

DUKE UNIVERSITY
Department of Computer Science

Test 1 Review Questions

PROBLEM 1 : (*Short Ones (12 points)*)

A. Consider the following lines of code:

```
Integer i = null;
Integer j = null;
Integer k = null;
i = new Integer(72);
j = i;
k = new Integer(72);
```

What is the value of the following expressions:

- I. $i == j$
- II. $i == k$
- III. $i.equals(k)$
- IV. $72 == j.intValue()$

B. Evaluate each of the following expressions:

- I. $((13 > 0) \ || \ (4 < 3)) \ \&\& \ (21 \% 7 == 0)$
- II. $7 * 8 \% 10 / (-2 * -2.0)$

C. (2) Which of the following functions grows fastest as n grows large:

- I. $n \log n$
- II. $1 + 2 + 3 + \dots + n - 1 + n$
- III. $n \log n^2$
- IV. $1 + 1/2 + 1/4 + \dots + 1/2^{n-1} + 1/2^n$
- V. $n\sqrt{n}$
- VI. There is a tie among two or more functions for fastest growth rate

D. (2) The following table gives approximate running times for a program with n inputs, for various values

n	time
1000	5 seconds
2000	20 seconds
5000	2 minutes
10,000	8 minutes

of this program for $n = 100,000$? Which of the following best describes the likely running time

- A. A few minutes

- B. Half an hour
 - C. A few hours
 - D. Half a day
 - E. A few days
 - F. A few weeks
- G. (3) Bill shows you an algorithm to optimally choose classes for all students at Duke. You, being a good Computer Science student, notice that the big-Oh complexity of the algorithm is $O(2^n)$ where n is the number of students. However, Bill demonstrates the program for a sample of 100 students and it returns the schedules almost immediately.

Bill says that his algorithm is good enough for Duke. He mentions something about Moore's Law and states that "Computers are getting faster at an exponential rate. That is, every 18 months, they double in speed. Even if the program is not fast enough now, it will be soon."

Bill is off a little bit on what Moore's law means. However, given that computers continue doubling in speed every 18 months, will the $O(2^n)$ algorithm ever be practical? Explain why or why not?

- H. (4) Suppose that `b[]` is an array of 100 elements, with all entries initialized to 0, and `a[]` is an array of N elements, each of which is an integer between 0 and 99. What is the effect of the following loop? (select all that apply)

```
for (int j = 0; j < N; j++)
    b[a[j]]++;
```

- I. Sets `b[0]` to 0, `b[1]` to 1, `b[2]` to 2, etc.
 - II. Sets `b[0]` to the number of 0s in `a[]`, `b[1]` to the number of 1s in `a[]`, etc.
 - III. Sets `b[0]` to `a[0]`, `b[1]` to `a[0] + a[1]`, `b[2]` to `a[0] + a[1] + a[2]`, etc.
 - IV. Sets all entries of `b[]` to 1.
 - V. Out-of-bounds array access (i.e. `ArrayIndexOutOfBoundsException`).
 - VI. None of the above.
- I. (1) Name an interesting subarea of computer science. If you spend more than one minute on this problem, you've spent *way* too much time.

PROBLEM 2 : (*Majority (5 points)*)

Write a method `majority` that takes three boolean inputs and returns the most common value. For example, `majority(false, false, true)` should return `false`, while `majority(false, true, true)` returns `true`.

PROBLEM 3 : (*Analyze (18 points)*)

- A. In this problem, you will analyze the big-Oh for 2 solutions sorting an array of integers. Your job is to analyze the two algorithms below, mark the line(s) of code where the majority of time is spent, and give the tight big-Oh bounds with justification. n is the number of elements in the array `a`.
- I. (4) Analyze `sort1`

```

public static void sort1(int[] a) {
    for (int i = 1; i < a.length; i += 1) {
        int x = a[i];
        int j;
        for (j = i; j > 0 && x < a[j-1]; j -= 1)
            a[j] = a[j-1];
        a[j] = x;
    }
}

```

II. (4) Analyze sort2

```

public static void sort2(int[] a )
{
    TreeSet<Integer> set = new TreeSet<Integer>();
    for (int i = 0; i < a.length; i += 1)
        set.add(i);
    int k = 0;
    for (int elem: set)
    {
        a[k] = elem;
        k = k + 1;
    }
}

```

III. (2) In most cases, `sort1` and `sort2` will produce the same correct results. However, in some cases, `sort2` will not sort `a`? Describe when and why `sort2` will not produce the proper result.

B. Below are two examples of counting unique words from class. Your job is to analyze the two algorithms below, mark the line(s) of code where the majority of time is spent, and give the tight big-Oh bounds with justification. n is the number of elements in the array `list`.

