

CS 162: Introduction to Computer Science II
Spring 2007
Midterm #2

Name: _____

Recitation Time: _____

This midterm consists of 7 pages and a total of 45 marks. It is closed book and closed notes. You have 50 minutes to complete this midterm. If you are taking too much time on one question, please move on to other questions on the test.

Section	Marks
Testing	/ 15
Recursion	/ 15
Algorithmic Analysis	/ 15
Total	/ 45

Section I: Testing [15 points]

1. Read through the calculateLetterGrade() method below. The comments above the function provide a description of the functionality of this method. What are tests that you should write in order to test this function thoroughly? Provide English descriptions of your tests. [10 points]

```
/**
 * Calculates the letter grade given the grade represented
 * as a percentage
 * @param grade The percentage form of the grade
 * @return A letter grade (either an A, B or an F)
 *         An A is > 90.0 and <= 100.0
 *         A B is > 80.0 and <= 90.0
 *         An F is >= 0 and <= 80.0
 * @throws Exception If the grade is < 0 or > 100.0
 */
public String calculateLetterGrade(double grade) throws
    Exception {

    if(( grade < 0 ) || ( grade > 100.0 )) {
        throw new Exception("Invalid grade");
    } else {
        if(( grade > 90.0 ) && ( grade <= 100.0 ))
            return "A";
        else if(( grade > 80.0 ) && ( grade <= 90.0 ))
            return "B";
        else
            return "F";
    }
}
```

2. Write down a reasonable set of preconditions and postconditions for the addSaleAmount method() below. [5 points]

```
public class SaleTracker {

    // An array that stores the sales for each store.
    // The array is indexed by the storeID
    private double sales[];

    /**
     * Adds the sales amount to the sales record of the
     * store with the given ID
     * @param storeID The ID of the store
     * @param amount The dollar amount of the sale.
     */
    public void addSaleAmount(int storeID, double amount)
    {
        sales[storeID] = sales[storeID] + amount;
    }

    public SaleTracker()
    {
        sales = new double[100];
        for( int i = 0; i < sales.length; i++ ) {
            sales[i] = 0;
        }
    }
}
```

Section II: Recursion [15 points]

3. Write a recursive function called `removeWhiteSpace(String s)` that takes a string `s` and returns a string with all spaces removed. For example, `removeWhiteSpace("must have coffee")` returns `"musthavecoffee"`. The following pieces of information may be helpful (you may choose to disregard them if you wish):

- The `substring(int beginIndex, int endIndex)` method of the `String` class returns the substring starting at index `beginIndex` and ending at index `endIndex - 1` inclusive. For example, `"hamburger".substring(4, 8)` returns `"urge"`
- To compare if a string `s` is a blank space, use `s.equals(" ");`
- To get the length of a string, use the `String` class' `length()` function.

[10 points]

4. Take a look at the recursive function below. The `isEven(n)` function returns true if `n` is an even number and false otherwise.

```
int puzzle(int n) {
    if (n == 1)
        return 1;
    if ( isEven(n) )
        return puzzle(n/2);
    else
        return puzzle(3*n+1);
}
```

What does `puzzle(3)` return? For full credit on this question, write out the value of the argument `n` passed to `puzzle()` each time the `puzzle()` method is called. [5 points]

Section III: Algorithmic Analysis [15 points]

5. Fill in the blank squares with the appropriate time complexity for the algorithm. Use Big-O notation with n being the size of the array to be sorted or searched. [8 points]

Algorithm	Best Case	Worst Case
Insertion Sort		$O(n^2)$
Selection Sort		$O(n^2)$
Quicksort (using a partition function that picks the first element as the pivot)		
Mergesort		
Linear search (unordered array)	$O(1)$	
Binary Search (ordered array)	$O(1)$	

6. If we call myFunction, how many times does “Hello World” get printed? Express this as a function of n and use Big-O notation. Minor point: the use of $/$ in the line $j = j / 2$ is an integer division. For example, $5 / 2 = 2$ because the value of 2.5 would be “truncated” by ignoring the decimal part. [2 points]

```
public void myFunction (int n) {
    for( int i = 0; i < n; i++ ) {
        for( int j = n; j > 0; j = j / 2 ) {
            System.out.println("Hello World!");
        }
    }
}
```

7. The two bubble sort methods below implement bubble sort so that it sorts an array in ascending order. For example, if you pass it an array with elements 4, 5, 2, 3, 1 both functions will sort the array so that it contains 1, 2, 3, 4, 5.

```
public void bubbleSort1(int[] a) {
    for (int i = (a.length-1); i >= 0; i--) {
        for (int j = 1; j<=i; j++){
            if (a[j-1] > a[j]){
                // Swap elements at j-1 and j
                int temp = a[j-1];
                a[j-1] = a[j];
                a[j] = temp;
            }
        }
    }
}
```

```
public void bubbleSort2(int[] a) {
    for (int i = (a.length-1); i >= 0; i--) {
        Boolean swappedElement = false;
        for (int j = 1; j<=i; j++){
            if (a[j-1] > a[j]){
                // Swap elements at j-1 and j
                int temp = a[j-1];
                a[j-1] = a[j];
                a[j] = temp;
                swappedElement = true;
            }
        }
        if( swappedElement == false ) {
            return;
        }
    }
}
```

In the best case scenario, the array is already sorted in ascending order. Using big-O notation, what is the best case running time for both bubbleSort1 and bubbleSort2? Explain why the best case running times are different for the two bubble sorts. [5 points]