CSE 12 SP24 Exam 2

Do not begin until instructed.

This exam is closed book, closed notes. No study aids or electronic devices are allowed. Turn off your phone and have no bags open.

You must hand in all sheets, including scratch paper, at the end of the exam.

This exam is designed to assess **your** understanding of topics from CSE12. You cannot communicate with other students during the exam. You cannot discuss this exam with anyone from now until you receive your grade on the exam. This includes posting about the exam on Piazza or elsewhere online.

Please write “I excel with integrity” in the box below:

Sign your name: _____

Print your name: _____

Print your email: _____

PID: _____

You have 45 minutes to complete the exam. Work to maximize points. **Write all your answers on the provided answer sheet on the back of this page.** If you get stuck, work through other problems, and come back to it.

Once the exam starts, you can rip off/remove the answer sheet.

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. If you ask a question during the exam, we may decline to answer. In general, assume that any necessary libraries (JUnit, ArrayList, List, and so on) have been imported.

Stay calm – you can do this!

Do not begin until instructed.

Write all answers below. There are 24 total points.

Question 1 (6 points, 2 points each) Write the possible return values from partition; list **all** that apply for each

A
B
C

Question 2 (7 points)
Parts 1 & 2 (1 point each) Give the tight Θ bound

1.	2.

Part 3 (5 points, 1 point each) circle neither, one, or both as appropriate

A O Ω	**B** O Ω	**C** O Ω	**D** O Ω	**E** O Ω

Question 3 (5 points)
Part 1 (2 points. Write code to complete minQsort, two lines only)

Part 2-4 (1 point each. Write the tight bound)

Worst (partition)	**Worst (minQsort)**	**Best (minQsort)**

Question 4 (6 points)
Part 1 (Write the **output** for the given Hasher) 3 points each

HasherA	**HasherB**

Part 2 (Write the length of the longest chain) 1 point each

HasherA	**HasherB**

Part 3 (What makes HasherA bad? Circle letter(s) that apply) 2 points

A B C D

Question 1

Consider this input array:

```
{ 12, 35, 48, 24, 9, 24 }
```

Consider this specification for *partition*:

> **`int partition(int[] numbers)`**
>
> Chooses a pivot value from the array. Then, changes the array so that all elements **smaller than** the pivot appear in indices less than the index of the pivot value, and all elements **larger than or equal** to 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 from its starting index. All elements in the input array should appear at some index in the output.

For the input array above, consider each of the following arrays *after* a call to `partition`. What are **all** the indices that could have been **returned** from `partition` for this to be a valid partition result? Write **all** possible **indices** (not values) as numbers directly on the answer sheet for each.

A. `{ 9, 12, 24, 35, 48, 24 }`
B. `{ 24, 9, 12, 35, 24, 48 }`
C. `{ 9, 24, 24, 12, 48, 35 }`

Question 2

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).

1. 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 = n; i > 1; i = i / 3) {
  for (int j = 0; j < n; j++) {
    doSomeWork();
  }
}
```

2. 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 = n; i >= 1; i -= 2) {
  for(int j = i; j >= 0; j -= 3) {
    doSomeWork();
  }
}
```

3. Consider the following pairs of functions. For each, indicate whether $f\ is\ O(g)$ and/or $f\ is\ \Omega(g)$. Circle **zero**, **one**, or **both** of the letters on the answer sheet as appropriate.

A. $f(x) = x^{3/2}$ $\quad$ $g(x) = x^{5/4}$

B. $f(x) = x^2 + x + 5$ $\quad$ $g(x) = \frac{x * (x^2 + 10)}{x}$

C. $f(x) = 2^{2x}$ $\quad$ $g(x) = x^{200}$

D. $f(x) = \sqrt{\frac{x * (x+3)}{2}}$ $\quad$ $g(x) = log(x) * log(x)$

E. $f(x) = 50! + x * log(x) + x^3$ $\quad$ $g(x) = x^2 * log(x) + x$!This page is intentionally left blank.

This page is intentionally left blank.