I. (4) Analyze uniqueCount

```

public class SortingUniqueCounter implements IUniqueCounter {
    public int uniqueCount(String[] list) {
        Arrays.sort(list);
        String last = list[0];
        int count = 0;
        // Invariant: count is number of unique words in list[0..k]

        for (int k = 1; k < list.length; k++) {
            if (!list[k].equals(last)) {
                last = list[k];
                count++;
            }
        }
        return count;
    }
}

```

II. (4) Analyze uniqueCount

```

public class SlowUniqueCounter implements IUniqueCounter {
    public int uniqueCount(String[] list) {
        int count = 0;
        int diffSize = list.length;

```

```

// Invariant: all words in range list[0..k] are unique
// Elements in list[diffSize..) are duplicates of those in
// list[0..diffSize)

for (int k = 0; k < diffSize; k++) {
    String word = list[k];
    count++;
    for (int j = k + 1; j < diffSize; j++) {
        if (list[j].equals(word)) {
            list[j] = list[diffSize - 1];
            diffSize--;
        }
    }
}
return count;
}
}

```

PROBLEM 4 : (Analyze (8 points))

In this problem, you will analyze the big-Oh for 2 solutions to the Maximum Contiguous Subsequence Problem. Given possibly negative integers A_1, A_2, \dots, A_n , find the maximum value of $\sum_{k=i}^j A_k$. The maximum contiguous subsequence would be zero (the empty sequence) if all integers all negative. For example, if the input is $\{-2, \mathbf{11}, -4, -\mathbf{13}, -5, 2\}$, then the answer is 20 for the subsequence in bold (elements 2-4). Your job is to analyze the two algorithms below, mark the line(s) of code that is executed the greatest number of times, and give the tight big-Oh bounds with justification. n is the number of elements in the array a .

A. (4 points) Analyze maxSubSum1

```

public static int maxSubSum1(int[] a )
{
    int seqStart, seqEnd;
    int maxSum = 0;
    for( int i = 0; i < a.length; i++ )
        for( int j = i; j < a.length; j++ )
        {
            int thisSum = 0;
            for( int k = i; k <= j; k++ )
                thisSum += a[ k ];
            if( thisSum > maxSum )
            {
                maxSum = thisSum;
                seqStart = i;
                seqEnd = j;
            }
        }
    return maxSum;
}

```

B. (4 points) Analyze maxSubSum2

```

public static int maxSubSum2(int[] a )
{

```

```

int seqStart, seqEnd;
int maxSum = 0;
int thisSum = 0;
for( int i = 0, j = 0; j < a.length; j++ )
{
    thisSum += a[ j ];
    if( thisSum > maxSum )
    {
        maxSum = thisSum;
        seqStart = i;
        seqEnd = j;
    }
    else if( thisSum < 0 )
    {
        i = j + 1;
        thisSum = 0;
    }
}
return maxSum;
}

```

PROBLEM 5 : (Pix (17 points))

In this problem, you will create a Pixmap *Command* that changes the target Pixmap by sorting each row of Colors according to the average of its three components (i.e., its grey-scale value). Thus, for each row of the Pixmap, the darker colors should be moved to the left and the lighter colors should be moved to the right. You should not change the value of any individual color, just its position in its row.

- A. (6) To start, you will need to complete the compare method below so that it correctly ranks the two given Color values relative to each other.

```

public class AverageComparator implements Comparator<Color>
{
    private int gray(Color c)
    {
        return (c.getRed() + c.getGreen() + c.getBlue())/3;
    }

    // returns:
    // < 0 if the grey-scale value of c1 is less than c2
    // 0 if the grey-scale value of c1 is equal to c2
    // > 0 if the grey-scale value of c1 is greater than c2
    public int compare (Color c1, Color c2)
    {
        return gray(c1) - gray(c2);
    }
}

```

- B. (8) Complete the execute method below that creates an instance of the AverageComparator when you call the sort method for each row.

Hint: You can sort the elements of an ArrayList using Collections.sort().

```

public class Student extends Command
{
    public Student ()
    {
        super("Sort Colors");
    }

    public void execute (Pixmap target)
    {
        Dimension size = target.getSize();
        AverageComparator comp = new AverageComparator();
        for (int j =0; j < size.height; j += 1)
        {
            // Add row to list
            ArrayList<Color> row = new ArrayList<Color>();
            for (int i=0; i < size.width; i+= 1)
                row.add(target.getColor(i, j));
            // sort list by grayscale (average)
            Collections.sort(row, comp);
            // Copy row back to target
            for (int i=0; i < size.width; i+= 1)
                target.setColor(i,j, row.get(i));
        }
    }
}

```

- C. (3) Given that the original Pixmap is $n \times m$ pixels. What is the big-Oh of your solution? Assume that *Collections.sort* calls the comparator $O(n \log n)$ times.

$O(m \times n \log n)$, Sorting a row with n columns takes $O(n \log n)$ time and there are m rows.

PROBLEM 6 : (*Close to Home (10 points)*)

In this problem, you will write a method `closestToOrigin` that given arrays of x-coordinates and y-coordinates corresponding to points, returns the point that is the minimum distance away from the origin (0,0).

