

## CSE 143 Sample Midterm Exam #3 Key

1.

### List

(a) [10, 20, 30]

(b) [8, 2, 9, 7, 4]

(c) [-1, 3, 28, 17, 9, 33]

### Output

[20, 10, 20, 30, 30, 20]

[8, 7, 8, 2, 9, 7, 4, 4, 2, 8]

[33, 28, 33, -1, 3, 28, 17, 9, 33, 17, -1, 33]

2. Two solutions are shown.

```
public static void removeBadPairs(ArrayList<Integer> list) {
    if (list.size() % 2 != 0) {
        list.remove(list.size() - 1);
    }

    for (int i = 0; i < list.size(); i += 2) {
        if (list.get(i) > list.get(i + 1)) {
            list.remove(i);
            list.remove(i);
            i -= 2;
        }
    }
}
```

```
public static void removeBadPairs(ArrayList<Integer> list) {
    if (list.size() % 2 != 0) {
        list.remove(list.size() - 1);
    }

    for (int i = list.size() - 1; i > 0; i--) {
        if (i % 2 != 0 && list.get(i - 1) > list.get(i)) {
            list.remove(i);
            list.remove(i - 1);
        }
    }
}
```

3. Two solutions are shown.

```
public static void mirrorHalves(Queue<Integer> q) {
    if (q == null || q.size() % 2 != 0) {
        throw new IllegalArgumentException();
    }

    Stack<Integer> s = new Stack<Integer>();
    int size = q.size();

    for (int i = 0; i < size / 2; i++) {
        int element = q.remove();
        s.push(element);
        q.add(element);
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
    }

    for (int i = 0; i < size / 2; i++) {
        int element = q.remove();
        s.push(element);
        q.add(element);
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
}
```

```
public static void mirrorHalves(Queue<Integer> q) {
    if (q == null || q.size() % 2 != 0) {
        throw new IllegalArgumentException();
    }
    Stack<Integer> s = new Stack<Integer>();
    int size = q.size();
    for (int i = 1; i <= 2; i++) {
        while (s.size() < size / 2) {
            s.push(q.peek());
            q.add(q.remove());
        }
        s2q(s, q);
    }
}
```

#### 4. Four solutions are shown.

```
public static int rarestAge(Map<String, Integer> m) {
    if (m == null || m.isEmpty()) {
        throw new IllegalArgumentException();
    }
    Map<Integer, Integer> counts = new TreeMap<Integer, Integer>();
    for (String name : m.keySet()) {
        int age = m.get(name);
        if (counts.containsKey(age)) {
            counts.put(age, counts.get(age) + 1);
        } else {
            counts.put(age, 1);
        }
    }

    int minCount = m.size() + 1;
    int rareAge = -1;
    for (int age : counts.keySet()) {
        int count = counts.get(age);
        if (count < minCount) {
            minCount = count;
            rareAge = age;
        }
    }

    return rareAge;
}
```

```
public static int rarestAge(Map<String, Integer> m) {
    if (m == null || m.isEmpty()) {
        throw new IllegalArgumentException();
    }
    Map<Integer, Integer> counts = new TreeMap<Integer, Integer>();
    for (int age: m.values()) {
        if (!counts.containsKey(age)) {
            counts.put(age, 0);
        }
        counts.put(age, counts.get(age) + 1);
    }

    int rareAge = -1;
    for (int age : counts.keySet()) {
        int count = counts.get(age);
        if (rareAge < 0 || counts.get(age) < counts.get(rareAge)) {
            rareAge = age;
        }
    }
    return rareAge;
}
```

```

public static int rarestAge(Map<String, Integer> m) {
    if (m == null || m.isEmpty()) {
        throw new IllegalArgumentException();
    }
    Map<Integer, Integer> counts = new TreeMap<Integer, Integer>();
    for (String name: m.keySet()) {
        if (counts.containsKey(m.get(name))) {
            counts.put(m.get(name), counts.get(m.get(name)) + 1);
        } else {
            counts.put(m.get(name), 1);
        }
    }

    int minCount = 999999999; // really big number to be overwritten
    for (int age : counts.keySet()) {
        minCount = Math.min(minCount, counts.get(age));
    }

    for (int age : counts.keySet()) {
        if (counts.get(age) == minCount) {
            return age;
        }
    }
    return -1; // won't reach here
}

```

```

public static int rarestAge(Map<String, Integer> m) {
    if (m == null || m.isEmpty()) {
        throw new IllegalArgumentException();
    }
    Map<Integer, Integer> counts = new HashMap<Integer, Integer>();
    for (String name: m.keySet()) {
        if (!counts.containsKey(m.get(name))) {
            counts.put(m.get(name), 0);
        }
        counts.put(m.get(name), counts.get(m.get(name)) + 1);
    }

    int minCount = 999999999; // really big number to be overwritten
    for (int age : counts.keySet()) {
        minCount = Math.min(minCount, counts.get(age));
    }

    int rareAge = -1;
    for (int age : counts.keySet()) {
        if (counts.get(age) == minCount && (rareAge < 0 || age < rareAge)) {
            rareAge = age;
        }
    }
    return rareAge;
}

