Exam Number:

| Points off | 1 | 2 | 3 | 4 | | Total off | Net Score |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

# CS 307 – Midterm 2 – Spring 2008

Name_____

UTEID login name _____

TA's Name: Mario Ruchica Vishvas (Circle One)

Instructions:

1. Please turn off your cell phones and other electronic devices.
2. There are 4 questions on this test.
3. You have 2 hours to complete the test.
4. You may not use a calculator on the test.
5. When code is required, write Java code.
6. When writing methods, assume the preconditions of the method are met.
7. In coding question you may add helper methods if you wish.
8. After completing the test please turn it in to one of the test proctors and show them your UTID.

1. (2 points each, 30 points total) Short answer. Place you answers on the attached answer sheet.

- If the code contains a syntax error or other compile error, answer "compile error".
- If the code would result in a runtime error / exception answer "Runtime error".
- If the code results in an infinite loop answer "Infinite loop".

Recall that when asked for Big O your answer should be the most restrictive correct Big O function. For example Selection Sort has an average case Big O of O(N^2), but per the formal definition of Big O it is correct to say Selection Sort also has a Big O of O(N^3) or O(N^4). I want the most restrictive correct Big O function. (Closest without going under.)

A. What is the output of `System.out.println( a(7) );`

```
public int a(int n){
    if( n > 10 )
        return 1;
    else
        return 2 + a(n + 1);
}
```

B. What is the output of `System.out.println( b(4) );`

```
public int b(int val){
    if( val <= 1 )
        return val;
    else
        return val + b(val - 1) + b(val - 2);
}
```

C. What is the output when method `c` is called?

```
public void c(){
    int[] data = {5, 3, 2, 6, 1, -5, 3};
    System.out.println( helper(data, 0) );
}

public int helper(int[] list, int p){
    if( p == list.length - 1 )
        return list[p];
    else
        return Math.max( helper(list, p + 1), list[p] );
}
```

D. What is the worst case Big O of method `d`? N = `nums.length`.

```
public int d(int[] nums){
    int total = 0;
    for(int i = 0; i < nums.length; i++)
        for(int j = nums.length - 1; j >= 0; j--)
            if( nums[i] == nums[j] )
                total++;
    return total;
}
```

E. What is the Big O of method `e`? N = `nums.length`.

```
public int e(int[] nums){
    int total = 0;
    for(int i = 0; i < nums.length; i++)
        total += nums[i];

    int limit = nums.length / 2;
    for(int j = 0; j < limit; j++)
        total += nums[j] * nums[j];

    return total;
}
```

F. What is the Big O of method `f`? N = `mat.length`

```
// pre: mat is a square matrix, mat.length > 0
public int f(int[][] mat){
    int row = 1;
    int col = 1;
    int total = 0;
    while( row < mat.length ){
        total += mat[row][col];
        row *= 3;
        col *= 3;
    }
    return total;
}
```

G. What is the best case Big O of method `g`? N = `numbers.length`

```
public static void g(int[] numbers) {
    int limit = numbers.length;
    for (int pass = 1; pass < limit; pass++) {
        for (int i = 0; i < limit - pass; i++) {
            if (numbers[i] > numbers[i + 1]) {
                int temp = numbers[i];
                numbers[i] = numbers[i + 1];
                numbers[i + 1] = temp;
            }
        }
    }
}
```

H. What is the Big O of method `h`? Method `compute` is O(N) where N is the length of the array sent as a parameter to the method. N = `table.length`.

```
// table is a square matrix
public int h(int[][] table){
    int answer = 0;

    for(int row = 0; row < table.length; row++)
        for(int col = row; col < table.length; col++)
            answer += compute(table[row], row, col);

    return answer;
}
```

I. What is the Big O of method `helper`? N = `list.length.`

```
public int helper(int[] list, int p){
    if( p == list.length - 1 )
        return list[p];
    else
        return Math.max( helper(list, p + 1), list[p] );
}
```

J. What is the Big O of the following code? N = `list.size()`

```
public double empty(ArrayList<String> list){
     int total = 0;
     int size = list.size();
     while( list.size() != 0 ){
          total += list.get(0).length();
          list.remove(0);
     }
     return 1.0 * total / size;
}
```

K. Consider the following methods from the Java ArrayList class.

| Method Summary | |
|---|---|
| void | **add**(E o)<br>Appends the specified element to the end of this list. |
| void | **add**(int index, E element)<br>Inserts the specified element at the specified position in this list. |
| E | **get**(int index)<br>Returns the element at the specified position in this list. |
| E | **remove**(int index)<br>Removes the element at the specified position in this list. Return the element that is removed from the list. |
| E | **set**(int index, E element)<br>Replaces the element at the specified position in this list with the specified element. Returns the element that was previously at index. |
| int | **size**()<br>Returns the number of elements in this list. |