For example, given the arrays

```
closestToOrigin({1, 0, 6, 3, 5}, {10, -12, 4, 4, 5})
```

should return the `Point (3,4)`. The class `Point` is defined as follows:

```

public class Point
{
    public int x;
    public int y;

    public Point(int x, int y){
        this.x = x;
        this.y = y;
    }

    public double distanceFrom(Point p){

```

```

    return Math.sqrt( (x-p.x)*(x-p.x) + (y-p.y)*(y-p.y) );
}
}

```

Complete *closestToOrigin* below.

```

public static Point closestToOrigin (int[] xVals, int[] yVals)
{

```

PROBLEM 7 : (Apart (10 points))

The input file is a text file that contains the information for a particular universe. The first value is an integer N which represents the number of particles. Each line after that has two values indicating the the x - and y - coordinates of each particle.

Given an initialized `Scanner` for reading from the file, complete *farthestApart* so that it prints out the two particles which are the greatest distance apart. You are given a working distance function below.

For example, given the following file:

```

3
0.0 0.0
-1.0 -1.0
10.0 20.0

```

You should print

`(-1.0, -1.0)` and `(10.0, 20.0)` are farthest apart

Notes:

- In the event of ties, you can print any maximum pair.
- Order of the points printed does not matter.
- Do not worry about formatting your numbers (i.e. worrying about the number of trailing zeroes or significant digits).

```

public class Distance {
    // returns Euclidean distance between (x1, y1) and (x2, y2)
    public static double distance(double x1, double y1, double x2, double y2) {
        double dx = x2 - x1;
        double dy = y2 - y1;

        return Math.sqrt( (dx*dx) + (dy*dy) );
    }

    public static void farthestApart(Scanner in)
    {

```

PROBLEM 8 : (Crunch (10 points))

Complete the method *improve* below that replaces all instances of "UNC" in an array with the empty string and puts them at the end of the array. The order of other entries should remain unchanged. Your function must operate in $O(n)$ time where n is the length of the array and cannot create a new array.

For example,

```
improve({"Duke", "UNC", "UNC", "UNC", "Maryland", "Duke", "Duke", "UNC"})
```

should return

```
{"Duke", "Maryland", "Duke", "Duke", "", "", "", ""}.
```

```
String[] improve(String[] data)
{
```

PROBLEM 9 : (*SortByFreqs APT (28 points)*)

The code below correctly solves the *SortByFreqs* APT whose description is the last pages of this test. You will be asked to reason about the code and improvements to it.

```
import java.util.*;

public class SortByFreqs {
    private class FreqCompare implements Comparator<String> {
        private List<String> myList;

        public FreqCompare(String[] a) {
            myList = Arrays.asList(a);
        }

        public int compare(String a, String b) {
            int afreq = Collections.frequency(myList, a);
            int bfreq = Collections.frequency(myList, b);
            int diff = bfreq - afreq;
            if (diff != 0)
                return diff;
            return a.compareTo(b);
        }
    }

    public String[] sort(String[] data) {
        FreqCompare comp = new FreqCompare(data);

        // Create set of unique words from data
        List<String> list = Arrays.asList(data);
        HashSet<String> unique = new HashSet<String>(list);

        String[] words = new String[unique.size()];

        // Copy unique set into words array
        int k = 0;
        for (String w : unique) {
```

```

        words[k++] = w;
    }

    Arrays.sort(words, comp);
    return words;
}
}

```

- A. (3) The line below calculates the number of times `a` appears in `myList`. What is its big-Oh complexity when `myList` contains n elements. Justify your answer.

```
int afreq = Collections.frequency(myList, a);
```

- B. (2) Using big-Oh, describe the complexity of the method `compare`.
- C. (3) The big-Oh complexity of `Arrays.sort` is $O(n \log n)$ for an n -element array. Assume there are n items in `data` of which u are unique. What is the big-oh complexity of the complete solution to *SortByFreqs* on the previous page? Briefly justify your answer.
- D. (3) In what case would performance be the worst (in big-Oh terms)?
- E. (3) Describe how the values in the returned array will change if the line

```
int diff = bfreq-afreq;
```

is changed to the following. Be brief and precise, not thorough.

```
int diff = afreq-bfreq;
```

- F. (4) The `compare` method `FreqCompare` class could be changed by replacing calls to `Collections.frequency` with look-ups to appropriate maps as shown below. Describe why this version of `compare` is better than the original in terms of being more efficient.

```

private class FreqCompare implements Comparator<String> {
    private HashMap<String, Integer> myMap;

    public FreqCompare(String[] a) {
        // To be filled in..
    }

    public int compare(String a, String b) {
        int afreq = myMap.get(a);
        int bfreq = myMap.get(b);
        int diff = bfreq - afreq;
        if (diff != 0)
            return diff;
        return a.compareTo(b);
    }
}

```

- G. (6) Complete the constructor for `FreqCompare` below, so that the map is correctly filled in. that is, `myMap` should map from entries in the original array to the number of times they occur.

```
public FreqCompare2(String[] a) {
```

- H. (4) What is the big-Oh complexity of the *SortByFreqs* solution using a map? How does it compare to the previous solution using a list and `Collections.freq`?