QUEENS COLLEGE Department of Computer Science
CSCI 313 Second Midterm Exam Fall 2019 11.26.19

Instructor: Alex Ryba

07.45am – 09.00am, Tuesday, November 26, 2019

Problem 1 For each of the following functions give a fully *simplified* Θ estimate for the run time in terms of `N`

(a)

```
public static int f1(int N) {
   int x = 0;
   for (int i = 0; i < N * N; i+=2)
      x++;
   return x;
}
```

Answer: $\Theta(N^2)$

(b)

```
public static int f2(int N) {
   int x = 0;
   for (int i = 0; i < Math.sqrt(N); i++)
      for (int j = 0; j < i; j++)
          x++;
   return x;
}
```

Answer: $\Theta(N)$

(c)

```
public static int f3(int N) {
   if (N <= 1) return 1;
   int x = 0;
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < N; j++)
         x += j * j;
      x -= f3(N / 2);
   }
   return x;
}
```

Answer: $\Theta(N \log N)$

(d)

```
public static int f4(int N) {
   if (N == 0) return 0;
   return f4(N/2) + f1(N) + f4(N/2);
}
```

Answer: $\Theta(N^2)$

(e)

```
public static int f5(int N) {
   int x = 0;
   for (int i = N; i > 0; i = i/2)
      for (int j = 0; j < i; j++)
        x += i * j;
   return x;
}
```

Answer: $\Theta(N)$

Problem 2 This question asks you to fill in the missing parts from the code for an implementation of a hash table that uses open addressing and linear probing. Nine small pieces of the code have been omitted and replaced by PART (a), PART (b) and so on in the following implementation. You should answer each part by supplying the omitted code. (In each case it is at most one line of code.)

```
public class OpenHashTable<E> implements Set<E> {
  int size;   int capacity;   E seats[];   E ghost;

  OpenHashTable(int cap, E ghostEntry) {
    this.ghost = ghostEntry;   capacity = cap;   size = 0;
    seats = (E[]) new Object[cap];
    for (int i = 0; i < cap; i++) seats[i] = null;
  }

  public boolean contains(E k) // standard code not shown
  public E get(E k)            // standard code not shown
  public void remove(E k)      // standard code not shown

  public void add(E k) throws Exception {
    int i = locate(k);
    if (i != -1) seats[i] = k;
    else {
      i = findSpace(k);
      if (i != -1) {  seats[i] = k;  size++; }
      else throw new Exception("Hash Table is Full");
    }
  }


  private int locate(E k) {
    int h = PART (a);
    int step = 0;
    while (step < capacity) {
      int index = resolve(h, step);
      if (seats[index] == null) return PART (b);
      if (k != ghost && k.equals(seats[index]))
        return PART (c);
      step++;
    }
    return PART (d);
  }

  private int findSpace(E k) {
    int h = PART (e);
    int step = 0;
    while (step < capacity) {
      int index = resolve(h, step);
      if (seats[index] == null) return PART (f);
      if (seats[index] == ghost) return PART (g);
      step++;
    }
    return PART (h);
  }

  private int resolve(int h, int step) {
    return PART (i);
  }
}
```

