Version: ~~3-21-2019 8:44 AM 3-22-2019 10:59 AM~~ 3-23-2019 ~~10:31 AM~~ 4:46 PM

CS 2223 D19 Term. Homework 2

Homework Instructions

- This homework is to be completed individually. If you have any questions as to what constitutes improper behavior, review the examples as I have posted online http://web.cs.wpi.edu/~heineman/html/teaching_/cs2223/d19/#policies.
- Due Date for this assignment is 2PM March 29th. Homeworks received after 2PM receive a 25% late penalty. Homeworks received after 6PM will receive zero credit.
- Submit your assignments electronically using the canvas site for CS2223. Login to canvas.wpi.edu and locate HW2. You must submit a single ZIP file that contains all of your code as well as the written answers to the assignment.
- All of your Java classes must be defined in a packager USERID where USERID is your CCC user id.

Questions

1. Mathematical Analysis [12 points] (4 bonus points)
2. Working with linked lists [12 points] (1 bonus point)
3. Data Type Exercise [76 points] (1 bonus point)

Getting Started

Copy the following files into your USERID.hw2 package:

- `algs.hw2.Composite`
- `algs.hw2.FindRepeater`
- `algs.hw2.Node`
- `algs.hw2.TertiarySearch`
- `algs.hw2.TestComposite`
- `algs.hw2.PerformanceTest` ← Use to validate you achieve expected performance

If you are attempting the bonus questions, then copy the following as well

- `algs.hw2.BonusComposite`
- `algs.hw2.TestBonusComposite`

Version: ~~3-21-2019 8:44 AM 3-22-2019 10:59 AM~~ 3-23-2019 ~~10:31 AM~~ 4:46 PM

Q1. Mathematical Analysis [12 points]

TERTIARYARRAYSEARCH (found in **`algs.hw2.TertiarySearch`**) attempts to out-perform BINARYARRAYSEARCH by subdividing a sorted collection into "thirds" and searching for a value according to logic modified to take this subdivision into account.

To see how this might work, consider looking for the value 22 in a sorted array containing 8 values:

8	11	14	17	22	24	35	49
lo		left			right		hi

The array is subdivided into "thirds" and the algorithm first inspects position `a[left]=14` which is smaller than 22, so it checks position `a[right]=24` to see that is greater than 22, which guarantees the value could only be in the range `a[left+1 .. right-1]` and the search proceeds in similar fashion to BINARYARRAYSEARCH.

Your program should create arrays of size N=1 to 20 containing (in ascending order) the integers from 0 to N-1. Then it should search for each of the N values, using `find(ar, target)`, recording the number of inspections needed to locate each item, which is computed after the completion of `find`.

Copy **`algs.hw2.TertiarySearch`** into your USERID.hw2 package and modify it to output the following:

```
N      Max    Total  Full Counts
1      1      1      1
2      2      3      1,2
3      3      6      1,2,3
4      3      8      2,1,3,2
5      3      11     2,1,3,2,3
6      4      15     2,1,3,2,3,4
7      4      18     2,3,1,3,4,2,3
8      4      22     2,3,1,3,4,2,3,4
9      5      27     2,3,1,3,4,2,3,4,5
10     5      31     2,3,4,1,3,4,5,2,3,4
11     5      36     2,3,4,1,3,4,5,2,3,4,5
12     5      40     2,3,4,1,3,4,5,2,4,3,5,4
13     5      43     3,2,4,3,1,4,3,5,4,2,3,4,5
14     5      47     3,2,4,3,1,4,3,5,4,2,4,3,5,4
15     5      52     3,2,4,3,1,4,3,5,4,2,4,3,5,4,5
16     5      56     3,2,4,3,4,1,4,3,5,4,5,2,4,3,5,4
17     5      61     3,2,4,3,4,1,4,3,5,4,5,2,4,3,5,4,5
18     6      67     3,2,4,3,4,1,4,3,5,4,5,2,4,3,5,4,5,6
19     6      72     3,2,4,3,4,5,1,4,3,5,4,5,6,2,4,3,5,4,5
20     6      78     3,2,4,3,4,5,1,4,3,5,4,5,6,2,4,3,5,4,5,6
```

