = new List<string> ( );
161             string work   = original;
162             work = work.Trim();
163
164             int col;
165             string token;
166
167             while (!String.IsNullOrEmpty (work))
168             {
169                 col = work.IndexOfAny (delimiters.ToCharArray ( ));
170                 if (col == 0)
171                     col = 1;
172                 token = work.Substring (0, col);
173                 Tokens.Add (token);
174                 work = work.Substring (col);
175                 work = work.Trim (" \t".ToCharArray ( ));
176             }
177             return Tokens;
178         }
```

a. What output is produced by the following code that uses the above method from the Utility class.

```
12          static void Main (string[ ] args)
13          {
14              String str = "Alpha = Beta+Gamma / (Delta -4);";
15              //str = "computer";
16              String delims = "+=-/* ()";
17              List<String> list = Utility.Tokenize (str, delims);
18              foreach (String s in list)
19              {
20                  Console.WriteLine (s);
21              }
22          }
```

Answer:

Alpha

Beta

Gamma

Delta

4

;

-1

b. If line 15 had been uncommented, please explain how the results of executing the code above would change.

-2 list would be empty and no output would be displayed

17. Consider the partial class defined below in answering the questions that follows it:

```
16  public class Name
17  {
18      ---------- Properties - first and last names ----------
19      public String First { get; set; }
20      public String Last { get; set; }
21
22      ---------- Constructors ----------
23      public Name ( ) ...
28
29      public Name (String name)
30      {
31          Trim leading and trailing spaces
32          name = name.Trim ( );
33
34          Compress multiple spaces into a single space
35          Regex MultipleSpacesPattern = new Regex (@"\s+");
36          name = MultipleSpacesPattern.Replace (name, " ");
37
38          Extract First and Last names
39          int comma = name.IndexOf (',');
40          int space = name.IndexOf (' ');
41
42          if (comma == -1)
43          {
44              if (space < 0)
45                  throw new ArgumentException ("The name does not include both a first and a last name");
46              First = name.Substring (0, space);
47              Last = name.Substring (space + 1);
48          }
49          else
50          {
51              Last = name.Substring (0, comma);
52              First = name.Substring (comma + 2);
53          }
54
55          Set correct casing
56          Last = Last[0].ToString ( ).ToUpper ( ) + Last.Substring (1).ToLower ( );
57          First = First[0].ToString ( ).ToUpper ( ) + First.Substring (1).ToLower ( );
58
59      }
60
```

a. The type of C# entity (not its data type) represented by ***First*** on 19 is an attribute

b. What is output by the following two lines of code that use the code above?

```
Name name = new Name ("  Jones,  John          David  ");
Console.WriteLine (name.First + " " + name.Last);
```

John David Jones

c. If ***name*** had been declared as ***Name name = new Name ("Prince");*** how would the result above have changed?

No comma or space exists; so an ArgumentException would be thrown.