What is output when method `k` is called?

```
public void k(){
    ArrayList<String> lt = new ArrayList<String>();

    lt.add( "C" );
    lt.add(0, "A");
    lt.add(1, "B");
    lt.add(0, "D");
    lt.set(1, "E" );
    lt.add( lt.remove(2) );
    for(int i = 0; i < lt.size(); i++)
        System.out.print( lt.get(i) );
}
```

L. What is output when method `el` is called? The `LinkedList` class implements the same methods as the `ArrayList` class.

```
public void el(){
    LinkedList<Integer> list = new LinkedList<Integer>();
    for(int i = 0; i < 5; i++)
        list.add( i );

    for(int j = list.size() - 2; j >= 0; j--)
        list.set( j, list.get(j) + list.get( j + 1 ) );

    for(int i = 0; i < list.size(); i++)
        System.out.print( list.get(i) + " " );
}
```

M. What is output when method `m` is called? Assume the `Stack` class implements a traditional stack.

```
public void m(){
    Stack<Integer> st = new Stack<Integer>();
    for(int value = 10; value >= -3; value -= 2)
        st.push( value );

    for(int i = 0; i < 3; i++)
        System.out.print( st.pop() + " " );
}
```

N. A method is $O(N^3)$. It takes 1 second for the method to complete on a data set with 5,000 items. What is the expected time for the method to complete with a data set of 20,000 items?

O. Method `quickSort` implements the Quicksort algorithm. It takes 10 seconds for the method to complete given a data set of 1,000,000 integers in random order. What is the expected time for the method to complete given a data set of 2,000,000 items in random order?

P. (Extra Credit 1 point) If a computer account had the user id of `student` what would be the password?

2. (Implementing data structures, 25 points). Write an instance method for a `LinkedList` class named `isSorted`. The method returns `true` if the data in the `LinkedList` is in sorted in ascending order, `false` otherwise. A `LinkeList` with 0 or 1 elements is considered to have its elements in sorted, ascending order.

This `LinkedList` class uses singly linked nodes. Each node stores a single `int`.

This `LinkedList` class contains a reference to the first node in the list and the last node in the list as well as a size variable. The last node in the list's `next` reference is set to `null`. When the list is empty `first` and `last` are set to `null`.

Examples:
`[].isSorted()` returns `true`
`[12].isSorted()` returns `true`
`[1, 1, 1].isSorted()` returns `true`
`[1, 2, 5].isSorted()` returns `true`
`[2, 4, 6, 5, 10, 11].isSorted()` returns `false`
`[5, -1].isSorted()` returns `false`

**You may not use any other methods in the `LinkedList` class, but you may create your own helper methods if you wish.**

**You solution must be O(1) space. In other words you can use an auxiliary array or ArrayList.**

```
public class Node
{     public Node(int value, Node next)

      public int getData()

      public Node getNext()

      public void setValue(int value)

      public void setNext(Node next)
}
```

```
public class LinkedList
{     private Node first; // points to the first node in the list
      private Node last; // points to the last node in the list
      private int size; // size of list
      // when size == 0, first = last = null

      // pre: none
      // post: return true if the elements in this list are
      // sorted in ascending order, false otherwise
      public boolean isSorted(){
      // complete this method on the next page
```

```
// pre: none
// post: return true if the elements in this list are
// sorted in ascending order, false otherwise
public boolean isSorted(){
```

3. (Implementing Data Structures 25 points) Consider an array based list class like the one we developed in class. The class is named `GenericList`. The class uses a native array of `Objects` as its internal storage container. The internal storage container may have extra capacity and thus its length will be greater than or equal to the length of the list that is being represented.

Complete an instance method that swaps the first half of the list with the second half of the list. The relative order of elements in each half is unchanged.

If the list has an odd number of elements the location of the middle element is not altered.