```

5. Four solutions are shown.

```
list.next.next.next = temp;           // 3 -> 4
temp.next.next = list.next.next;     // 5 -> 3
list.next.next = null;                // 2 /
ListNode temp2 = temp.next;          // temp2 -> 5
temp.next = list.next;                // 4 -> 2
list = temp2;                          // list -> 5
```

```
temp.next.next = list.next.next;     // 5 -> 3
list.next.next.next = temp;          // 3 -> 4
temp = temp.next;                     // temp -> 5
list.next.next.next.next = list.next; // 4 -> 2
list = temp;                           // list -> 5
list.next.next.next.next = null;      // 2 /
```

```
temp.next.next = list.next.next;     // 5 -> 3
list.next.next = temp;                // 2 -> 4
temp = temp.next;                     // temp -> 5
temp.next.next = list.next.next;     // 3 -> 4
temp.next.next.next = list.next;     // 4 -> 2
temp.next.next.next.next = null;     // 2 /
list = temp;                           // list -> 5
```

```
ListNode temp2 = list;                // temp2 -> 1
list = temp.next;                      // list -> 5
list.next = temp2.next.next;          // 5 -> 3
list.next.next = temp;                // 3 -> 4
list.next.next.next = temp2.next;     // 4 -> 2
list.next.next.next.next = null;     // 2 /
```

6. Four solutions are shown.

```
public void compress(int factor) {
    ListNode current = front;
    while (current != null) {
        int i = 1;
        ListNode current2 = current.next;
        while (current2 != null && i < factor) {
            current.data += current2.data;
            current.next = current.next.next;
            i++;
            current2 = current2.next;
        }
        current = current.next;
    }
}
```

```
public void compress(int factor) {
    ListNode current = front;
    while (current != null) {
        ListNode current2 = current.next;
        for (int i = 1; i < factor; i++) {
            if (current2 != null) {
                current.data += current2.data;
                current.next = current.next.next;
                current2 = current2.next;
            } else break; // break is optional
        }
        current = current.next;
    }
}
```

```
public void compress(int n) {
    if (front != null) {
        ListNode current = front;
        int i = 1;
        while (current.next != null) {
            if (i == n) {
                current = current.next;
                i = 1;
            } else {
                current.data += current.next.data;
                current.next = current.next.next;
                i++;
            }
        }
    }
}
```

```
public void compress(int n) {
    ListNode current = front;
    while (current != null && current.next != null) {
        for (int i = 1; i < n; i++) {
            current.data += current.next.data;
            current.next = current.next.next;
            if (current.next == null) {
                break;
            }
        }
        current = current.next;
    }
}
```

7. Two solutions are shown.

```
public class Food implements Comparable<Food> {
    ...

    public int compareTo(Food other) {
        // first compare by kind of food
        if (kind.toLowerCase().compareTo(other.getKind().toLowerCase()) > 0) {
            return 1;
        } else if (kind.toLowerCase().compareTo(
            other.getKind().toLowerCase()) < 0) {
            return -1;
        } else {
            // break tie by price
            if (price > other.getPrice()) {
                return 1;
            } else if (price < other.getPrice()) {
                return -1;
            } else {
                return 0;
            }
        }
    }
}

public class Food implements Comparable<Food> {
    ...

    public int compareTo(Food other) {
        int comp = kind.toLowerCase().compareTo(other.getKind().toLowerCase());
        if (comp != 0) {
            return comp;
        } else {
            return (int) Math.signum(price - other.getPrice());
        }
    }
}
```

8.

(a) Indexes examined: 5, 2, 3, 4  
Value returned: -5

(b) Initial array: {6, 3, 9, 7, 4, 1, 8, 2}  
after 1 pass: {1, 3, 9, 7, 4, 6, 8, 2}  
after 2 passes: {1, 2, 9, 7, 4, 6, 8, 3}  
after 3 passes: {1, 2, 3, 7, 4, 6, 8, 9}

(c) {6, 3, 9, 7, 4, 1, 8, 2}  
{6, 3, 9, 7} {4, 1, 8, 2}  
{6, 3} {9, 7} {4, 1} {8, 2}  
{6}{3} {9}{7} {4}{1} {8}{2}  
{3, 6} {7, 9} {1, 4} {2, 8}  
{3, 6, 7, 9} {1, 2, 4, 8}  
{1, 2, 3, 4, 6, 7, 8, 9}

9.

Call	Output
mystery(7);	7
mystery(825);	258
mystery(38947);	47893
mystery(612305);	0523610
mystery(-12345678);	-785634120



## 10. Four solutions are shown.

```
public static boolean isReverse(String s1, String s2) {
    if (s1.length() == 0 && s2.length() == 0) {
        return true;
    } else if (s1.length() == 0 || s2.length() == 0) {
        return false; // not same length
    } else {
        String s1first = s1.substring(0, 1);
        String s2last = s2.substring(s2.length() - 1);
        return s1first.equalsIgnoreCase(s2last) &&
            isReverse(s1.substring(1), s2.substring(0, s2.length() - 1));
    }
}

public static boolean isReverse(String s1, String s2) {
    if (s1.length() != s2.length()) {
        return false; // not same length
    } else if (s1.length() == 0 && s2.length() == 0) {
        return true;
    } else {
        s1 = s1.toLowerCase();
        s2 = s2.toLowerCase();
        return s1.charAt(0) == s2.charAt(s2.length() - 1) &&
            isReverse(s1.substring(1, s1.length()),
                s2.substring(0, s2.length() - 1));
    }
}

public static boolean isReverse(String s1, String s2) {
    if (s1.length() == s2.length()) {
        return isReverse(s1.toLowerCase(), 0, s2.toLowerCase(), s2.length() - 1);
    } else {
        return false; // not same length
    }
}

private static boolean isReverse(String s1, int i1, String s2, int i2) {
    if (i1 >= s1.length() && i2 < 0) {
        return true;
    } else {
        return s1.charAt(i1) == s2.charAt(i2) &&
            isReverse(s1, i1 + 1, s2, i2 - 1);
    }
}

public static boolean isReverse(String s1, String s2) {
    return reverse(s1.toLowerCase()).equals(s2.toLowerCase());
}

private static String reverse(String s) {
    if (s.length() == 0) {
        return s;
    } else {
        return reverse(s.substring(1)) + s.charAt(0);
    }
}
```