

Section 03: Recurrences, Master Theorem, Tree Method

Main Problems

1. Finding Bounds

For each of the following code blocks, what is the worst-case runtime? Give a big- Θ bound.

(a)

```
int result = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        result++;
    }
}
```

(b)

```
public IList<String> repeat(DoubleLinkedList<String> list, int n) {
    IList<String> result = new DoubleLinkedList<String>();
    for(String str : list) {
        for(int i = 0; i < n; i++) {
            result.add(str);
        }
    }
    return result;
}
```

(c)

```
public void foo(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 5; j < i; j++) {
            System.out.println("Hello!");
        }

        for (int j = i; j >= 0; j -= 2) {
            System.out.println("Hello!");
        }
    }
}
```

(d)

```
public int num(int n){
    if (n < 10) {
        return n;
    } else if (n < 1000) {
        return num(n - 2);
    } else {
        return num(n / 2);
    }
}
```

(e)

```
public int foo(int n) {
    if (n <= 0) {
        return 3;
    }
    int x = foo(n - 1);
    System.out.println("hello");
    x += foo(n - 1);
    return x;
}
```

2. Binary Search Trees

(a) Write a method `validate` to validate a BST. Although the basic algorithm can be converted to any data structure and work in any format, if it helps, you may write this method for the `IntTree` class:

```
public class IntTree {
    private IntTreeNode overallRoot;

    // constructors and other methods omitted for clarity

    private class IntTreeNode {
        public int data;
        public IntTreeNode left;
        public IntTreeNode right;

        // constructors omitted for clarity
    }
}
```

3. Code To Recurrence

(a) Consider the following method.

```
public static int f(int N) {
    if (N <= 1) {
        return 0;
    }

    int result = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < i; j++) {
            result++;
        }
    }

    return 5 * f(N / 2) + 3 * result + 2 * f(N / 2) + f(N / 2) + f(N / 2);
}
```

Give a recurrence formula for the running time of this code. It's OK to provide a \mathcal{O} for the non-recursive terms (for example if the running time is $A(n) = 4A(n/3) + 25n$, you need to get the 4 and the 3 right but you don't have to worry about getting the 25 right). Just show us how you got there.

Hint: Notice that the main loop is the exact same code as Problem 1A.

(b) Consider the following method.

```
public static int g(n) {
    if (n <= 1) {
        return 1000;
    }
    if (g(n / 3) > 5) {
        for (int i = 0; i < n; i++) {
            System.out.println("Hello");
        }
        return 5 * g(n / 3);
    } else {
        for (int i = 0; i < n * n; i++) {
            System.out.println("World");
        }
        return 4 * g(n / 3);
    }
}
```

(i) Find a recurrence $S(n)$ modeling the worst-case runtime of $g(n)$.

(ii) Find a recurrence $X(n)$ modeling the *returned integer output* of $g(n)$.

(iii) Find a recurrence $P(n)$ modeling the *printed output* of $g(n)$.

(c) Consider the following set of recursive methods.

```
public int test(int n) {
    IDictionary<Integer, Integer> dict = new AvlDictionary<>();
    populate(n, dict);
    int counter = 0;
    for (int i = 0; i < n; i++) {
        counter += dict.get(i);
    }
    return counter;
}

private void populate(int k, IDictionary<Integer, Integer> dict) {
    if (k == 0) {
        dict.put(0, k);
    } else {
        for (int i = 0; i < k; i++) {
            dict.put(i, i);
        }
        populate(k / 2, dict);
    }
}
```

(i) Write a mathematical function representing the *worst-case runtime* of `test`.

You should write two functions, one for the runtime of `test` and one for the runtime of `populate`.

4. Master Theorem

For each of the recurrences below, use the Master Theorem to find the big- Θ of the closed form or explain why Master Theorem doesn't apply. (See the last page for the definition of Master Theorem.)

- (a) $T(n) = \begin{cases} 18 & \text{if } n \leq 5 \\ 3T(n/4) + n^2 & \text{otherwise} \end{cases}$
- (b) $T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 9T(n/3) + n^2 & \text{otherwise} \end{cases}$
- (c) $T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ \log(n)T(n/2) + n & \text{otherwise} \end{cases}$
- (d) $T(n) = \begin{cases} 1 & \text{if } n \leq 19 \\ 4T(n/3) + n & \text{otherwise} \end{cases}$
- (e) $T(n) = \begin{cases} 5 & \text{if } n \leq 24 \\ 2T(n-2) + 5n^3 & \text{otherwise} \end{cases}$

Hashing Problems

1. Hash Table Insertion!

- (a) Suppose we have a hash table implemented using separate chaining. This hash table has an internal capacity of 10. Its buckets are implemented using a linked list where new elements are appended to the end. Do not worry about resizing.

