1. **Basic Operations**

1. **Min Heap**

   (a) Draw the Min Heap that results if we delete the smallest item from the heap.

   ![Min Heap Diagram](image)

   (b) Draw the Min Heap that results if we insert the elements 6, 5, 4, 3, 2 into an empty heap.

   (c) Given an array, heapify it such that after heapification it represents a Max Heap.

   ```
   int[] a = {5, 12, 64, 1, 37, 90, 91, 97}
   ```

2. **External Chaining**

   Consider the following External Chaining Hash Set below, which doubles in size when the load factor reaches 1.5. Assume that we’re using the default hashCode for integers, which simply returns the integer itself.

   ![External Chaining Diagram](image)
(a) Draw the External Chaining Hash Set that results if we insert 18.

(b) Draw the External Chaining Hash Set that results if we insert 5 after the insertion done in part (a).

2 Invalid Hashes

Which of the hashCodes are invalid? Assume we are trying to hash the following class:

```java
import java.util.Random;

class Point{
    private int x;
    private int y;
    private static count = 0;

    public Point(int x, int y){
        this.x = x;
        this.y = y;
        count += 1;
    }
}
```

(a) public void hashCode() { System.out.print(this.x + this.y); }

(b) public int hashCode() {
    Random randomGenerator = new Random();
    return randomGenerator.nextInt(Int);
}

(c) public int hashCode() { return this.x + this.y; }

(d) public int hashCode() { return count; }

(e) public int hashCode() { return 4; }

3 Search structure Runtimes

Assume you have $N$ items. Using Theta notation, find the worst case runtime of each function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Unordered List</th>
<th>Sorted Array</th>
<th>Bushy Search Tree</th>
<th>&quot;Good&quot; Hash Table</th>
<th>Max Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add (Amortized)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>find largest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove largest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4 Range Query

Write a function that counts the number of elements greater than a given number in a binary search tree. Assume the BST node has methods `label()`, `left()`, and `right()`.

```java
static int findGreater(BST<Integer> T, int x) {
    if (______________) {
        return ________
    }
    if (__________________) {
        return ____________________________________________;
    } else {
        return ____________________________________________;
    }
}
```
5 Bits (Midterm Review)

Fill in the following function that returns true if the input, an unsigned integer, is 1 more or 1 less than a power of 2. Use bit operations. You may not declare booleans, nor may you assign static numerical values to int variables (i.e. int y = 5).

```java
static boolean almostPowerOfTwo(int x) {
    if (_______________) {
        return false;
    }
    x = x >>> 1;
    ______________;
    if (_______________) {
        while (_______________) {
            if (_______________) {
                return false;
            }
            x = x >>> 1;
        }
    } else {
        ______________;
        while (_______________) {
            ______________;
            x = x >>> 1;
            if (_______________ && (_______________)) {
                return false;
            }
        }
    }
    return true;
}
```

6 Asymptotics (Midterm Review)

Find the tightest possible bound for the given functions below. For a), assume that instead of using linked lists for hashing, we are using BST. Do not assume anything about the hashing function.

(a) __________ private static void f(int n) {
    Random generator = new Random();
    Hashset<Integer> hash = new Hashset<Integer>();
    for(int i = 0; i < n; i++) {
        if(generator.nextBoolean()){
            hash.add(n-i);
        }
    }
    if(hash.contains(5)){
        f(n-1);
private static void f1(int n) {
    int a = 0;
    int b = 0;
    while (b < n) {
        a += 1;
        b = b + a;
    }
}

private static void f2(int a, int b) {
    if(a <= 0) return;
    for(int i = 0; i < b; i+=1){
        f2(a-1, b);
    }
}