1 Heap

(a) Suppose we have the min heap below, with array representation as shown. Show the heap and array representation after the smallest value is removed, using the procedure described in class.

We first swap the root and the last leave (arr[1] and arr[10]). Then, we can safely remove the last leave at arr[10], which now stores 0 (the smallest value), while holding the tree completeness. Then, we sink 6 from the root to its proper place, as shown above.

2 Left-Leaning Red-Black Tree

LLRB Tree Invariants:
- Every root-to-leaf path has the same number of black edges.
- Red edges lean left.
- No node has two red edges connected to it, either above/below or left/right.

LLRB Tree Problems and Solutions:
- Red link to right child and no red link to left child?: Rotate left.
- Two left reds in a row?: Rotate right.
- Red links to both children?: Flip colors.

(a) Draw the 2-3 tree corresponding to the following left-leaning red-black tree.
(b) Draw the left-leaning red-black tree after inserting 5. Label red edges red.

Another approach is to insert 5 into the corresponding 2-3 tree (from 2(a)) and convert it back to LLRB tree.
3 Hashing

For the following problems, assume that..

- **IntList** is a list of integers.
- The **hash code** of an **IntList** is the sum of the integers in the list.
- **IntLists** are considered equal only if they have the same size and the same values in the same order.
- **FourBucketHashMap** uses separate chaining and that new items are added to the **back** of each bucket.
- **FourBucketHashMap** always has **four** buckets and never resizes.

(a) Draw the hash table that is created by the following code. The result of the first put is provided for you.

```java
FourBucketHashMap<IntList, String> fbhm = new FourBucketHashMap<>();
fbhm.put(IntList.of(1, 2), "dog");
fbhm.put(IntList.of(3, 1), "bear");
fbhm.put(IntList.of(9), "rat");
fbhm.put(IntList.of(3, 3, 2), "tiger");
```

(b) Consider the following code:

```java
IntList list1 = IntList.of(1, 2);
FourBucketHashMap<IntList, String> fbhm = new FourBucketHashMap<>();
fbhm.put(list1, "dog");
\// Part i
list1.add(3);\// Part ii
```

i) At Part i (line 4), what will be returned from the following statement?

At line 4, our hash table looks like:

```
0 → [3, 1], "bear" → [3, 3, 2], "tiger"
1 → [9], "rat"
2 →
3 → [1, 2], "dog"
```
fbhm.get(IntList.of(1, 2));  ○ "dog" ○ [1, 2] ○ null

This will look up the bucket (1 + 2) mod 4 = 3. In the bucket 3, IntList.of(1, 2) is equivalent to [1, 2], so "dog" which is the stored value is returned.

ii) At Part ii (line 6), what will be returned from the following statements?

At line 6, our hash table looks like:

```
<table>
<thead>
<tr>
<th>Bucket</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>→</td>
</tr>
<tr>
<td>1</td>
<td>→</td>
</tr>
<tr>
<td>2</td>
<td>→</td>
</tr>
<tr>
<td>3</td>
<td>→ [1, 2, 3], &quot;dog&quot;</td>
</tr>
</tbody>
</table>
```

fbhm.get(IntList.of(1, 2));  ○ "dog" ○ [1, 2] ○ null

This will look up the bucket (1 + 2) mod 4 = 3. In the bucket 3, IntList.of(1, 2) is NOT equivalent to [1, 2, 3], so we cannot find the matched key. Hence, return null.

fbhm.get(IntList.of(1, 2, 3));  ○ "dog" ○ [1, 2, 3] ○ null

This will look up the bucket (1 + 2 + 3) mod 4 = 2. Since the bucket 2 is empty, we definitely cannot find the matched key. Hence, return null.

iii) Is there a problem with the code? If so, explain below.

Adding 3 into list1 changes its hash code, causing list1 to live in the wrong bucket.