In this table, each row has a value N, the computed maximum (i.e., the worst case to find one of the values from 0 to N-1), the total (of the values in FullCounts0, and then N numbers separated by commas, which reflect the individual number of array inspections needed to locate a_k using TERTIARYARRAYSEARCH

for k=0 to N-1. For example, with N=4 elements, you start TERTIARYARRAYSEARCH with lo=0 and hi=3. The first position searched is `left = lo + (hi-lo)/3 = 1`.

	1		

If the value being searched turns out to be smaller than a[1], then you would recursively check `a[lo..left-1]` and there is only one element there, which is found in just one additional array inspection, resulting in a total of 2.

2	1		

If the value being searched is larger than a[1], then you would set `right=left + len + 1 = 3` we would inspect a[3] using a second comparison.

2	1		2

If the value being searched is smaller than a[3], then recursively check `a[left+1 .. right-1]` which results in a[2] requiring a total of 3. Note that these values match the output of the row marked "4". Your program is done when it outputs the above table. The column labeled **Max** contains the largest value in the **Full Counts** column, while ~~average~~ Total contains the Total ~~average (to 2 digits of precision)~~ of the values in the Full Counts column.

Bonus Question 1.1 (1pt.)

Can you come up with a formula that predicts when Max increases? For example, in your table, Max increases at rows 1, 2, 3, 6, 9, 18, 27…

Bonus Question 1.2 (1pt.)

Does TERTIARYARRAYSEARCH outperform BINARYARRAYSEARCH on average? That is, provide additional code to report on the different average number of array inspections on searching for target values for N=3^k for k=1 to 10. What is your assessment? Please provide a definitive empirical statement that relates the average # array inspections for TERTIARYARRAYSEARCH to the average number of array inspections for BINARYARRAYSEARCH.

Bonus Question 1.3 (1pt.)

Given a series of values A_0, A_1, A_2, …, one can sometimes construct a recursive formula for A_n which is different from a closed formula because it can refer to prior terms in the series. For example, the Fibonacci numbers (1, 1, 2, 3, 5, 8, 13, 21, …) are generated by adding two consecutive numbers together to generate the next number in the series (1+1 = 2, 1+2 = 3, 2+3 = 5, …). A recursive formula for Fibonacci is defined as:

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$

Version: ~~3-21-2019 8:44 AM 3-22-2019 10:59 AM~~ 3-23-2019 ~~10:31 AM~~ 4:46 PM

For this bonus question, determine a recursive formula for S_n = the total number of array inspections for n > 1. For this recursive formula, set the following:

- $S_0 = 0$
- $S_1 = 1$

Bonus Question 1.4 (1pt.)

Can you prove that $F_n + F_{n+3} = 2*F_{n+2}$ for $n \geq 0$?

Version: ~~3-21-2019 8:44 AM 3-22-2019 10:59 AM~~ 3-23-2019 ~~10:31 AM~~ 4:46 PM

Q2. Working with Linked Lists (12 pts)

You are given an array containing *n* integers in an arbitrary arrangement. Your goal is to find a *repeater*, that is, an element that is duplicated at least $\frac{n}{2} + 1$ times. For example, in the following array containing eight elements, 2 is a repeater:

[2, 1, 2, 3, 2, 2, 2, 1]

Naturally, there can be no more than a single repeater in an array.

Copy the **algs.hw2.FindRepeater** class into your `USERID.hw2` directory and modify the `find(int[] array)` method to properly return the repeater value or throw a `RuntimeException` if no such value exists in the array.

Your implementation must use the provided `Node` class to construct linked lists.

The goal is to have the find() method construct a linked list of values (with their associated counts) for the array that is passed into the function. This linked list is created anew inside of find whenever it is called. As you process the elements in the array, build up the linked list and modify its values as appropriate. I will discuss in class on Monday.

