Jay Offerdahl
EECS 560
Lab 13/14 Report Part 1

Verifying the Construction of a Pairing Heap

After writing the code to insert into a pairing heap, I am confident that I have properly constructed the data structure. I have setup a basic GUI in my main.cpp class to handle user interaction, and through this, I was able to test several different cases of input while building a pairing heap. For example, when inserting values in decreasing order, the output will simply be a heap with nodes having left children down to the first value inserted. Please note, my output below shows numbers in parentheses which denotes the value of the left child carried by that node.

Inserting 10, 9 , 8, 7, 6, 5, 4, 3, 2, 1:

```
Level 0:  1 (2) -> *
Level 1:  2 (3) -> *
Level 2:  3 (4) -> *
Level 3:  4 (5) -> *
Level 4:  5 (6) -> *
Level 5:  6 (7) -> *
Level 6:  7 (8) -> *
Level 7:  8 (9) -> *
Level 8:  9 (10) -> *
Level 9:  10 -> *
Level 10:  *
```

When inserting in increasing order, the smallest value will always stay at the root, and all of the following values will be sent to the second level, as shown below:

Inserting 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

```
Level 0:  1 (10) -> *
Level 1:  10 -> 9 -> 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> *
Level 2:  *  *  *  *  *  *  *  *  *
```

Next, to generate a more complex heap, I tried inserting values which would cause more ‘zig-zags’.

Inserting 10, 11, 12, 9, 6, 7, 8, 3, 5, 4, 1, 2

```
Level 0:  1 (2) -> *
Level 1:  2 -> 3 (4) -> *
Level 2:  *  4 -> 5 -> 6 (8) -> *
Level 3:  *  *  8 -> 7 -> 9 (10) -> *
Level 4:  *  *  10 (12) -> *
Level 5:  12 -> 11 -> *
Level 6:  *  *
```

The pairing heap generated here is correct, and we can tell because of the helpful (num) statements. 1 is the root with a left child 2, 2 has right sibling 3 which has left child 4, and so on. At this stage it’s not possible to make more complex trees because inserting does not use the combine siblings method. With that said, zig-zag trees are the best form of testing at this point.