CS 61B Spring 2018

Midterm 2 Guerrilla Section

March 17, 2018

Instructions

Form a small group. Start on the first problem. Check off with a helper or discuss your *solution process* with another group once everyone understands *how to solve* the first problem and then repeat for the second problem ...

You may not move to the next problem until you check off or discuss with another group and *everyone understands why the solution is what it is.* You may use any course resources at your disposal: the purpose of this review session is to have everyone learning together as a group.

1 Asymptotic Approach

1.1 Give a tight asymptotic runtime bound on `foo(n)`.

```
int foo(int n) {
    if (n == 0) {
        return 0;
    }
    baz(n);
    return foo(n / 3) + foo(n / 3) + foo(n / 3);
}

int baz(int n) {
    for (int i = 0; i < n; i += 1) {
        System.out.println("Help me! I'm trapped in a loop!");
    }
    return n;
}
```

1.2 For each of the following methods, give a tight asymptotic runtime bound with respect to N.

(a)
```
public void mystery1(int N) {
    for (int i = 0; i < N; i += 1) {
        for (int j = 0; j < N; j += 1) {
            i = i * 2;
            j = j * 2;
        }
    }
}
```

(b)
```
public void mystery2(int N) {
    for (int i = N; i > 0; i = i / 2) {
        for (int j = 0; j < i * 2; j += 1) {
            System.out.println("Hello World");
        }
    }
}
```

(c)
```
public void mystery3(int N) {
    for (int i = N; i > 0; i = i / 2) {
        for (int j = 0; j < i * i; j += 1) {
            System.out.println("Hello World");
        }
    }
}
```

(d)
```
public void mystery4(int N) {
   int i = 1, s = 1;
   while (s <= N) {
        i += 1
        s = s + i;
        System.out.println(s);
    }
}
```

2 Iterator or Iterable?

2.1 Implement the `Filter` class such that its `main` method correctly prints out the even numbers in the given collection: `0 20 14 50 66`. Assume we have already imported `java.util.*` in each file.

```
1  public interface FilterCondition<T> {
2      /** Returns true if the item meets a certain condition (is even, etc.). */
3      boolean eval(T item);
4  }
```

```
1  public class EvenCondition implements FilterCondition<Integer> {
2      public boolean eval(Integer i) {
3          return i % 2 == 0;
4      }
5  }
```

```
public class Filter implements Iterable<Integer> {



    public Filter(Iterable<Integer> it, FilterCondition<Integer> cond) {


    }

    public Iterator<Integer> iterator() {

    }

    private class FilterIterator implements Iterator<Integer> {
```

```
    }

    public static void main(String[] args) {
        List<Integer> arr = Arrays.asList(new Integer[]{0, 11, 20, 13, 14, 50, 66});
        for (int i : new Filter(arr, new EvenCondition())) {
            System.out.print(i);
        }
    }
}
```

3 Stacks of Fun

3.1 An SQueue is a queue implemented using two stacks.

```
1  public class SQueue {
2      private Stack in, out;
3
4      public SQueue() {
5          in = new Stack();
6          out = new Stack();
7      }
8
9      public void enqueue(int item) {
10         in.push(item);
11     }
12
13     public void dequeue() {
14         if (out.isEmpty()) {
15             while (!in.isEmpty()) {
16               out.push(in.pop());
17             }
18         }
19         return out.pop();
20     }
21 }
```

Now, suppose we construct an SQueue and enqueue 100 items.

(a) How many calls to push and pop result from the next call to dequeue?

(b) How many calls to push and pop result from each of the next 99 calls to dequeue?

(c) How many calls to push and pop (total) were required to dequeue 100 elements? How many operations is this per element dequeued?

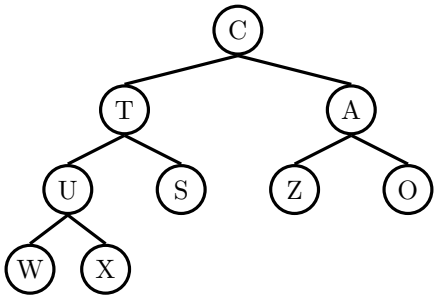
(d) What is the worst-case time to dequeue an item from an SQueue containing N elements? What is the runtime in the best case? Answer using $\Theta(\cdot)$ notation. You may assume that both push and pop run in $\Theta(1)$.

(e) What is the amortized time to dequeue an item from an SQueue containing N elements? Again, answer using $\Theta(\cdot)$ notation.

4 Greetings, Tree Traveler

4.1 Draw a full binary tree that has the following pre-order and post-order traversals. Each node should contain exactly one letter. A full binary tree is a tree such that all nodes except leaf nodes have exactly 2 children.

- Pre-order: C T U W X S A Z O
- Post-order: W X U S T Z O A C



(a) What is the in-order traversal of this tree?

(b) Can a tree have the same in-order and post-order traversals? If so, what can you say about the tree?

(c) What about a tree with the same pre-order and post-order traversals?

