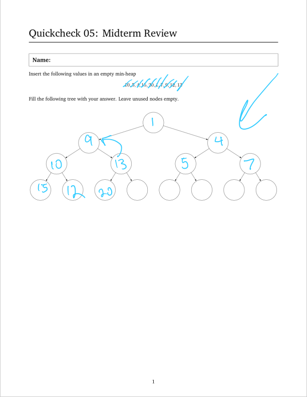
Section 5

Thursday, November 1, 2018 3:02 PM

quickcheck 05



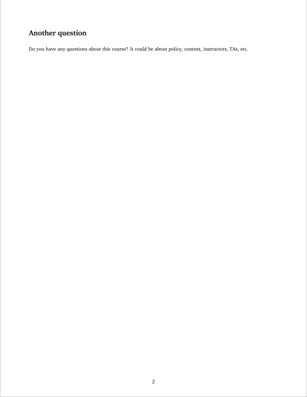
Quickcheck 05: Midterm Review

Name:

Insert the following values in an empty min-heap

10, 5, 4, 15, 20, 1, 7, 9, 12, 13

Fill the following tree with your answer. Leave unused nodes empty.

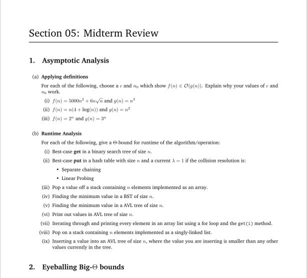


Another question

Do you have any questions about this course? It could be about policy, content, instructors, TAs, etc.

---

section05



Section 05: Midterm Review

1. Asymptotic Analysis

(a) **Applying definitions**

For each of the following, choose a c and n_0 which show $f(n) \in \mathcal{O}(g(n))$. Explain why your values of c and n_0 work.

- (i) $f(n) = 5000n^2 + 6n\sqrt{n}$ and $g(n) = n^3$
- (ii) $f(n) = n(4 + \log(n))$ and $g(n) = n^2$
- (iii) $f(n) = 2^n$ and $g(n) = 3^n$

(b) **Runtime Analysis**

For each of the following, give a Θ-bound for runtime of the algorithm/operation:

- (i) Best-case **get** in a binary search tree of size n.
- (ii) Best-case **put** in a hash table with size n and a current $\lambda = 1$ if the collision resolution is:
  - Separate chaining
  - Linear Probing
- (iii) Pop a value off a stack containing n elements implemented as an array.
- (iv) Finding the minimum value in a BST of size n.
- (v) Finding the minimum value in a AVL tree of size n.
- (vi) Print out values in AVL tree of size n.
- (vii) Iterating through and printing every element in an array list using a for loop and the `get(i)` method.
- (viii) Pop on a stack containing n elements implemented as a singly-linked list.
- (ix) Inserting a value into an AVL tree of size n, where the value you are inserting is smaller than any other values currently in the tree.

2. Eyeballing Big-Θ bounds

For each of the following code blocks, what is the worst-case runtime? Give a big-Θ bound. You do not need to justify your answer.

(a)
```
void f1(int n) {
    int i = 1;
    int j;
    while(i < n*n*n*n) {
        j = n;
        while (j > 1) {
            j -= 1;
        }
        i += n;
    }
}
```

(b)
```
int f2(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            System.out.println(”j = ” + j);
        }
        for (int k = 0; k < i; k++) {
            System.out.println(”k = ” + k);
            for (int m = 0; m < 100000; m++) {
                System.out.println(”m = ” + m);
            }
        }
    }
}
```

(c)
```
int f3(n) {
    count = 0;
    if (n < 1000) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < i; k++) {
                    count++;
                }
            }
        }
    } else {
        for (int i = 0; i < n; i++) {
            count++;
        }
    }
    return count;
}
```

(d)
```
void f4(int n) {
    IList<Integer> arr = new DoubleLinkedList<>();
    for (int i = 0; i < n; i++) {
        if (list.size() > 20) {
            list.remove(0);
        }
        list.add(i);
    }
    for (int i = 0; i < list.size; i++) {
        System.out.println(list.get(i));
    }
}
```

over here!

