Q1 Excel with Integrity Pledge

0 Points

I will complete this exam in a fair, honest, respectful, responsible and trustworthy manner. This means that I will complete the exam as if the professor was watching my every action. I will act according to the professor's instructions, and I will neither give nor receive any aid or assistance other than what is authorized. I know that the integrity of this exam and this class is up to me, and I pledge to not take any action that would break the trust of my classmates or professor, or undermine the fairness of this class.

I promise to complete this exam in keeping with the Excel with Integrity Pledge.

Fill in your Name and today’s Date below:

Sophia Yermolenko 6/7/2022

Q2 Instructions

0 Points

You have until 8 AM on Friday 6/10 to complete Final Exam Part 2. This exam is untimed. Work to maximize points. If you get stuck, work through other problems and come back to it.

In general, if you think you've spotted a typo in the exam, do your best to answer in the spirit of the question. Keep in mind that some questions have interesting code examples with intentional bugs for you to find as part of the question. **Questions asked during the exam will not be answered, so do your best to interpret the questions.**

In general, assume that any necessary libraries (JUnit, ArrayList, List, and so on) have been imported.

You can use your notes and a compiler. However, the test is designed as if you were taking it during lecture without a compiler and it's highly suggested you not rely on your compiler and use pen & paper to figure out your answers.

Stay calm – you can do this!

Q3 Partition

6 Points

Consider this specification for partition:

```
int partition(int[] numbers)
```

Partition() method description:
Chooses a pivot value from the array. Then, changes the array so that all elements smaller than or equal to the pivot appear in indices less than the index of the pivot value, and all elements larger than the pivot appear in indices greater than the index of the pivot value. Returns the final index of the pivot value, which may be different than its starting index. All elements in the input should appear at some index in the output.

Q3.1 A

2 Points

Consider this *input* array:

```
{ 61, 33, 11, 87, 48, 72 }
```

Consider the following array *after* a call to **partition()**, for the input array above. What are **all** the indices that could have been **returned** from **partition()** for this to be a valid partition result? Select **all** possible indices.

```
{ 33, 11, 48, 87, 61, 72 }
```

- □ 0
- □ 1
- ☑ 2
- □ 3
- □ 4
- □ 5

Q3.2 B

2 Points

Consider this *input* array:

```
{ 61, 33, 11, 87, 48, 72 }
```

Consider the following array *after* a call to **partition()**, for the input array above. What are **all** the indices that could have been **returned** from **partition()** for this to be a valid partition result? Select **all** possible indices.

```
{ 11, 48, 33, 72, 61, 87 }
```

☑ 0

□ 1

□ 2

□ 3

□ 4

☑ 5

Q3.3 C

2 Points

Consider this *input* array:

```
{ 61, 33, 11, 87, 48, 72 }
```

Consider the following array *after* a call to **partition()**, for the input array above. What are **all** the indices that could have been **returned** from **partition()** for this to be a valid partition result? Select **all** possible indices.

```
{ 33, 11, 48, 61, 87, 72 }
```

☐ 0

☐ 1

☑ 2

☑ 3

☐ 4

☐ 5

Q4 More partition

3 Points

Consider this specification for **partition2()**:

```
int partition2(int[] numbers)
```

Description

Chooses a pivot value from the array at random. Then, changes the array so that all elements **smaller than or equal** to the pivot appear in indices **less than the index** of the pivot value, and all elements larger than the pivot appear in indices greater than the index of the pivot value. Returns the final index of the pivot value, which may be different than its starting index. All elements in the input should appear at some index in the output.

Consider the following output array states and return values from **partition2()**. Which are possible valid outputs of partition for some input? We intentionally don't show the input. Note that each could come from a different implementation of and call to **partition2()**.

Select **all** the possible answers corresponding to the valid results below. If none are valid, select None.