(a) Give a replacement for PART (a). **Answer:** `k.hashCode() % capacity`

(b) Give a replacement for PART (b). **Answer:** `-1`

(c) Give a replacement for PART (c). **Answer:** `index`

(d) Give a replacement for PART (d). **Answer:** `-1`

(e) Give a replacement for PART (e). **Answer:** `k.hashCode() % capacity`

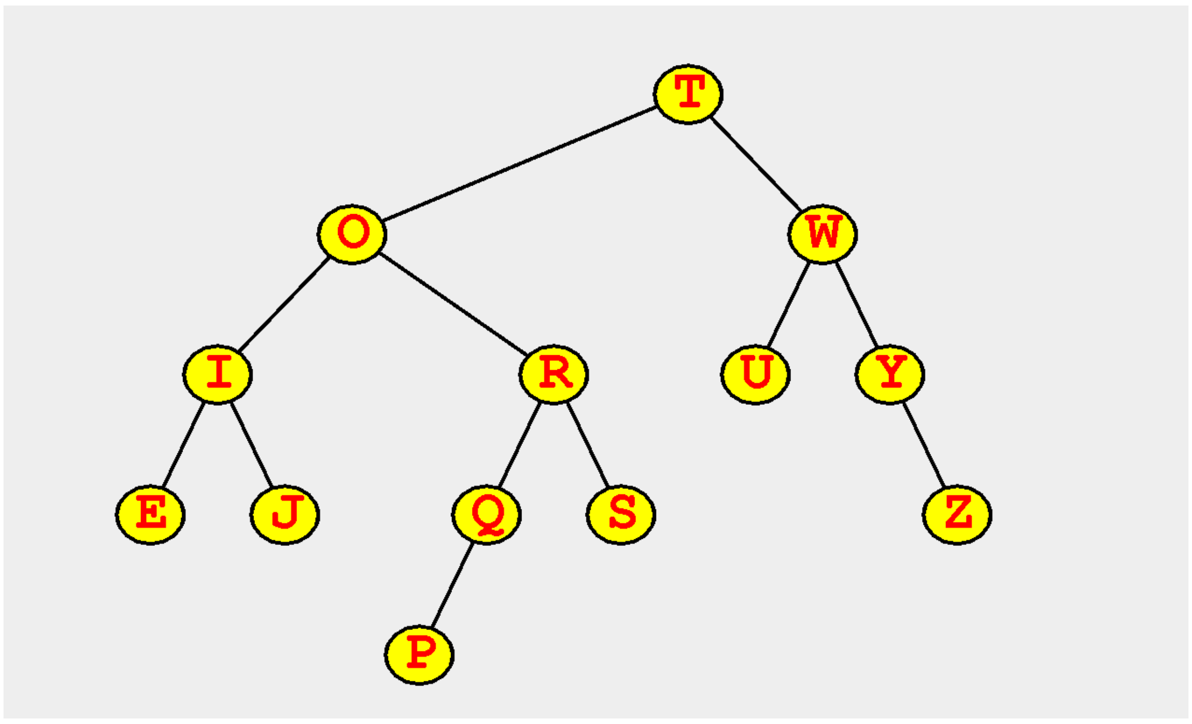
(f) Give a replacement for PART (f). **Answer:** `index`

(g) Give a replacement for PART (g). **Answer:** `index`

(h) Give a replacement for PART (h). **Answer:** `-1`

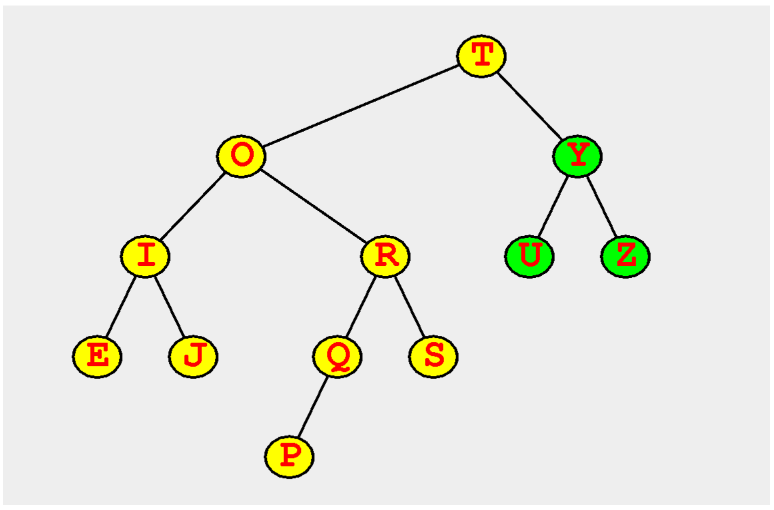
(i) Give a replacement for PART (i). **Answer:** `(h + step) % capacity`

Problem 3 This problem requires you to step through the process of removing a data element from an AVL tree. Suppose that the data element `W` is removed from the following AVL tree.