$T(n) = n^2 + T(\frac{n}{2})$

$n^2 + (\frac{n}{2})^2 + T(\frac{n}{4})$

$n^2 + (\frac{n}{2})^2 + (\frac{n}{4})^2 + T(\frac{n}{8})$

$n^2 + \frac{n^2}{4} + \frac{n^2}{16} + \frac{n^2}{64} + T(\frac{n}{16})$

i=0 i=1 i=2

n $\frac{n}{2}$ $\frac{n}{4}$

want i where input = 4

term i $\boxed{\frac{n}{2^i}}$

$\frac{n}{2^i} = 4$

$\frac{n}{4} = 2^i$

$\log_2(\frac{n}{4}) = i$

$\log_2(n) - \cancel{\log_2}(4) = i$ 2

$\log_2 n - 2 = i$

$$\boxed{5 + \sum_{i=0}^{\log n - 2} \frac{n^2}{4^i}}$$

$$5 + n^2 \sum_{i=0}^{\log n - 2} \frac{1}{4^i}$$

$$5 + n^2 \cdot \sum_{i=0}^{(\log n - 1) - 1} (\frac{1}{4})^i$$

$$\boxed{5 + n^2 \cdot \left(\frac{1 - (\frac{1}{4})^{(\log(n) - 1)}}{1 - (\frac{1}{4})} \right)}$$

3. Recurrences and Summations

Find a closed form of each of the following expressions

(a) $T(n) = \begin{cases} T(n/2) + n^2 & \text{if } n \geq 4 \\ 5 & \text{otherwise} \end{cases}$

(b) $T(n) = \begin{cases} T(n-1) + n^2 & \text{if } n > 19 \\ 57 & \text{otherwise} \end{cases}$

4. Modeling

Consider the following method. Let n be the integer value of the `n` parameter, and let m be the length of `DoubleLinkedList`. You may assume that $n > 7$.

```
public int mystery(int n, DoubleLinkedList<Integer> list) {
    if (n < 7) {
        System.out.println("???");
        int out = 0;
        for (int i = 0; i < n; i++) {
            out += i;
        }
        return out;
    } else {
        System.out.println("???");
        System.out.println("???");
        out = 0;
        for (int i : list) {
            out += 1;
            for (int j = 0; j < list.size(); j++) {
                System.out.println(list.get(j));
            }
        }
        return out + 2 * mystery(n - 4, list) + 3 * mystery(n / 2, list);
    }
}
```

m

0 ...... m-1

m^2

$T(n, m)$

$$\begin{cases} n & \text{if } n<7 \\ m^3 + T(n-4, m) + T(\frac{n}{2}, m) & \text{else} \end{cases}$$

also can say O(1)!

$\frac{j=0}{1} + \frac{j=1}{2} + \frac{j=2}{3} + \dots + \frac{j=m-1}{m}$

m^2

Note: your answer to all three questions should be a recurrence, possibly involving a summation. You do not need to find a closed form.

(a) Construct a mathematical function modeling the *approximate* worst-case runtime of this method in terms of n and m.

(b) Construct a mathematical function modeling the *exact* integer output of this method in terms of n and m.

(c) Construct a mathematical function modeling the *exact* number of lines printed out in terms of n and m.

5. AVL/BST

(a) Insert $\{94, 33, 50, 76, 96, 67, 56, 65, 83, 34\}$ into an initially empty AVL tree.

(b) Insert $\{6, 5, 4, 3, 2, 1, 10, 9, 8, 6, 7\}$ into an initially empty AVL tree.

(c) Suppose you insert 7 elements into a BST. What are the possible heights after the insertions? (Think about different orders of inserting elements).

(d) If you insert 7 elements into an AVL tree, what are the possible heights of the tree?

(e) More generally, what is the minimum number of nodes in an AVL tree of height 4? Draw an instance of such an AVL tree.

(f) Describe an implementation of `delete()` for a BST.

(g) Adapt `delete()` for a BST to work for an AVL tree. Don't worry if there are some edge cases that don't quite work yet, the next part will help with those.

(h) Make a more robust (not buggy) `delete()` for an AVL tree, and try to be as (**hint**) lazy as possible.

6. Hash tables

(a) What is the difference between primary clustering and secondary clustering in hash tables?

(b) Suppose we implement a hash table using double hashing. Is it possible for this hash table to have clustering?