☑ Resulting array: { 0, 1, 2, 3, 4, 5 }; Return value: 5

☑ Resulting array: { 103, 201, 304, 705, 504, 401 }; Return value: 2

□ Resulting array: { 22, 14, 75, 54, 41, 36 }; Return value: 0

☑ Resulting array: { 4, 2, 11, 8, 11, 65 }; Return value: 4

□ Resulting array: { 23, 23, 64, 44, 31, 26 }; Return value: 3

□ Resulting array: { 13, 32, 51, 46, 64, 55 }; Return value: 5

□ None

Q5 Run-time

9 Points

Recall that we say *f is O(g)* if there exist *n0* and *C* such that for all *n > n0*, *f(n) < C * g(n)*

Recall that Θ (big-Theta) represents a *tight* bound, O (big-O) an *upper* bound, and Ω (big-Omega) a *lower* bound. (We intentionally do not give definitions for Θ and Ω).

Q5.1 1

1 Point

Give a Θ bound for the number of steps the following program takes in terms of n, assuming **doSomeWork()** does a constant amount of work.

```
for(int i = 0; i <= n * n; i = i + 2) {
  for(int j = n; j > 0; j -= 1) {
    doSomeWork();
  }
}
```

○ $\Theta(1)$

○ $\Theta(\log n)$

○ $\Theta(n)$

○ $\Theta(n \log n)$

○ $\Theta(n^2)$

● $\Theta(n^3)$

○ $\Theta(2^n)$

○ $\Theta(n!)$

Q5.2 2

1 Point

Give a Θ bound for the number of steps the following program takes in terms of n, assuming **doSomeWork()** does a constant amount of work.

```
for(int i = 0; i < n / 2; i += 1) {
  for(int j = 1; j < n; j = j * 2) {
    doSomeWork();
  }
}
```

○ $\Theta(1)$

○ $\Theta(\log n)$

○ $\Theta(n)$

● $\Theta(n \log n)$

○ $\Theta(n^2)$

○ $\Theta(n^3)$

○ $\Theta(2^n)$

○ $\Theta(n!)$

Q5.3 3

1 Point

Give a Θ bound in terms of n for the number of steps the following method **foo()** takes in the worst case, assuming **doSomeWork()** does a constant amount of work.

```
void foo(int n, int k) {
  for(int i = 0; i < n; i = i + 1) {
    if(i == k) {
      for(int j = 0; j < i; j += 1) {
        doSomeWork();
      }
    }
  }
}
```

○ $\Theta(1)$

○ $\Theta(\log n)$

● $\Theta(n)$

○ $\Theta(n \log n)$

○ $\Theta(n^2)$

○ $\Theta(n^3)$

○ $\Theta(2^n)$

○ $\Theta(n!)$

Q5.4 4

1 Point

Give a Θ bound in terms of n for the number of steps the following method **foobar()** takes in the worst case.

```
int foobar(int n) {
  if (n <= 1) { return 1; }
  else if (n % 2 == 0) {
    return foobar(n / 2) * 10;
  }
  else {
    return foobar(n / 2) * 10 + 1;
    }
  }
```

○ $\Theta(1)$

● $\Theta(\log n)$

○ $\Theta(n)$

○ $\Theta(n \log n)$

○ $\Theta(n^2)$

○ $\Theta(n^3)$

○ $\Theta(2^n)$

○ $\Theta(n!)$

Q5.5 5a

1 Point

Consider the following pair of functions. Indicate whether *f is O(g)* and/or *f is Ω(g)*. Select one, both or the choice of Neither is correct as appropriate.

$f(x) = log(x) + 4 * x$ $g(x) = 10 * x * log(x)$

☑ *f is O(g)*

□ *f is Ω(g)*

□ Neither is correct

Q5.6 5b

1 Point

Consider the following pair of functions. Indicate whether *f is O(g)* and/or *f is Ω(g)*. Select one, both or the choice of Neither is correct as appropriate.

$f(x) = x^2 + x + log(x)$ $g(x) = x * log(x) + x^2$

☑ *f is O(g)*

☑ *f is Ω(g)*

□ Neither is correct

Q5.7 5c

1 Point

Consider the following pair of functions. Indicate whether *f is O(g)* and/or *f is Ω(g)*. Select one, both or the choice of Neither is correct as appropriate.

