1 November - Midterm Review

Logistics

- Hw 4 extended to next Friday: implies there will be no tree coding problems on exam
- You will take your mid-term in your lab section you are registered in
- Linux computers in lab, Windows computers in DAC testing center
  - Both have VS Code and OpenJDK
  - Also OK to use your own laptop
- Exam will be published on Canvas
  - Let me know when DAC exam is scheduled so I can set up your exam
  - Can take exam in DAC testing center any day next week

Allowed materials

- Official docs allowed: https://docs.oracle.com/en/java/javase/17/docs/api/help-doc.html
- Recommend navigating the docs before exam
- One, single-sided sheet of notes (handwritten or typed) to be turned in when you leave the exam
- Blank scratch paper
- ANY other accessed web materials or local materials (phones, etc) will be grounds for a zero and referral to Academic Integrity.
- Coding problem will have starter code and tests for download, then you will re-upload code to Canvas. Programming, running and testing code will all happen locally. Can use any editor, but no AI / coPilot / etc.
- Coding problem scope: implementing one or two methods for insertion/deletion in data structure we have covered.
- Not testing on Git / Linux skills

Exam Topics

Topic recap:

- Java understanding, reading and analyzing code
- Runtime Analysis
- Stacks, Queues, Linked Lists ("Linear" data structures)
- Hash Tables, Hashing
- Binary Search
- Sorting
- Binary Search Trees
- More general trees

Open floor for questions

Practice problems

Code interpretation and runtime

Consider the following code. What is the output of `fn1(8)` and `fn2(7)`?

```
fn1(int n) {
    if (n >= 0) {
        System.out.print(n);
        fn1(n-2);
    }
}
```

Output: 8 6 4 2 0

```
fn2(int n) {
    if (n >= 0) {
        fn2(n-2);
        System.out.print(n);
    }
}
```

Output: 1 3 5 7

Runtime

Consider the following psuedocode for computing the nth Fibonacci number F_n. Recall that $F_n = F_{n-1} + F_{n-2}$ and $F_0 = 0$ and $F_1 = 1$.

```
fib(n):
    if n <= 1:
        return n;
    return fib(n-1) + fib(n-2);
```

How many times is the `fib` function called if you call `fib(3)` (including the original call of `fib(3)`)?

- first call: fib(3)-recurses;
- first recurse: second call, fib(2)-recurses; third call, fib(1)-returns;
- second recurse: fourth call, fib(1)-returns; fifth call, fib(0)-returns;
- five total calls

Is the code efficient? Why or why not?

- Looking for answers stating no, this approach involves repeated computation of the same value in different branches of the recursion.
- Actual complexity is $\mathcal{O}(2^N)$ (not expected to know or derive)
- To compute the Nth Fibonacci number we can do an iterative approach:

```
fib = 1;
f(int N) {
    for i in (1..N) {
        fib += fib;
    }
    return fib;
}
```

Sorting

1. What is the worst-case runtime of merge-sort on an array of size n? Explain your reasoning.

Answer: O(n log(n)), see for example a reverse-sorted array: will have to build full binary tree and merge all individual elements.

2. True or false:

- If you are sorting an array where the key is unique for each element, you will get the same result whether your sort algorithm is stable or not.
  - True
- Quick sort has the same average case time complexity as its worst-case time complexity.
  - False: Worst is $\mathcal{O}(n^2)$ (always pick largest/smallest as pivot), average is $\mathcal{O}(nlogn)$ (similar to merge sort).

3. Consider insertion sort, selection sort, merge sort, and quick sort. Which sort(s) never compare the same two elements twice?

- Quicksort: elements are divided according to pivot, then sorted within branches
- Mergesort: once we compare two elements and merge, these sub-lists are sorted
- Insertion sort: when we insert element, we are comparing to sorted elements, compare once
- Selection sort: DOES potentially compare twice, because we *select* the next smallest element from unsorted list and swap with the

Hashing

- What is the time complexity of adding an element to a hash map?
  - O(1)
- Multiple choice: what is a *collision* in a hash table?
  - Two key-value pairs that have equal keys but hash to different indices.
  - Two key-value pairs that have different keys but hash to the same index. (Correct)
  - Two key-value pairs that have equal keys but different values.
  - Two key-value pairs that have different keys and hash to different indices.

Trees

- What is the asymptotic (big-O) height of a balanced binary tree with N elements?
  - O(log N)
- What is the big-O height of a worst-case unbalanced binary tree with N elements?
  - O(N)
- Given tree, write out order of elements in (post | pre | in) order.