(c) Suppose you know your hash table needs to store keys where each key's hash code is always a multiple of two. In that case, which resizing strategy should you use?

(d) How would you design a hash table that is designed to store a large amount of data – more then you can fit on RAM?

(e) Consider the following key-value pairs.

(6, a), (29, b), (41, d). (34, e), (10, f), (64, g), (50, h)

Suppose each key has a hash function $h(k) = 2k$. So, the key 6 would have a hash code of 12. Insert each key-value pair into the following hash tables and draw what their internal state looks like:

- (i) A hash table that uses separate chaining. The table has an internal capacity of 10. Assume each bucket is a linked list, where new pairs are appended to the end. Do not worry about resizing.
- (ii) A hash table that uses linear probing, with internal capacity 10. Do not worry about resizing.
- (iii) A hash table that uses quadratic probing, with internal capacity 10. Do not worry about resizing.

(f) Consider the three hash tables in the previous question. What are the load factors of each hash table?

7. Debugging

Suppose we are trying to implement an algorithm `isTree(Node node)` that detects whether some binary tree is a valid one or not, and returns `true` if it is, and `false` if it isn't. In particular, we want to confirm that the tree does not contain any *cycles* – there are no nodes where the `left` and `right` fields point to their parents.

Assume that the node passed into the method is supposed to be the root node of the tree.

(a) List at least four different test cases for each problem. For each test case, be sure to specify what the input is (drawing the tree, if necessary), and what the expected output is (assuming the algorithm is implemented correctly).

(b) Here is one (buggy) implementation in Java. List every bug you can find with this algorithm.

```java
public class Node {
    public int data;
    public Node left;
    public Node right;

    // We are simplifying what the equals method is supposed to look like in Java.
    // This method is technically invalid, but you may assume it's
    // implemented correctly.
    public boolean equals(Node n) {
        if (n == null) {
            return false;
        }
        return this.data == n.data;
    }

    public int hashCode() {
        // Pick a random number to ensure good distribution in hash table
        return Random.randInt();
    }
}

boolean isTree(Node node) {
    ISet<Node> set = new ChainedHashSet<>();
    return isTreeHelper(node, set)
}

boolean isTreeHelper(Node node, ISet<Node> set) {
    ISet<Node> set = new ChainedHashSet<>();
    if (set.containsKey(node)) {
        return false;
    } else {
        set.add(node);
        return isTreeHelper(node.left, set) || isTreeHelper(node.right, set);
    }
}
```

---

8. Design Decisions

There isn't a design decisions problem in this handout, but you should be prepared for that kind of question on the midterm.

Design questions in previous sections:

Section 1: Problem 2, 3
Section 3: Problem 7
Section 4: Problem 6, 9, 10

---

section04

Section 04: Dictionaries

Review Problems

1. AVL Trees

Draw an AVL Tree as each of the following keys are added in the order given. Show intermediate steps.

(a)

{“penguin”, “stork”, “cat”, “fowl”, “moth”, “badger”, “otter”, “shrew”, “lion”, “raven”, “bat”}

(b)

$$\{6, 43, 7, 42, 59, 63, 11, 21, 56, 54, 27, 20, 36\}$$

(c)

{“indigo”, “fuscia”, “pink”, “goldenrod”, “violet”, “khaki”, “red”, “orange”, “maroon”, “crimson”, “green”, “mauve”}

2. True or false?

(a) An insertion in an AVL tree with n nodes requires $\Theta(\log(n))$ rotations.

(b) A set of numbers are inserted into an empty BST in sorted order and inserted into an empty AVL tree in random order. Listing all elements in sorted order from the BST is $\mathcal{O}(n)$, while listing them in sorted order from the AVL tree is $\mathcal{O}(\log(n))$.

(c) If items are inserted into an empty BST in sorted order, then the BST's `get()` is just as asymptotically efficient as an AVL tree whose elements were inserted in unsorted order.

(d) An AVL tree will always do a maximum of two rotations in an insert.

3. Big-$\mathcal{O}$

Write down the big-$\mathcal{O}$ for each of the following:

(a) Insert and find in a BST.

(b) Insert and find in an AVL tree.

(c) Finding the minimum value in an AVL tree containing n elements.

(d) Finding the k-th largest item in an AVL tree containing n elements.