Show what this hash table internally looks like after inserting the following key-value pairs in the order given using the hash function $h(x) = 4x$:

$$(1, a), (4, b), (2, c), (17, d), (12, e), (9, e), (19, f), (4, g), (8, c), (12, f)$$

- (b) Consider the following scenario:

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?

2. More 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

(d) 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 $2^{20}, 2 \cdot 2^{20}, 3 \cdot 2^{20}, 4 \cdot 2^{20}, \dots$ 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.

3. Even More Hash Table Insertion!

(a) 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.

4. Hashing and Mutation

For the following problems, assume that:

1. IntList is a list of integers.
2. The hash code of an IntList is the sum of the integers in the list.
3. IntLists are considered equal only if they have the same size and the same values in the same order.
4. FourBucketHashMap uses separate chaining and the new items are added to the back of each bucket.
5. FourBucketHashMap always has four buckets and never resizes.

Consider the following code:

```
FourBucketHashMap<IntList, String> fbhm = new FourBucketHashMap<>();
IntList list1 = IntList.of(1, 2);
fbhm.put(list1, "dog");
// Part i
list1.add(3);
// Part ii
```

(a) At Part i (line 4), what will be returned from the following statement?

```
fbhm.get(IntList.of(1, 2));
```

(b) At Part II (line 6), what will be returned from the following statements?

```
fbhm.get(IntList.of(1, 2));
```

```
fbhm.get(IntList.of(1, 2, 3));
```

(c) Is there a problem with the code? If so, explain.

5. Debugging a Hash Table

Suppose we are in the process of implementing a hash map that uses open addressing and quadratic probing and want to implement the delete method.

(a) Consider the following implementation of delete. List every bug you can find.

Note: You can assume that the given code compiles. Focus on finding run-time bugs, not compile-time bugs.

```
public class QuadraticProbingHashTable<K, V> {
    private Pair<K, V>[] array;
    private int size;

    private static class Pair<K, V> {
        public K key;
        public V value;
    }

    // Other methods are omitted, but functional.

    /**
     * Deletes the key-value pair associated with the key, and
     * returns the old value.
     *
     * @throws NoSuchElementException if the key-value pair does not exist in the method.
     */
    public V delete(K key) {
        int index = key.hashCode() % this.array.length;

        int i = 0;
        while (this.array[index] != null && !this.array[index].key.equals(key)) {
            i += 1;
            index = (index + i * i) % this.array.length;
        }

        if (this.array[index] == null) {
            throw new NoSuchElementException("Key-value pair not in dictionary");
        }

        this.array[index] = null;

        return this.array[index].value;
    }
}
```

(b) Let's suppose the Pair array has the following elements (pretend the array fit on one line):

["lily", V ₂]	["castle", V ₆]	["resource", V ₁]	["hard", V ₉]	["bathtub", V ₀]
["wage", V ₄]	["refund", V ₇]	["satisfied", V ₆]	["spring", V ₈]	["spill", V ₃]

And, that the following keys have the following hash codes:

Key	Hash Code
"bathtub"	9744
"resource"	4452
"lily"	7410
"spill"	2269
"wage"	8714
"castle"	2900
"satisfied"	9251
"refund"	8105
"spring"	6494
"hard"	9821

What happens when we call `delete` with the following inputs? Be sure write out the resultant array, and to do these method calls *in order*. (**Note:** If a call results in an infinite loop or an error, explain what happened, but don't change the array contents for the next question.)

- (i) `delete("lily")`
 - (ii) `delete("spring")`
 - (iii) `delete("castle")`
 - (iv) `delete("bananas")`
 - (v) `delete(null)`
- (c) List four different test cases you would write to test this method. For each test case, be sure to either describe or draw out what the table's internal fields look like, as well as the expected outcome (assuming the `delete` method was implemented correctly). **Hint:** You may use the inputs previously given to help you identify tests, but it's up to you to describe what kind of input they are testing generally.

