Version: ~~April-4-2019 1:45 PM 8:50 PM~~ April 5th 1:30 PM

CS 2223 D19 Term. Homework 3

This homework covers material that extends back to HW2. Based on the performance of the midterm, I think this review is worthwhile.

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 April 12th. 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. Submit your homework under "HW3". 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.hw3 where USERID is your CCC user id (i.e., your email address without the @wpi.edu).

Version: ~~April-4-2019 1:45 PM 8:50 PM~~ April 5th 1:30 PM

Q1. HeapSort Empirical Evaluation (25 pts)

Algorithm 2.7 in Sedgewick, Heapsort, shows how to use a heap to sort a comparable array. The code is provided for you in **algs.hw3.Heap**. The first step is to construct a heap from a Comparable[] array. This takes place in the constructHeap(a) method.

```
// construct heap from the raw array of which we know nothing.
int n = a.length;
for (int k = n/2; k >= 1; k--) {
  sink(a, k, n);
}
```

If you look at this code with an eye towards its performance, it sure looks like the **for** loop will execute n/2 times (which means its performance is linearly dependent on the size of the array). You also know that the **sink** method behavior (in the worst case) is directly proportional to the number of elements in the heap. Thus, at first glance, it looks like this behavior will be proportional to ~ (N*log N)/2.

It turns out that you can mathematically prove that the performance constructHeap is in direct proportion to N alone. Your task is to instead count the number of comparisons and exchanges, and validate the proposition (page 323) that it will, in fact, require fewer than 2N compares and fewer than N exchanges to construct a heap from N items.

Copy algs.hw3.Heap into USERID.hw3 and modify it to record empirical results. For the domain of data, use uniformly computed random numbers from 0 to 1 (such as you can generate from StdRandom.uniform()), and generate a table of results (showing N, # of exchanges, and #comparisons) for N=16, 32, 64, … 512. For each size N, run T=100 trials, and record the maximum number of exchanges and comparisons you witnessed **solely during the constructHeap construction**.

Do your empirical results support the proposition? Explain why or why not.

~~Your~~ Copy the Question1 class to USERID.hw3 and modify it so its output should look something like this:

```
N       MaxComp       MaxExch
16      22            8
32      xxx           yyy
64      xxx           yyy
128     xxx           yyy
256     xxx           yyy
512     xxx           yyy
```

Version: ~~April-4-2019 1:45 PM 8:50 PM~~ April 5th 1:30 PM

Q2. Recurrence Relationship (10 pts)

Recurrence relationships are key to understanding the mathematical modeling behind most algorithms, whether iterative, as in Selection Sort, or recursive, as in Merge Sort. Solving these explains the inner workings of the major algorithmic performance families that we have discussed in class: Logarithmic, Linear, Linearithmic, Quadratic. For all problems, assume that the value of N is a power of 2, or $2^k = N$.

On page 272-273, you can see Proposition F which presents **two** recurrence formulae (or recursive formulae) for the # of compares needed to sort an array of length N. There are two formulae because the upper bound (worst case) and lower bound (best case) are different. If there is no difference between the upper and lower bound, then the same recurrence formula is used for both the best case (lower bound) and worst case (upper bound).

Binary Array Search on N ordered values (**`algs.hw3.RecursiveBinaryArraySearch`**)

Let C(N) be the number of times you compare target with an element from array. Write a recurrence relationship

[2 pts.] Write C(N) as a recursive formula

[4 pts.] Solve C(N) in closed form so it can be written without using recursion. Use the telescoping approach I have presented in lecture.

[2 pts.] What is the lower bound of C(N), or in other words, the best case

[2 pts.] What is the upper bound of C(N), or in other words, the worst case

Version: ~~April-4-2019 1:45 PM 8:50 PM~~ April 5th 1:30 PM

Q3. Sorted Linked Lists (25 pts).

The `SequentialSearchST` class (day 12) could improve its performance of its key operations **if** the keys were stored in the linked list in ascending sorted order. Your task for this question is to copy `SequentialSearchSortedST` into your USERID.hw3 package and modify it to maintain the linked list of keys in ascending sorted order.

Copy the `SeparateSortedChainingHashST` into your USERID.hw3 package since it is the class which uses these (now sorted) `SequentialSearchST` objects.

Note to run the final timing example, you will need to copy the **`words.english.txt`** file into your MyCS2223 project. **Note: do not include `words.english.txt` when submitting your homework since it is a 3.3MB file!**

Report on the results of the timing experiment. What is your conclusion regarding the benefits of sorting the linked lists within the `SeparateSortChainingHashST`.

Version: ~~April-4-2019 1:45 PM 8:50 PM~~ April 5th 1:30 PM

Q4. Binary Search Trees (45 pts).

Using Binary Search Trees to build a Symbol Table that counts the number of occurrences of unique words in *The Tale Of Two Cities* by Charles Dickens. You will need to copy `tale.txt` from the **Algos-D19** git repository to your MyCS2223 project as a top-level file. Just make sure you don't submit this as part of your HW3 solution. Copy `algs.hw3.Question4` into your `USERID.hw3` package and modify it for this problem.

Note that you must use the adapted BST class with the following type signature:

```
public class BST<Key extends Comparable<Key>, Value extends Comparable<Value>>
```

because then you know that both the Key and Value classes are comparable with each other. Copy this file from `algs.hw3.BST` into your `USERID.hw3` package and modify it directly for this assignment.

Q4.1 (10 points) Complete the method that counts the number of leaves in a binary tree:

```
public int numLeaves() { … }
```

You can add any number of helper methods as part of this assignment.

Q4.2 (10 points) Complete the method that returns the Key whose Value is highest.

```
public Key maxValue() { … }
```

You can add any number of helper methods as part of this assignment.

Q4.3 (20 points) `Question4` should construct a BST from the *Tale of Two Cities* and outputs the 10 most common words (together with their corresponding appearance counts). It does this by repeatedly deleting the key with maximum count from the BST and the recomputing the (new) Key string with maximum value.

Version: ~~April-4-2019 1:45 PM 8:50 PM~~ April 5th 1:30 PM

Q4. Bonus Questions (1 pt each)

I will post bonus questions here

Change Log

1. Reminded students to copy Question1 from algs.hw3 into your USERID.hw3 package
2. Reminded students to do same thing for algs.hw3.Heap