(e) Listing elements of an AVL tree in sorted order

Section Problems

4. Hash table insertion

For each problem, insert the given elements into the described hash table. Do not worry about resizing the internal array.

(a) Suppose we have a hash table that uses separate chaining and has an internal capacity of 12. Assume that each bucket is a linked list where new elements are added to the front of the list.

Insert the following elements in the EXACT order given using the hash function $h(x) = 4x$:

0, 4, 7, 1, 2, 3, 6, 11, 16

(b) Suppose we have a hash table that uses linear probing and has an internal capacity of 13.

Insert the following elements in the EXACT order given using the hash function $h(x) = 3x$:

2, 4, 6, 7, 15, 13, 19

(c) Suppose we have a hash table that uses quadratic probing and has an internal capacity of 10.

Insert the following elements in the EXACT order given using the hash function $h(x) = x$:

0, 1, 2, 5, 15, 25, 35

5. Analyzing dictionaries

(a) What are the constraints on the data types you can store in an AVL tree? When is an AVL tree preferred over another dictionary implementation, such as a HashMap?

(b) When is using a BST preferred over an AVL tree?

(c) Consider an AVL tree with n nodes and a height of h. Now, consider a single call to `get(...)`. What's the maximum possible number of nodes `get(...)` ends up visiting? The minimum possible?

(d) **Challenge Problem:** Consider an AVL tree with n nodes and a height of h. Now, consider a single call to `insert(...)`. What's the maximum possible of nodes `insert(...)` ends up visiting? The minimum possible? Don't count the new node you create or the nodes visited during rotation(s).

6. Design

Imagine a database containing information about all trains leaving the Washington Union station on Monday. Each train is assigned a departure time, a destination, and a unique 8-digit train ID number.

What data structures you would use to solve each of the following scenarios. Depending on scenario, you may need to either (a) use multiple data structures or (b) modify the implementation of some data structure.

Justify your choice.

(a) Suppose the schedule contains 200 trains with 52 destinations. You want to easily list out the trains by destination.

(b) In the question above, trains were listed by destination. Now, trains with the same destination should further be sorted by departure time.

t

ID

time

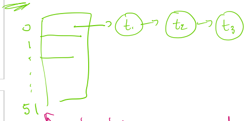
dest → uw

Part A

put("uw", [t1, t2 ...]

uw → 0

DT → 1

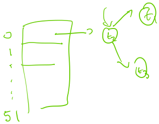


bucket per destination

(c) A train station wants to create a digital kiosk. The kiosk should be able to efficiently and frequently complete look-ups by train ID number so visitors can purchase tickets or track the location of a train. The kiosk should also be able to list out all the train IDs in ascending order, for visitors who do not know their train ID.

Note that the database of trains is not updated often, so the removal and additions of new trains happen infrequently (aside from when first populating your chosen DS with trains).

Part B



Switch chain to AVL Tree

(order trains by departure time.)

7. Evaluating hash functions

Consider the following scenarios.

(a) Suppose we have a hash table with an initial capacity of 12. We resize the hash table by doubling the capacity. Suppose we insert integer keys into this table using the hash function $h(x) = 4x$.

Why is this choice of hash function and initial capacity suboptimal? How can we fix it?

(b) Suppose we have a hash table with an initial capacity of 8 using quadratic probing. We resize the hash table by doubling the capacity.

Suppose we insert the integer keys $2^{20}, 2 \cdot 2^{20}, 3 \cdot 2^{20}, 4 \cdot 2^{20}, \ldots$ using the hash function $h(x) = x$.

Describe what the runtime of the dictionary operations will over time as you keep adding these keys to the table.

8. Code analysis

For this problem, we will consider a hypothetical hash table that uses linear probing and implements the `IDictionary` interface. Specifically, we will focus on analyzing and testing one potential implementation of the `remove` method.

(a) Come up with at least 4 different test cases to test this `remove(...)` method. For each test case, describe what the expected outcome is (assuming the method is implemented correctly).

Try and construct test cases that check if the `remove(...)` method is correctly using the key's hash code. (You may assume that you can construct custom key objects that let you customize the behavior of the `equals(...)` and `hashCode()` method.)