Math Review

6. Best case and worst case runtimes

For the following code snippet give the big- Θ bound on the worst case runtime as well the big- Θ bound on the best case runtime, in terms of n the size of the input array.

```

void print(int[] input) {
    int i = 0;
    while (i < input.length - 1) {
        if (input[i] > input[i + 1]) {
            for (int j = 0; j < input.length; j++) {
                System.out.println("uh I don't think this is sorted plz help");
            }
        } else {
            System.out.println("input[i] <= input[i + 1] is true");
        }
        i++;
    }
}

```

8. Big-O, Big-Omega True/False Statements

For each of the statements determine if the statement is true or false. You do not need to justify your answer.

- (a) $n^3 + 30n^2 + 300n$ is $\mathcal{O}(n^3)$
- (b) $n \log(n)$ is $\mathcal{O}(\log(n))$
- (c) $n^3 - 3n + 3n^2$ is $\mathcal{O}(n^2)$
- (d) 1 is $\Omega(n)$
- (e) $.5n^3$ is $\Omega(n^3)$

9. 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) {
    // NOTE: This is your data structure from the first project.
    LinkedDeque<Integer> deque = new LinkedDeque<>();
    for (int i = 0; i < n; i++) {
        if (deque.size() > 20) {
            deque.removeFirst();
        }
        deque.addLast(i);
    }
    for (int i = 0; i < deque.size(); i++) {
        System.out.println(deque.get(i));
    }
}
```

10. Modeling

Consider the following method. Let n be the integer value of the n parameter, and let m be the size of the `LinkedDeque`.

```
public int mystery(int n, LinkedDeque<Integer> deque) {
    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;
        // NOTE: Assume LinkedDeque has working, efficient iterator.
        for (int i : deque) {
            out += 1;
            for (int j = 0; j < deque.size(); j++) {
                System.out.println(deque.get(j));
            }
        }
    }
}
```

```

    }
    return out + 2 * mystery(n - 4, deque) + 3 * mystery(n / 2, deque);
}

```

Give a recurrence formula for the **worst-case** running time of this code. It's OK to provide a \mathcal{O} for non-recursive terms (for example if the running time is $A(n) = 4A(n/3) + 25n$, you need to get the 4 and the 3 right but you don't have to worry about getting the 25 right). Just show us how you got there.

Master Theorem

For recurrences in this form, where a, b, c, e are constants:

$$T(n) = \begin{cases} d & \text{if } n \leq \text{some constant} \\ aT(n/b) + e \cdot n^c & \text{otherwise} \end{cases} \quad T(n) \text{ is } \begin{cases} \Theta(n^c) & \text{if } \log_b(a) < c \\ \Theta(n^c \log n) & \text{if } \log_b(a) = c \\ \Theta(n^{\log_b(a)}) & \text{if } \log_b(a) > c \end{cases}$$

Useful summation identities

Splitting a sum

$$\sum_{i=a}^b (x + y) = \sum_{i=a}^b x + \sum_{i=a}^b y$$

Adjusting summation bounds

$$\sum_{i=a}^b f(x) = \sum_{i=0}^b f(x) - \sum_{i=0}^{a-1} f(x)$$

Factoring out a constant

$$\sum_{i=a}^b cf(i) = c \sum_{i=a}^b f(i)$$

Summation of a constant

$$\sum_{i=0}^{n-1} c = \underbrace{c + c + \dots + c}_{n \text{ times}} = cn$$

Note: this rule is a special case of the rule on the left

Gauss's identity

$$\sum_{i=0}^{n-1} i = 0 + 1 + 2 + \dots + n - 1 = \frac{n(n-1)}{2}$$

Finite geometric series

$$\sum_{i=0}^{n-1} x^i = 1 + x + x^2 + \dots + x^{n-1} = \frac{x^n - 1}{x - 1}$$

Infinite geometric series

$$\sum_{i=0}^{\infty} x^i = 1 + x + x^2 + \dots = \frac{1}{1-x}$$

Note: applicable only when $-1 < x < 1$

Useful Log Rules

Power of a log identity

$$a^{\log_b c} = c^{\log_b a}$$

Product rule

$$\log_c(a * b) = \log_c a + \log_c b$$

Quotient rule

$$\log_c(a/b) = \log_c a - \log_c b$$

Power rule

$$\log_c(a^b) = b * \log_c a$$

Change of base formula

$$\log_b a = (\log_c a) / (\log_c b)$$