CS 61B Heaps and Hashing Fall 2021

1 Heaps of Fun

(a) Consider an array-based min-heap with N elements. What is the worst case asymptotic runtime of each of the following operations if we ignore resizing? What is the worst case asymptotic runtime if we take resizing into account?

	Without Resizing	With Resizing
Insert		
Find Min		
Remove Min		

(b) What are the advantages of using an array-based heap over a pointer-based heap?

(c) How can you implement a max-heap of integers if you only have access to a min-heap?

(d) Given an array and a min-heap, describe an algorithm that would allow you to sort the elements of the array in ascending order. Give the best and worst case runtime of your algorithm.

2 HashMap Modification (61BL Summer 2010, MT2)

(a) If you modify a **key** that has been inserted into a `HashMap`, can you retrieve that entry again? Explain.

☐ Always ☐ Sometimes ☐ Never

(b) If you modify a **value** that has been inserted into a `HashMap`, can you retrieve that entry again? Explain.

☐ Always ☐ Sometimes ☐ Never

3 Hash Code

In order for a hash code to be valid, objects that are equivalent to each other (i.e. `.equals()` returns true) must return equivalent hash codes. If an object does not explicitly override the `hashCode()` method, it will inherit the `hashCode()` method defined in the `Object` class, which returns the object's address in memory.

Here are four potential implementations of `Integer`'s `hashCode()` function. Assume that `intValue()` returns the value represented by the `Integer` object. Categorize each `hashCode()` implementation as either a valid or an invalid hash function. If it is invalid, explain why. If it is valid, point out a flaw or disadvantage.

```
(1) public int hashCode() {
        return -1;
    }
```

```
(2) public int hashCode() {
        return intValue() * intValue();
    }
```

```
(3) public int hashCode() {
        Random rand = new Random();
        return rand.nextInt();
    }
```

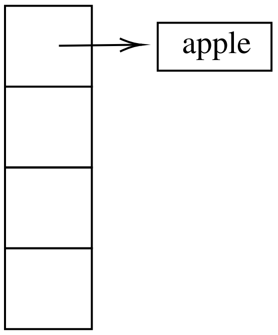
```
(4) public int hashCode() {
        return super.hashCode();
    }
```

4 Hashing Practice

Given the provided `hashCode()` implementation, hash the items listed below with external chaining (the first item is already inserted for you). Assume the load factor is 1. Use geometric resizing with a resize factor of 2. You may draw more boxes to extend the array when you need to resize.

```
/** Returns 0 if word begins with 'a', 1 if it begins with 'b', etc. */
public int hashCode() {
    return word.charAt(0) - 'a';
}

["apple", "cherry", "fig", "guava", "durian", "apricot", "banana"]
```



Extra: Suppose that we represent Tic-Tac-Toe boards as 3×3 arrays of integers (with each integer in the range [0, 2] to represent blank, 'X', and 'O', respectively). Describe a hash function for Tic-Tac-Toe boards that are represented in this way such that boards that are not equal will never have the same hash code.