**Your method may not use any other methods in the `GenericList` class, but you may create your own helper methods if you wish. Your solution must be O(1) space. You may not use any other Java classes in your solution.**

Here are some examples of the expected behavior of the method on various lists:

```
[].swapHalves() -> [] // An empty list is unchanged.
[1].swapHalves() -> [1] // A list with one element is unchanged.
[1, 2].swapHalves() -> [2, 1] //
[5, 4, 7].swapHalves() -> [7, 4, 5] // Middle element unchanged.
[5, 4, 7, 8].swapHalves() -> [7, 8, 5, 4] // Notice relative order
[5, 4, 7, 8, 3].swapHalves() -> [8, 3, 7, 5, 4]
[5, 4, 7, 8, 3, 12].swapHalves() -> [8, 3, 12, 5, 4, 7]
```

```
public class GenericList{

     private Object[] myContainer; // internal storage container
     // myContainer is never set to null

     private int size; // number of elements in list


     // pre: none
     // post: first and second halves of the list are swapped
     // per the question statement.
     public void swapHalves(){
          // Complete this method on the next page.
          //
          // Complete this method on the next page.
          //
          // Complete this method on the next page.
          //
          // Complete this method on the next page.
          //
          // Complete this method on the next page.
          //
          // Complete this method on the next page.
```

```
// pre: none
// post: first and second halves of the list are swapped
// per the question statement.
public void swapHalves(){
```

4. (Recursion, 20 points) Assume a 2 dimensional array of integers is used to represent a map of the plants that make up a forest. Each cell of the 2d array represents one unit of area. (The actual units, square meters, square feet, is irrelevant for this problem.) The integer represents the dominate plant growing in that cell (or unit area).

Write a method that, given a starting location in the map, determines how large a contiguous (connected) area the dominate plant at the specified cell occupies.

A given cell has at most four neighboring cells: the cells above, below, to the left, and to the right. These correspond to the directions north, south, east and west. Diagonal cells are **not** considered to be connected.

Here is an example. Assume we have the following 5 by 6, two dimensional array of ints, representing a map of a forest. Each int is a code for the dominate plant in that cell of the forest.

| | 0 | 1 | 2 | 3 | 4 | 5 | column indices |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 5 | 2 | 5 | 9 | 1 | |
| 1 | 5 | 2 | 2 | 5 | 5 | 5 | |
| 2 | 5 | 6 | 7 | 5 | 9 | 9 | |
| 3 | 1 | 5 | 7 | 5 | 7 | 5 | |
| 4 | 5 | 5 | 5 | 5 | 5 | 5 | |
| row indices | | | | | | | |

If the specified coordinates are row 0 and column 0 the dominate plant in that cell is specified by a 2. The cell is a special case because it only has 2 neighbors, the cells to the right and below it. Neither of these cells (at (1,0) and (0,1) ) contain a 2, so the result here is 1.

If the specified coordinates are row 0 and column 2 the dominate plant in that cell is again a 2. This cell has 3 neighboring cells. The cell to the south at (1, 2) also contains a 2. That cell, at (1, 2), has 4 neighboring cells. The one to the north at (0, 2) has already been visited so don't count that one again. But the cell to the west at (1, 2) has not been visited and has a 2 in it as well. Thus the cell starting at location (0, 2) is part of a contiguous area of 3 cells that contain the same plant. The result would be 3.

The largest contiguous area in the example is outlined below.

| | 0 | 1 | 2 | 3 | 4 | 5 | column indices |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 5 | 2 | 5 | 9 | 1 | |
| 1 | 5 | 2 | 2 | 5 | 5 | 5 | |
| 2 | 5 | 6 | 7 | 5 | 9 | 9 | |
| 3 | 1 | 5 | 7 | 5 | 7 | 5 | |
| 4 | 5 | 5 | 5 | 5 | 5 | 5 | |
| row indices | | | | | | | |

Here are some expected results if map equals the 2d array on the previous page. (Do not hard code a solution to the example map!)

```
sizeOfPlantArea(map, 0, 0) -> 1
sizeOfPlantArea(map, 0, 1) -> 1
sizeOfPlantArea(map, 1, 0) -> 2
sizeOfPlantArea(map, 0, 2) -> 3
sizeOfPlantArea(map, 4, 0) -> 14
```

Hint: You may find it useful to create a helper method and use the specified method to kickoff the recursion.

Complete the following method:

```
// pre: map != null, map is a rectangular matrix
// row and col are both inbounds
// post: Return the size of the contiguous growth that the
// starts at the cell specified by row and col.
// The elements of map are not altered as a result
// of this method call.
public int sizeOfPlantArea(int[][] map, int row, int col){
```

```
// more room for question 4 if necessary
```

Scratch Paper

Name:_____

TAs name: _____

Answer sheet for question 1, short answer questions

A. _____

B. _____

C. _____

D. _____

E. _____

F. _____

G. _____

H. _____

I. _____

J. _____

K. _____

L. _____

M. _____

N. _____

O. _____

P. (extra credit) _____