~~Version: 4-5-2018 10:00 AM~~ 4-8-2018 10:27 PM

CS 2223 D18 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.

Note: I am releasing this homework even though I am still formatting the 4th and final question of this assignment. That will be completed Thursday April 5th. In the meantime, you can get started on the first three questions.

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/d18/#policies.
- Due Date for this assignment is 2PM April 13th. 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).

Q1. Stack Exercise (24 pts)

Suppose that a client performs an intermixed sequence of stack *push* and *pop* operations. The push operations put the integers 9 descending to 0 **in order onto the stack**; the pop operations print out the value popped from the stack. Which of the following sequences **can** occur, and which ones **cannot** occur.

(a) 8 7 6 5 4 3 0 1 2 9
(b) 5 3 1 2 4 6 7 0 9 8
(c) 7 4 3 2 5 1 0 6 8 9
(d) 5 6 7 8 9 0 1 2 3 4
(e) 5 6 7 8 9 4 3 2 1 0
(f) 9 5 3 4 6 1 8 2 7 0
(g) 8 5 2 0 1 3 4 6 9 7
(h) 7 8 5 6 3 4 1 2 0 9

For example, with the input (note that the digits 9 to 0 always must appear in descending order in the input):

9 8 7 - - 6 5 - 4 - - 3 - 2 1 - - - 0 -

the resulting output is:

~~Version: 4-5-2018 10:00 AM~~ 4-8-2018 10:27 PM

7 8 5 4 6 3 1 2 9 0

For the sequences that you believe cannot occur, provide a brief explanation.

You can experiment with stack using the algs.hw3.Stack class. Just execute it and type in input, as shown above, and the results will be displayed in the Console.

Q2. 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 first few lines of the sort 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 can behavior (in the worst case) 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 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.

You can instrument the Heap class to store the experimental result. For the domain of data, use uniformly computed random numbers from 0 to 1, and generate a table of results (showing N, # of exchanges, and #comparisons) for N=16, 32, 64, … 512. For each size N, run T=10 trials, and record the maximum number of exchanges and comparisons you witnessed **solely during the buildHeap construction**.

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

Your 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: 4-5-2018 10:00 AM~~ 4-8-2018 10:27 PM

Q3. Recurrence Relationship (28 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).

Q3a. SelectionSort on an array of N comparable values (**`algs.hw3.Select`**)

Let C(N) be the number of times you compare two values in the array. When looking at the code, you should see that each step through the loop advances the problem by one step. Here is the recurrence relationship

C(N) = (N-1) + C(N-1)

[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

Q3b. 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

Q3c. MergeSort on an array of N comparable values (**`algs.hw3.Merge`**)

Let C(N) be the number of times you compare two values in the array

[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: 4-5-2018 10:00 AM~~ 4-8-2018 10:27 PM

Q4. Binary Search Tree (23 pts)

Working with Binary Search Tree structures there is a concept of *height* and *depth*. These are complementary. Your task is to validate proposition C (p. 403) and proposition D (p. 404) from Sedgewick. The final details of this question will be completed Thursday April 5th, including a point breakdown and expected output

> Proposition C: **Search hits** in a BST built from N random keys requires ~ 1.39 log N compares, on average.
>
> The number of compares used for a search hit ending at a given node is 1 plus the depth. Adding the depths of all nodes, we get a quantity known as the _internal path length_ of the tree. Thus the desired quantity (i.e., average # of compares on random BST) is 1 plus the average internal path length of the BST. Let Cn be the internal path length of a BST built from inserting N randomly ordered distinct keys, so the average cost of a search hit is 1 + Cn/N.

You will generate N random keys from the uniform distribution of using StdRandom.uniform(). As you have seen in past homeworks, your job is to evaluate random BSTs of size of (N=64, 128, …, 1024). For a given problem size N, you are to compute an array of N random double values. Use these N keys to construct a BST by inserting these keys into the BST. It doesn't matter what values are to be stored with the keys, so for convenience, just use Boolean as I show in the sample code, and always store the value "true" with each key.

To confirm proposition C, invoke get(key) for each of these values in the BST and you will guarantee a search hit. Compute the necessary statistics and output a table showing the average number of compares for a problem of size N.

To confirm proposition D, construct a new array of N different double values (assuming the unlikelihood of generating a perfect match) and then invoke get(key) for these values, and each one should miss. Compute the necessary statistics and output a table showing the average number of compares for a problem of size N.

N	Cn	H-Ave	M-Ave	Model
64	427	7.67	8.67	7.67
128	xxx	yy	zz	ss
256	xxx	yy	zz	ss
512	xxx	yy	zz	ss
1024	xxx	yy	zz	ss
2048	xxx	yy	zz	ss
4096	xxx	yy	zz	ss

~~Version: 4-5-2018 10:00 AM~~ 4-8-2018 10:27 PM

Q5. Bonus Question (1 pt)

A bonus question will be added shortly.