4.2 Insert the following list into a 2-3 tree: `20, 10, 35, 40, 50, 5, 25, 15, 30, 60`.

4.3 Draw the corresponding left-leaning red-black tree.

4.4 For each scenario below, indicate whether the node's edge to its parent is **red**, **black**, or **either** red or black. If it could be either color, explain.

(a) The largest value in a tree with more than one node.

(b) The smallest value in a tree with more than one node.

(c) A node with a red *grandparent* edge.

(d) A node whose children are the same color.

(e) The last node inserted, after any rotations and color flips.

5 Quick Union

The quick union data structure is handles set union and membership operations.

- `connect(a, b)`: connects the set of `a` to the set of `b`.
- `isConnected(a, b)`: returns true if `a` and `b` are in the same set.

Internally, quick union sets are trees. Sets can be connected by adding one set's tree to the root of another set's tree. Weighted quick union data structures improve upon quick union data structures by always adding the shorter tree to the root of the taller tree during `connect` operations.

5.1 Draw the Weighted Quick Union object that results after the following method calls.

```
connect(1, 3)
connect(0, 4)
connect(0, 1)
connect(0, 2)
```

5.2 (a) What is the worst way to connect the list `1, 2, 3, 4, 5`? Give an answer in the form of a series of calls to the `connect` method.

(b) Assuming a single node has a height of 0, what is the shortest and tallest height possible for a quick union object with N elements?

(c) Give the best and worst case runtimes for `isConnected` and `connect`.

(d) What is the shortest and tallest height possible for a weighted quick union with N elements? What does this mean for the best and worst-case runtimes for `isConnected` and `connect`?

6 Hashing

6.1 Illustrate the box-and-pointer diagram that results from inserting **galumphing, frumious, slithy, borogroves, mome, bandersnatch** into a new hash table, if `String::hashCode` just returns the length (this is *not* how strings actually work).

Use external chaining to resolve collisions. Assume 4 is the initial size of the hash table's internal array, and double the array's size when the load factor equals 1.

6.2 Describe a potential problem with each of the following `hashCode`s.

(a) `String::hashCode` simply returns the length of the string.

(b) `String::hashCode` simply returns a random number each time.

(c) Overriding the `equals` method without overriding the `hashCode` method.

(d) Overriding the `hashCode` method of a class without overriding the `equals` method.

(e) Mutating an object after inserting it into a HashSet.

6.3 (a) Give the worst-case runtime bound for inserting a single entry into a `HashSet` containing N elements. Assume that hashing a key takes constant time. Then, describe a situation in which we would achieve the above runtime.

(b) After finishing the hashing lab, Janice decides to create her own hash table implementation, `SmallMap`. In order to avoid costly resizing operations, `SmallMap` 's internal array does not resize: it has a fixed length of 1,024. What is the best-case and worst-case runtime for insertion into a `SmallMap` containing N elements? Assume that the N elements are distributed uniformly.

7 Heaps of Fun

7.1 Describe a way to modify the usual max heap implementation so that finding the minimum element takes constant time without incurring more than a constant amount of additional time and space for the other operations.

7.2 In class, we looked at one way of implementing a priority queue: the binary heap. Recall that a binary heap is a nearly complete binary tree such that any node is smaller than all of its children. There is a natural generalization of this idea called a d-ary heap. This is also a nearly complete tree where every node is smaller than all of its children. But instead of every node having two children, every node has d children for some fixed constant d.

(a) Describe how to insert a new element into a d-ary heap. (This should be very similar to the binary heap.) What is the running time in terms of d and N, the number of elements?

(b) What is the running time of finding the minimum element in a min-d-ary heap with N nodes in terms of d and N?

(c) Describe how to remove the minimum element from a min-d-ary heap. What is the running time in terms of d and N?

7.3 Suppose a value in a max heap changed. To maintain heap invariants, would you bubble the value up, bubble it down, both, or neither? Explain.

7.4 Suppose you are given an array `A` with N elements such that no element is more than `k` slots away from its position in the sorted array. Assume that $k > 0$ and that k, while not constant, is much less than N ($k \ll N$).

(a) Implement `zorkSort` such that the array `A` is sorted after execution. The important operations on a `PriorityQueue` are `add(x)`, `remove()` (remove smallest), and `isEmpty()`.

```
1  public static void zorkSort(int[] A, int k) {
2          int N = A.length, i = 0;
3          PriorityQueue<Integer> pq = new PriorityQueue<>();
4          while (i < k) {
5                  ____________________________________________
6                  i += 1;
7          }
8          while (____________________) {
9                  A[i - k] = ____________________________________
10                 ____________________________________________
11                 i += 1;
12         }
13         while (____________________) {
14                 ____________________________________________
15                 i += 1;
16         }
17 }
```

(b) What is the running time of this algorithm, as a function of N and k?

7.5 President Trump's speaking style has attracted a great deal of attention in recent years. We'd like to determine the top k most common words found in the list of all N words that the president has ever spoken. Describe how you would solve this problem efficiently and which data structure(s) you would use. The algorithm should be able to return the president's k most frequently-used words in $O(N \log k)$. Assume $k \ll N$.