Question 3

Consider these helper methods for the below Quicksort implementation that partitions an array such that the element with the **minimum** value is always chosen as the pivot, regardless of the order of the elements:

```
// finds the minimum element in arr between low and high-1
// and swaps it with the element at index low
int findMin(int[] arr, int low, int high) {
  int min = arr[low];
  int indexOfMin = low;
  for (int i = low; i < high; i++) {
    if (arr[i] < min) {
      min = arr[i];
      indexOfMin = i;
    }
  }
  swap(arr, indexOfMin, low);
  return low;
}

void swap(int[] arr, int i1, int i2) {
  int temp = arr[i1];
  arr[i1] = arr[i2];
  arr[i2] = temp;
}

int partition(int[] arr, int low, int high) {
  int pivotStartIndex = findMin(arr, low, high);
  int pivot = arr[pivotStartIndex];
  int smallerBefore = low + 1, largerAfter = high - 1;

  while (smallerBefore <= largerAfter) {
    if (arr[smallerBefore] < pivot) {
      smallerBefore += 1;
    } else {
      swap(arr, smallerBefore, largerAfter);
      largerAfter -= 1;
    }
  }

  swap(arr, low, smallerBefore - 1);
  return smallerBefore - 1;
}
```

1. Consider this method skeleton:

```
void minQsort(int[] arr, int low, int high) {
  if (high - low <= 1) { return; }
  int splitAt = partition(arr, low, high);
  // write in answer sheet
  // write in answer sheet
}
```

   Fill in the **two lines** that would complete the method and cause it to always change the input array to be in **increasing sorted order**. Write code for this directly on the answer sheet (only write two lines of code, not the entire method).

2. Give a Θ bound (a tight bound) for the number of steps the partition() method takes in terms of n in the **worst case** arrangement of the elements in `arr`, and assuming `arr.length` is at least as large as n.

3. Give a Θ bound (a tight bound) for the number of steps the completed minQsort() method you wrote takes in terms of `arr.length` in the **worst case** arrangement of the elements in `arr`, for input arrays of arbitrary size.

4. Give a Θ bound (a tight bound) for the number of steps the completed minQsort() method you wrote takes in terms of `arr.length` in the **best case** arrangement of the elements in `arr`, for input arrays of arbitrary size.

Question 4

Consider this implementation of a hash table based on one from the lecture. Bolded portions are especially relevant.

```
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 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 these implementations of the `Hasher` interface:

```
class HasherA implements Hasher<String> {
  public int hash(String s) { return (s.length() * 5) % 2; }
}

class HasherB implements Hasher<String> {
  int num;
  public HasherB() { this.num = 1; }
  public int hash(String s) {
       int hashValue = num + s.length();
       if (Character.isUpperCase(s.charAt(0))) {
          num += 2;
       }
       return hashValue;
  }
}
```

Recall that `s.charAt(i)` retrieves the character at index `i` of string `s`. `Character.isUpperCase(c)` returns true if character `c` is an uppercase character and false otherwise.

Consider this sequence of operations:

```
1   HashTable<String, Integer> ht = new HashTable<>(???, 6);
2   ht.set("Red", 300);
3   System.out.println(ht.size);
4   ht.set("brown", 500);
5   ht.set("Blue", 400);
6   System.out.println(ht.size);
7   System.out.println(ht.get("Red"));
8   ht.set("brown", 200);
9   System.out.println(ht.size);
10  System.out.println(ht.get("brown"));
11  ht.set("Blue", 600);
12  System.out.println(ht.size);
```

Consider using each of the two Hasher implementations to replace the **???** above by filling it in with either `new HasherA()` or `new HasherB()`.

1. For each Hasher, write what would print directly in the answer sheet (6 lines of output each)

2. For each Hasher, at the end of running all the statements, what would be the length of the longest List in the entries array? Write your answer as a number directly on the answer sheet.

3. Ignore the fact that HasherA is simpler than one that would be used in the real world. What makes HasherA a **bad/subpar** hashing algorithm for hash tables? **Circle the letters** of all that apply.
    A. HasherA prevents retrieving the key from the hash value.
    B. HasherA generates the same hash value from different keys.
    C. HasherA generates different hash values from the same key.
    D. HasherA creates a nonuniform distribution in the hash table.

Scratch Paper