Topics

- Exam I and II stuff, plus...
- List-Based Algorithms
  - Specific algorithms you may be tested on include: linear search, binary search, selection sort, and bubble sort. Understand their mechanisms and complexities.
  - Know how to write various list manipulation algorithms. Here are some examples: reversing a list, palindrome, printing unique pairs of elements, etc.
- I/O
  - Know how to read and write from files using Scanner and PrintWriter classes, respectively.
  - Understand how to accept and manipulate user input via keyboard (Scanner).
- Complexity Analysis
  - Understand how the time complexity for given algorithms is derived.
  - Understand and be able to give examples of algorithms belonging to various complexity classes: constant, logarithmic, linear, quadratic, and exponential.
  - Be able to analyze the best/worst/average-case complexity of any given algorithm as a function of the problem size.
  - You do not need to know how to analyze recursive algorithms.
- Recursion
  - Know how to read (and trace) recursive methods.
  - Be able to fix bugs in recursive methods.
  - With some guidance, be able to solve certain problems recursively instead of iteratively (with loops).
- Misc.
  - Information hiding: purpose of visibility modifiers.
  - The purpose and effect of the static keyword.
  - Significance of the main() method.
- Remember to bring your APIs to the exam! The following are acceptable: Scanner/PrintWriter, ArrayList, HashMap, String.
Practice Problems

1. Go collect some data from Twitter. Save each tweet as a line to a file. Use your TweetParser homework assignment (or my solution) to process the statistics for all tweets in the file. You can compare your solution to mine (see: File I/O TweetParser).

   Now write a class with a `main()` method that allows users to input tweets repeatedly as strings. Process each tweet until the user decides to stop, and print out the stats collected on the entered tweets.
2. When using binary search, and the array is not sorted, what can we expect to happen? (a) the program will crash, (b) you will never find the value you are looking for even if it does exist in the array, or (c) sometimes the search will be successful, but sometimes it will fail even if the value does exist in the array.

3. Consider the following algorithm that searches an array of Strings for duplicate elements.

```java
public static boolean hasDuplicates(String[] list)
{
    for (int i = 0; i < list.length; i++) {
        for (int j = i + 1; j < list.length; j++) {
            //found one!
            if (list[i].equals(list[j])) {
                return true;
            }
        }
    }
    return false;
}
```

Assuming that $N$, the list size, is arbitrarily large, analyze the best and worst case complexities for this algorithm. Explain the conditions under which each case would be observed, and give the complexity as a mathematical expression in terms of $N$. 

4. Write a static method called `flatten()` that inputs a 2D array of integers and returns a 1D array of integers. This method “flattens” the data in a 2D array with a $M$ rows and $N$ columns into a 1D array of size $MN$. For instance:

```java
int[][] table = {
    {5,2,0},
    {6,7,1},
};
int[] line = flatten(table);
> [Contents of line]
> [5, 2, 0, 6, 7, 1]
```

Assuming that $M$ and $N$ can be arbitrarily large, analyze the best, worst, and average-case complexities for this algorithm. Explain the conditions under which each case would be observed, and give the complexity as a mathematical expression in terms of $M$ and $N$.

5. Write a static method called `rotateRight()` that inputs a 1D array of integers. This method should return a 1D array of integers, where each element has been shifted one position to the right. The exception is the last element, which should move to the front of the array. For instance:

```java
int[] list = {1,2,3,4};
list = rotateRight(list);
> [Contents of list]
> [4, 1, 2, 3]

int[] list2 = {1};
list2 = rotateRight(list2);
> [Contents of list2]
> [1]
```

Assuming that $N$, the list size, is arbitrarily large, analyze the best, worst, and average-case complexities for this algorithm. Explain the conditions under which each case would be observed, and give the complexity as a mathematical expression in terms of $N$. 
6. Write a static method `boolean exists(int[] list, int k, int m)` that inputs a 1D array of integers, a search key `K`, and a positive integer `M(M ≥ 0)`. This method should determine if there are at least `M` copies of `K` in the list. For instance:

```java
int[] list = {5,1,2,3,4,2,3};
System.out.println(exists(list, 2, 1));  // true
System.out.println(exists(list, 2, 2));  // true
System.out.println(exists(list, 10, 0)); // true
System.out.println(exists(list, 5, 2));  // false
```

Assuming that `N`, the list size, is arbitrarily large, analyze the best, worst, and average-case complexities for this algorithm. Explain the conditions under which each case would be observed, and give the complexity as a mathematical expression in terms of `M` and `N`. Furthermore, does list order matter, and would it affect the best/worst/average cases?

---

7. **A prime number is an integer ≥ 2 that’s only evenly divisible by 1 and itself (e.g., 2, 3, 5, 7...). Consider the following method which inputs an integer `N` and determines whether it is prime.**

```java
public static boolean isPrime(int N) {
    int divisor = 2;
    while (divisor < N && (N % divisor) != 0) { // try each divisor...
        divisor++;
    } // if divisor is N then all numbers between 2 and N-1 failed to divide N evenly, meaning N must be prime
    return (divisor == N);
}
```

Analyze the best, worst, and average-case complexities for this algorithm. Explain the conditions under which each case would be observed, and give the complexity as a mathematical expression in terms of `N`, the number that is being checked. For the average case, just explain whether you’d expect it to be closer to the best case or worst case.

**Hint:** For the average case, is an “average” integer prime?
8. **Consider the following recursive method.**

```java
public static int mystery(int[] list, int head) {
    if (head == list.length - 1) {
        if (list[head] % 2 == 0) {
            return 1;
        } else {
            return 0;
        }
    }
    else if (list[head] % 2 == 0) {
        return 1 + mystery(list, head + 1);
    } else {
        return 0 + mystery(list, head + 1);
    }
}
```

(a) Indicate directly on the code the base case(s) and the recursive case(s).
(b) What does this method return if `list = {3, 3, 4, 2}` and `head = 0`?
(c) Is it possible to cause an infinite recursion? If so, what input(s) would trigger it? Otherwise, state why not.
(d) In a couple brief sentences, describe what this method does.
9. The *exponentiation* of $a$ to the power of $b$ can be expressed as follows:

$$a^b = a \times a \times \ldots \times a$$

Provide a **recursive** method, `exp()`, that inputs two integers, $a$ and $b$, and returns the value of $a^b$. You may assume that $b \geq 0$. Recall from algebra that when $b = 0$, then $a^b = 1$.

```plaintext
> exp(2,3)
  8 (int)

> exp(3,3)
  27 (int)

> exp(10,0)
  1 (int)
```

10. The Collatz conjecture concerns with what happens when we take a positive integer $n$ and apply the following repeatedly:

- If $n$ is odd, then $n = 3n + 1$
- Otherwise, $n = n/2$

The conjecture, which has yet to be proven, states that the sequence will always converge to 1. For instance, `collatz(35)` will produce the sequence: 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1. Write a recursive method `void collatz(int n)` (without using loops) that prints the Collatz sequence for a given $n$. The Collatz sequence can be printed as follows,

- Print the current value of $n$
- If $n$ is odd, then print the Collatz sequence for $n = 3n + 1$
- Otherwise, print the Collatz sequence for $n = n/2$