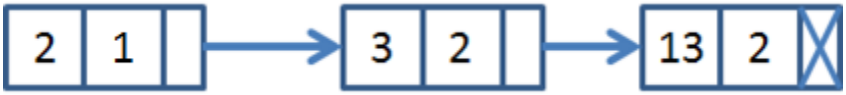
Bonus Question 2.1 (1 pt)

For this question you are to count the number of times you compare two values. Assuming that $n = 2^k$, can you determine – in the worst case – the maximum number of comparisons needed to determine the repeater (if it exists). Provide a sample array of size 8 that offers the "worst case" behavior for this algorithm. Determine which is more costly: (a) the maximum number of comparisons to find a repeater in an array that contains a repeater; or (b) the maximum number of comparisons to try to find a repeater in an array that **does not** contain a repeater.

Version: ~~3-21-2019 8:44 AM 3-22-2019 10:59 AM~~ 3-23-2019 ~~10:31 AM~~ 4:46 PM

Q3. Data Type Exercise (76 pts)

This assignment gives you a chance to demonstrate your ability to program with Linked Lists. You are to implement the following data type *Composite* which represents a positive integer greater than 1. According to the Fundamental Theorem of Arithmetic, every integer greater than 1 is either a prime number itself or can be represented as the product of prime numbers. For example, $3042 = 2x3x3x13x13$ which can be written more concisely as $3042 = 2x3^2x13^2$. Using a linked list representation, this value would be stored in a linked list with three nodes:



Each node contains a factor (left-box) and a power (right-box). The factors must be stored in **ascending** sorted order, and each power value is > 0. Note for completeness, a Composite object whose `head` is `null` is treated as the value 1.

This assignment will prove to be a challenging programming assignment. Do not procrastinate!

```
package USERID.hw2;

public class Node {
    long       factor;
    int        power;
    Node       next;

    public Node (long factor, int power)
    public String toString()
}

public class Composite {
  Node head;    // first factor in linked list

  // operations that are independent of the size of the Composite
  public boolean isUnit() { ... }
  public boolean isPrime() { ... }

  // operations that are dependent (in some way) on N. See documentation
  public Composite(BigInteger val) { ... }
  public String toString() { ... }
  public boolean equals(Object o) { ... }
  public BigInteger value() { ... }
  public Composite add(Composite comp) { ... }
  public Composite multiply(Composite comp) { ... }
  public Composite gcd(Composite comp) { ... }
  public Composite lcm(Composite comp) { ... }

  // static method whose performance depends on the value of num
  public static Composite factorize(BigInteger num) { ... }
```

Copy `algs.hw2.Composite` into USERID.hw2 and complete its implementation, which must conform to performance specifications that are included in the sample code. More documentation is found in the

sample file. In all of the performance specifications, N refers to the total number of factors in a Composite object.

We will validate the output against the set of test cases in **TestComposite** that we develop for the grading. Individual breakdown of points is found on the rubric. There is a **PerformanceTest** included which reveals the total time to compute $(n!)^2$ for standard values of n (to run this performance test, copy it to your `USERID.hw2` package and execute).

This assignment is challenging if you have not programmed extensively with linked lists. Do not wait. Get started on this assignment as soon as you can.