(b) Now, consider the following (buggy) implementation of the `remove(...)` method. List all the bugs you can find.

```java
public class LinearProbingDictionary<K, V> implements IDictionary<K, V> {
    // Field invariants:
    //
    // 1. Empty, unused slots are null
    // 2. Slots that are actually being used contain an instance of a Pair object

    private Pair<K, V>[] array;

    // ...snip...

    public V remove(K key) {
        int index = key.hashCode();

        while ((this.array[index] != null) && !this.array[index].key.equals(key)) {
            index = (index + 1) % this.array.length;
        }

        if (this.array[index] == null) {
            throw new NoSuchKeyException();
        }
        V returnValue = this.array[index].value;
        this.array[index] = null;
        return returnValue;
    }
}
```

(c) Briefly describe how you would fix these bug(s).

9. Algorithm design: easier

When writing mathematical expression, we typically write expressions in *infix* notation: in the form `NUM OPERATOR NUM`. An example of an expression written in infix notation is $4 + 6 * 5$. This expression evaluates to 34.

An alternative way we can write this expression is using *post-fix* notation: in the form `NUM NUM OPERATOR`. For example, consider the following expression written in post-fix notation:

$$4, 6, 5, *, +$$

This expression is interpreted in the following way:

- Read and store 4
- Read and store 6
- Read and store 5
- Multiply the last two stored values (and remove them from storage), then store the result
- Add the last two stored values (and remove them from storage), then store the result

The last result stored is the final "output". In this case, the expression above also evaluates to 34.

(a) Explain how you might apply or adapt the ADTs and data structures you've studied so far to evaluate an expression written in post-fix notation. Assume you accept the expression you need to evaluate as a string.

(b) Give pseudocode for this algorithm.

10. Algorithm design: harder

(a) Given a BST, describe how you could convert it into an AVL tree. What is the runtime of your algorithm?

(b) Give pseudocode for an algorithm that verifies that a tree satisfies all of the AVL invariants in $\mathcal{O}(n)$ time.

Assume every node object has five fields: `key`, `value`, `height`, `left`, and `right`.

Be sure to verify that:

- The tree is actually binary search tree
- The height information of every node is correct
- Every node is balanced

Hint: rather then trying to check all three of these things in a single pass, try writing three separate methods: one per each invariant. While it's possible to check everything in one pass, doing so will be more challenging to implement.

(As a reminder: If each method takes $\mathcal{O}(n)$ time, running all three of them will still take $\mathcal{O}(n)$ time.)

Challenge Problems

11. Random Hash Functions

In class we talked about various strategies to minimize collisions. In this question we discuss how to use randomness to "spread out" collisions from a small set of very bad inputs into a larger set of almost-always-fine inputs. The last two parts of this problem are beyond the scope of this course, but are interesting nonetheless.

For simplicity, assume our keyspace (the set of possible keys) is the set $\{0, 1, 2, \ldots, 2^{30} - 1\}$. Suppose we have a hashtable of size 2^{10}. Let a be an odd integer less than 2^{30}.

Consider the hash function

$$h_a(x) = \left\lfloor \frac{(ax) \mod 2^{30}}{2^{20}} \right\rfloor$$

Notice that the function changes depending on the value of a we choose, so this is really a set of possible functions.

(a) Show that for any a, h_a outputs an integer between 0 and $2^{10} - 1$ (i.e. we can use this as a hash function for our table size)

(b) Choose $a = 1$, i.e. the hash function simplifies to

$$h_1(x) = \left\lfloor \frac{x \mod 2^{30}}{2^{20}} \right\rfloor$$

For this function, find a large set of elements that all hash to 0.

(c) Let x, y be any of the two elements you found in the last part. Choose a few thousand values of a, and check whether $h_a(x) = h_a(y)$ for each of them (write code for this part). For what fraction of these hash functions do x, y collide? If the values of the hash function were totally random, how often would you expect collisions?

(d) The following statement is true (explaining why is beyond the scope of the course): For any x, y if you choose a at random, the probability that $h_a(x) = h_a(y)$ is at most $2/2^{10}$.

Use this fact, or your observations in the last part, to explain why you might decide to choose a random a instead of just choosing $a = 1$ (hint: imagine you know someone is using the hash function with $a = 1$, how can you use the first part to slow their code down? Can you do the same for a random a?)