CSCI 161 Introduction to Computer Science

Review Guide 3

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`).
- Code Writing
  - Know how to parse/tokenize Strings.
  - Know how to write methods that utilize HashMaps for problem solving.
  - Know how to write various list manipulation algorithms. Here are some examples: reversing a list, palindrome, printing unique pairs of elements, *etc.*
- 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. **[On your own]** 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. **[Code Management]** What is the significance of declaring a method to be `static`? When might you consider creating a static method?

3. **[Code Management]** What is an `enum` class used for?

4. **[Search]** 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.

5. **[Sorting]** The bubble sort algorithm is given below:

```
1  public void bubbleSort() {
2      boolean swapped = true;
3      for (int i=0; swapped && i < A.length; i++) {
4          swapped = false;
5          for (int j=1; j < A.length - i; j++) {
6              if (A[j-1] > A[j]) {
7                  int temp = A[j-1];
8                  A[j-1] = A[j];
9                  A[j] = temp;
10                 swapped = true;
11             }
12         }
13     }
14 }
```

   (a) What are the correct contents in `A` after each pass, if `A` is input as: 15,20,18,10? (4pts)

      i. $15, 10, 20, 18 \rightarrow 15, 10, 20, 18 \rightarrow 15, 10, 18, 20 \rightarrow 10, 15, 18, 20$

      ii. $15, 18, 10, 20 \rightarrow 15, 10, 18, 20 \rightarrow 10, 15, 18, 20$

      iii. $15, 20, 10, 18 \rightarrow 10, 15, 20, 18 \rightarrow 10, 15, 20, 18 \rightarrow 10, 15, 18, 20$

      iv. $15, 18, 10, 20 \rightarrow 10, 15, 18, 20$

   (b) With regards to time-complexity, the best case for bubbleSort() is observed when the list `A` is already sorted in ascending order. Explain why.

6. **[Complexity]** What does it mean when we say that an algorithm's time complexity is *consistent*? Why is that significant?

7. **[Complexity]** Consider the following algorithm that searches an array of Strings for duplicate elements.

```
1  public static boolean hasDuplicates(String[] list)
2  {
3      for (int i = 0; i < list.length; i++) {
4          for (int j = i + 1; j < list.length; j++) {
5              //found one!
6              if (list[i].equals(list[j])) {
7                  return true;
8              }
9          }
10     }
11     return false;
12 }
```

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. You can ignore the average case.

8. **[Complexity]** In class, we saw that the time it takes to execute the `binarySearch` method depends on the number of elements in the input array. Suppose that it takes 3 seconds on average to complete a `binarySearch` of an array with 1024 elements. Approximately how long would it take to search 16,777,216 elements, and why? In explaining your answer, make specific reference to the mathematical formula(s) we studied in class. Recall that $\log_a b = \frac{\ln b}{\ln a}$.

9. **[Code Writing (ArrayLists)]** Write a `static` method `ArrayList<Integer> diff(ArrayList<Integer> list1,`
`ArrayList<Integer> list2)` that returns a list of String that appear in `list1` but not in list2. The returned list should not contain duplicated elements. For instance:

```
// assume ArrayLists a and b are already defined

System.out.println(a.toString());
> [5,1,2,3,4,2,3]

System.out.println(b.toString());
> [9,5,0,3]

System.out.println(diff(a,b).toString());
> [1,2,4]

System.out.println(diff(b,a).toString());
> [9,0]

System.out.println(diff(a,a).toString());
> []
```

10. **[Code Writing (ArrayLists)]** Write a `static` method `ArrayList<Integer> intersect(ArrayList<Integer> list1, ArrayList<Integer> list2)` that returns a list of common integers in the input lists. The returned list should not contain duplicated elements. For instance:

```
// assume ArrayLists a and b are already defined

System.out.println(a.toString());
> [5,1,2,3,4,2,3]

System.out.println(b.toString());
> [9,5,0,3]

System.out.println(intersect(a,b).toString());
> [5,3]

System.out.println(intersect(a,a).toString());
> [5,1,2,3,4]
```

11. **[Code Writing (String Parsing, Arrays, HashMaps)]** Write a `static` method `HashMap<String,Double> gobble(String sentence)` that inputs a list of strings and returns a HashMap keyed on the first letter of each word in the sentence, and maps to the average length of the words in the sentence starting with that letter. You may assume that the the input will be without punctuation. (Hint: declare two HashMaps – one that keys on each word and maps to its length, and one to return). You may also want to write a helper method that determines the number of words in the given that starts with a given letter. Sample output:

```
System.out.println(gobble("AA AAAAAA AA AA AA"));
> "a" = 2.8 (due to (2+6+2+2+2)/5)

System.out.println(gobble("b aba abaa baaa ba aaaba"));
> "b" = 2.33333
> "a" = 4.0

System.out.println(gobble("That flashy fox jumps over the flying flamingo"));
> "t" = 3.5
> "f" = 5.75
> "j" = 5.0
> "o" = 4.0
```

12. **[Recursion]** The *exponentiation* of a to the power of b can be expressed as follows:

$$a^b = \underbrace{a \times a \times ... \times a}_{b \text{ times}}$$

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 $a^b = 1$.

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

> exp(3,3)
27   (int)

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