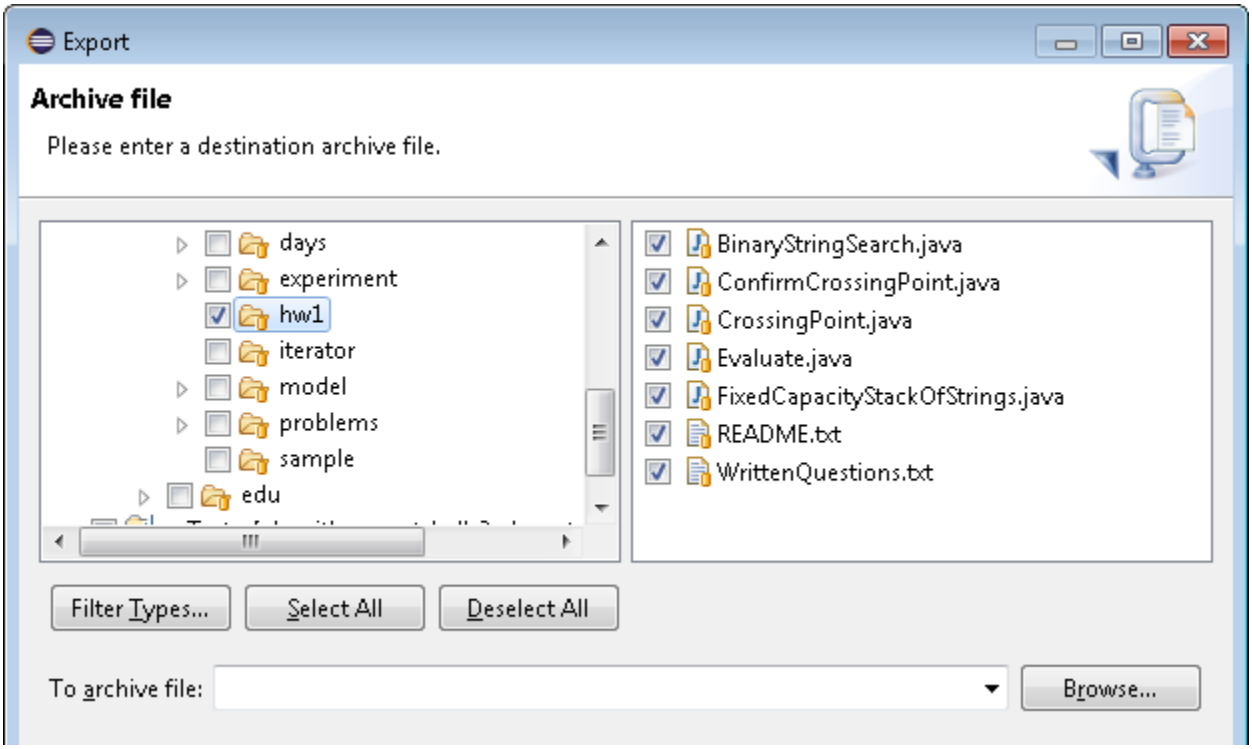
Bonus Question 3.1. (1 pt) Complete the `maximumRoot()` method in the `BonusComposite` class. You can copy `algs.hw2.TestBonusComposite` to your `USERID.hw2` package to validate.

Version: ~~3-21-2019 8:44 AM 3-22-2019 10:59 AM~~ 3-23-2019 ~~10:31 AM~~ 4:46 PM

Submission Details

Each student is to submit a single ZIP file that will contain the implementations. In addition, there is a file "WrittenQuestions.txt" in which you are to complete the short answer problems on the homework.

The best way to prepare your ZIP file is to export your entire **`USERID.hw2`** package to a ZIP file using Eclipse. Select your package and then choose menu item "**Export…**" which will bring up the Export wizard. Expand the **General** folder and select **Archive File** then click **Next**.



You will see something like the above. Make sure that the entire "hw2" package is selected and all of the files within it will also be selected. Then click on **Browse…** to place the exported file on disk and call it USERID-HW2.zip or something like that. Then you will submit this single zip file in canvas.wpi.edu as your homework2 submission.

Addendum

If you discover anything materially wrong with these questions, be sure to contact the professor or TA/SAs posting to the discussion forum for HW2 on piazza;

When I make changes to the questions, I enter my changes in red colored text as shown here.

Change Log

1. Fixed the linked list image for the last question to show that the exponent for the factor 2 is 1.

Version: ~~3-21-2019 8:44 AM 3-22-2019 10:59 AM~~ 3-23-2019 ~~10:31 AM~~ 4:46 PM

2. Added comment in Composite code to remind everyone **not** to store a BigInteger value of the Composite number; that is, only maintain the linked list of factors/exponents.
3. Added the 'head' field into the code which is clearly stated on the homework description.
4. Question 1 computes the Total of the numbers in Full Counts; I mistakenly refer to the average later in the question, and this has been fixed.