$f(x) = x * log(x)$ $g(x) = x * log(x) + 4 * x$

☑ *f is O(g)*

☑ *f is Ω(g)*

□ Neither is correct

Q5.8 5d

1 Point

Consider the following pair of functions. Indicate whether *f is O(g)* and/or *f is Ω(g)*. Select one, both or the choice of Neither is correct as appropriate.

$f(x) = \frac{x * (x-1)}{2}$ $g(x) = 5 * x * log(x)$

☐ *f is O(g)*

☑ *f is Ω(g)*

☐ Neither is correct

Q5.9 5e

1 Point

Consider the following pair of functions. Indicate whether *f is O(g)* and/or *f is Ω(g)*. Select one, both or the choice of Neither is correct as appropriate.

$$f(x) = \frac{x * (x-1)}{x} \qquad g(x) = 3 * x * log(x)$$

☑ *f is O(g)*

☐ *f is Ω(g)*

☐ Neither is correct

Q6 HashTable

5 Points

Consider this implementation of a hash table based on one from the lecture notes and Exam 2 (there are no differences or surprises).

```
interface Hasher<K> { int hash(K key); }
class HashTable<K,V> {
  class Entry {
    K k; V v;
    public Entry(K k, V v) { this.k = k; this.v = v; }
  }

  List<Entry>[] buckets; // An array of Lists of Entries
  int size;
  Hasher<K> hasher;

  public HashTable(Hasher<K> h, int startCapacity) {
    this.size = 0;
    this.hasher = h;
    this.buckets = (List<Entry>[])(new List[startCapacity]);
  }

  public double loadFactor() {
    return (double)(this.size) / this.buckets.length;
  }

  public V get(K k) {
    int hashCode = this.hasher.hash(k);
    int index = hashCode % this.buckets.length;
    if(this.buckets[index] == null) {
      return null;
    }
    else {
      for(Entry e: this.buckets[index]) {
        if(e.k.equals(k)) { return e.v; }
      }
      return null;
    }
  }

  public void set(K k, V v) {
    int hashCode = this.hasher.hash(k);
    int index = hashCode % this.buckets.length;
    if(this.buckets[index] == null) {
      this.buckets[index] = new ArrayList<Entry>();
      this.buckets[index].add(new Entry(k, v));
    }
    else {
      for(Entry e: this.buckets[index]) {
        if(e.k.equals(k)) { e.v = v; return; }
      }
      this.buckets[index].add(new Entry(k, v));
    }
    this.size += 1;
  }
}
```

Consider adding this NEW method, keys(), to the hash table:

```
  public List<K> keys() {
    List<K> keysList = new ArrayList<>();
    for(List<Entry> items: buckets) {
      if(items == null) { continue; }
      for(Entry e: items) { keysList.add(0, e.k); }
    }
```

```
        return keysList;
    }
```

Q6.1 Runtime of keys

1 Point

What is the **worst-case** runtime of **keys()** in terms of the number of entries in the hash table? Give your answer as a tight big-O bound. Assume `keysList` is a circular array list, i.e. to insert an element, $O(1)$ time is required.

○ $\Theta(1)$

○ $\Theta(\log n)$

○ $\Theta(n)$

○ $\Theta(n \log n)$

● $\Theta(n^2)$

○ $\Theta(2^n)$

○ $\Theta(n^3)$

Q6.2 Implementation of keys

2 Points

Which of the following is **true** of this implementation of **keys()**? Choose ALL that apply.

□ In some cases when there are hash collisions, the result of keys() will not include some keys that are in the table.

☑ Its best case has a different tight big-O bound on number of steps taken than its worst case.

□ No matter what order the keys are added in, the keys returned will be in the same order in which they were added to the hash table.

☑ If we call keys() twice on a hash table without inserting or removing elements, the keys will be returned in the same order the second time.

□ No matter what order the keys are added in, they will be returned in sorted order from lowest to highest according to a consistent Comparator

□ None

Q6.3 Method body for test demonstrating different orders

2 Points

Write a test that demonstrates that the following statement is **not true**:

If two HashTables contain the same keys and use the same hash method, then calling the **keys()** method on both will produce the same keys in the same order.