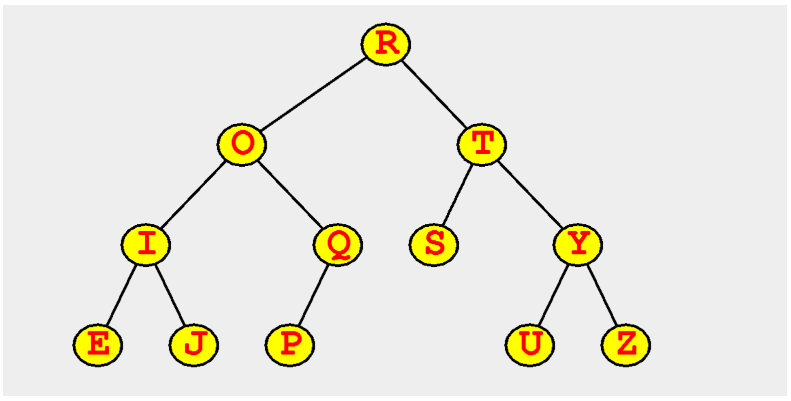


(a) The first step of removal is to promote data from a lower node to replace W. Which two data elements could be promoted? **Answer:** U and Y

(b) After one of these promotions the node that originally contained `W` must be rebalanced. Show the state of the tree after this node is rebalanced but before any other rebalancing is applied. **Answer:**



(c) Is a further rebalance operation needed? If so draw the state of the tree after it is applied. **Answer:**



Problem 4 This question asks you to write the code for three missing methods of the `class MultiMap<K, V>`. In a MultiMap each key (of type K) is associated to some values (of type V). In contrast in a Map each key is associated to just one value. In a multimap a key such as 211 can be associated to both the values `"Waxman"` and `"Alayev"`. The three methods of a MultiMap are

- `get` which returns the Set of associated values for a key.
- `put` which associates an additional value to a key.
- `remove` which removed an associated value from a key.

The following code implements a MultiMap as a Map adapter. The Map stores the Set of all values associated to a key. The code for the three methods is omitted.

HINTS: Since you are writing an adapter, the question is testing whether you can apply the methods of the classes Map and Set to perform the required tasks. The code you need to write is very short (no function needs more than 4 instructions) and all of the instructions will call Map or Set methods. In case you ever need to construct a Map or a Set object you would use a HashMap or HashSet object as shown in the given constructor.

```
import java.util.Map;
import java.util.Set;
import java.util.HashMap;
import java.util.HashSet;

public class MultiMap<K, V> {

  Map<K, Set<V>> data;

  public MultiMap() {
    data = new HashMap<>();
  }

  public Set<V> get(K k) {
    // code omitted
  }

  public void put(K k, V v) {
    // code omitted
  }

  public void remove(K k, V v) throws Exception {
     // code omitted
  }

  public static void main(String args[]) throws Exception {
    MultiMap<Integer, String> classes = new MultiMap<>();
    classes.put(211, "Waxman");   classes.put(211, "Alayev");
    classes.put(320,  "Boklan"); classes.put(320, "Obrenic");
    classes.put(320,  "Teitelman");
    for (String x:classes.get(320))
      System.out.print(x + " ");  // output: Boklan Teitelman Obrenic
    System.out.println();
    classes.remove(320,  "Teitelman");
    for (String x:classes.get(320))
      System.out.print(x + " ");  // output: Boklan Obrenic
    System.out.println();
  }
}
```

(a) Give an implementation for the method *get*. **Answer:**

```
  public Set<V> get(K k) {
    return data.get(k);
  }
```

(b) Give an implementation for the method *put*. **Answer:**

```
  public void put(K k, V v) {
    if (!data.containsKey(k))
      data.put(k,  new HashSet<V>());
    Set<V> s = data.get(k);
    s.add(v);
  }
```

(c) Give an implementation for the method *remove*. **Answer:**

```
  public void remove(K k, V v) throws Exception {
    Set<V> s = data.get(k);
    if (s == null) throw new Exception("Not present in map");
    if (!s.contains(v)) throw new Exception("Not present in map");
    s.remove(v);
  }
```