You can assume the following starter code and fill in the blank (assume appropriate JUnit libraries are imported):

```
class HashLength implements Hasher<String> {
  public int hash(String s) { return s.length(); }
}

class TestHashTable {
  @Test
  public void testHashTable() {
  /* ... fill in here ... */
  }
}
```

If you need to use `<` or `>`, Use spaces before and/or after the `<` and the `>`. Otherwise, the `<` and `>` will disappear.

```
HashTable < String, Integer > firstHashTable = new HashTable(new HashLength(), 3);
HashTable < String, Integer > secondHashTable = new HashTable(new HashLength(), 3);

firstHashTable.set("Hello", 2);
firstHashTable.set("Goodbye!", 5);
secondHashTable.set("Goodbye!", 5);
secondHashTable.set("Hello", 2);

List < String > firstKeys = firstHashTable.keys();
List < String > secondKeys = secondHashTable.keys();

for (int i = 0; i < firstKeys.size(); i++) {
	assertEquals(firstKeys.get(i), secondKeys.get(i));
}
```

Q7 Merge Sort

2 Points

Suppose we have a modified version of the merge sort called **threeMergeSort()**, and instead of cutting in the middle, it always cuts at the 1/3 location of the list (i.e. 1/3 of the list is in the front, and 2/3 are in the back). The **threeMergeSort()** method is given as follows and you can assume that **merge()** works well for this code.

```
public static void threeMergeSort(int[] in){
  if (in.length > 2){
    int mid = in.length/3;
    //left inclusive, right exclusive when using copyOfRange
    int[] front = Arrays.copyOfRange(in, 0, mid);
    int[] back = Arrays.copyOfRange(in, mid, in.length);
    threeMergeSort(front);
    threeMergeSort(back);
    merge(front, back, in);
  }
  else{
    if(in.length == 1){
      return;
    }
    if(in.length == 2){
      Arrays.sort(in);
    }
  }
}
```

Answer the following two questions about **threeMergeSort()**

Q7.1

1 Point

Please answer the following question about **threeMergeSort()** if we try to sort the following array

{9, 4, 5, 2, 3, 6, 1}

What is the correct collection of lowest-level sub-arrays that have reached base cases?

○ {9}, {4, 5}, {2}, {3}, {6, 1}

○ {9}, {4}, {5}, {2}, {3}, {6}, {1}

● {9, 4}, {5}, {2}, {3}, {6, 1}

○ {9}, {4, 5}, {2, 3}, {6, 1}

Q7.2

1 Point

Is it true that this **threeMergeSort()** has the same efficiency as the normal merge sort asymptotically?

◉ true

○ false

Final Exam - Part 2

● **GRADED**

STUDENT

Sophia Yermolenko

TOTAL POINTS

23 / 25 pts

QUESTION 1		
Excel with Integrity Pledge		**0** / 0 pts
QUESTION 2		
Instructions		**0** / 0 pts
QUESTION 3		
Partition		**6** / 6 pts
3.1	A	**2** / 2 pts
3.2	B	**2** / 2 pts
3.3	C	**2** / 2 pts
QUESTION 4		
More partition		**3** / 3 pts
QUESTION 5		
Run-time		**9** / 9 pts
5.1	1	**1** / 1 pt
5.2	2	**1** / 1 pt
5.3	3	**1** / 1 pt

5.4	4	**1** / 1 pt
5.5	5a	**1** / 1 pt
5.6	5b	**1** / 1 pt
5.7	5c	**1** / 1 pt
5.8	5d	**1** / 1 pt
5.9	5e	**1** / 1 pt

QUESTION 6

HashTable **3** / 5 pts

6.1	Runtime of keys	**0** / 1 pt
6.2	Implementation of keys	**1** / 2 pts
6.3	Method body for test demonstrating different orders	**2** / 2 pts

QUESTION 7

Merge Sort **2** / 2 pts

7.1	(no title)	**1** / 1 pt
7.2	(no title)	**1** / 1 pt