3134 Data Structures Homework 4 Written Solutions

1. **(7 pts): Weiss, Exercise 5.1**

(a) Separate chaining hash table

0-
1 - 4371
2-
3 - 1323 -> 6173
4 - 4344
5-
6-
7-
8-
9 - 4199 -> 9679 -> 1989

(b) Hash table using linear probing

0 - 9679
1 - 4371
2 - 1989
3 - 1323
4 - 6173
5 - 4344
6-
7-
8-
9 - 4199

(c) Hash table using quadratic probing

0 - 9679
1 - 4371
2-
3 - 1323
4 - 6173
5 - 4344
6-
7-
8 - 1989
9 - 4199

(d) Hash table with second hash function h2(x) = 7 – (x mod 7)

0-
1 - 4371
2-
3 - 1323
4 - 6173
5 - 9679
6-
7 - 4344
8-
9 - 4199

*1989 cannot be inserted

2. **Weiss, Exercise 5.2**

(a) Separate chaining hash table

0 - 4199
1 - 4371
2
3
4
5
6
7
8 - 9679
9
10
11
12 - 1323 --> 4344
13 - 1989
14
15
16
17 - 6173
18

(b) Hash table using linear probing

0	4199
1	4371
2	
3	
4	
5	
6	
7	
8	9679
9	
10	
11	
12	1323
13	1989
14	4344
15	
16	
17	6173
18	

(c) Hash table using quadratic probing

0	4199
1	4371
2	
3	
4	
5	
6	
7	
8	9679
9	
10	
11	
12	1323
13	4344
14	1989
15	
16	
17	6173
18	

(d) Hash table with second hash function h2(x) = 7 – (x mod 7)

0	- 4199
1	- 4371
2	
3	
4	
5	
6	
7	
8	- 9679
9	
10	
11	
12	- 1323
13	- (1989) may not be present if scanning from previous table
14	
15	- 4344
16	
17	- 6173
18	

3. **Weiss, Exercise 6.2**

a) Build heap by insertion

```
-  10

-  10
   /
 12

-  10
   /   \                SWAP
 12  1

-    1
   /    \
  12  10

-   1
   /   \
 12  10             SWAP
 /   \
14 6

-   1
   /   \
  6   10
 /  \
14 12

-    1
   /     \
  6       10        SWAP
 /    \    /
14 12 5

-     1
    /    \
   6      5
 /   \     /
14 12 10
```

```
-      1
   /       \
  6         5
 /   \     /   \
14 12 10  8
```

```
-        1
      /        \
     6          5
   /    \    /      \
  14 12 10    8
 /
15
```

```
-        1
      /        \
     6          5
   /    \    /      \
 14 12 10     8              SWAP
 /  \
15 3
```

```
-         1
     /             \
    3               5
  /       \       /      \
  6       12  10    8          SWAP
 /   \      /
15 14 9
```

```
-            1
       /               \
       3                5
  /           \          /   \
  6           9    10     8      SWAP
 /   \       /  \
15 14 12 7
```

```
-            1
       /            \
      3              5
  /        \         /   \
  6        7    10    8        SWAP
 /   \      /  \   /
15 14 12 9 4

-             1
       /              \
      3                 4
   /        \          /   \
  6          7      5      8
 /   \      /  \    /  \
15 14 12 9 10 11

-             1
       /              \
      3                   4
   /        \          /     \
  6          7      5       8
 /   \      /  \    /  \      /
15 14 12 9 10 11 13

-             1
       /              \
      3                   4
   /        \          /     \
  6          7      5       8                SWAP
 /   \      /  \    /  \      /   \
15 14 12 9 10 11 13  2

-             1
       /              \
      3                   2
   /        \          /     \
  6          7      5       4
 /   \      /  \    /  \      /   \
15 14 12 9 10 11 13  8
```

b) Build heap with linear-time algorithm

```
-          10
      /            \
     12              1
  /      \        /    \
 14       6      5      8
/  \     / \   / \    /  \
15 3  9    7 4  11 13  2
```

```
-          10
      /            \
     12              1
  /      \        /    \
 14       6      5      2
/  \     / \   / \    /  \
15 3  9    7 4  11 13  8
```

```
-          10
      /            \
     12              1
  /      \        /    \
 14      6       4      2
/  \     / \   / \    /  \
15 3  9    7 5  11 13   8
```

Entire right subtree is now built.

```
-          10
      /            \
     12              1
  /      \        /    \
 3        6     4      2
/  \     / \   / \    /  \
15 14  9   7 5  11 13  8
```

```
-          10
      /            \
     3              1
  /      \        /    \
 12       6      4      2
/  \     / \   / \    /  \
15 14  9   7 5  11 13  8
```

Now bubble 10 all the way down by swapping with children.

```
-           1
      /              \
     3               10
  /       \         /     \
 12        6      4      2
 /   \      /  \    /  \     /   \
15 14  9   7 5  11 13  8
```

```
-           1
      /              \
     3               2
  /       \         /     \
 12        6      4      10
 /   \      /  \    /  \     /   \
15 14  9   7 5  11 13  8
```

```
-           1
      /              \
     3               2
  /       \         /     \
 12        6      4      8
 /   \      /  \    /  \     /   \
15 14  9   7 5  11 13  10
```

4. Weiss, Exercise 6.3

Heap from part a

```
                  ~~1~~
           /              \
          3               2
        /   \              /     \
     6      7           5        4
   /  \      /  \        /  \       /   \
15  14  12   9   10  11   13  8
```

```
              8
          /       \
        3           2
      /  \        /   \
    6     7      5     4
  / \    / \    / \    /
15  14  12  9  10  11  13
```

```
              2
          /       \
        3           8
      /  \        /   \
    6     7      5     4
  / \    / \    / \    /
15  14  12  9  10  11  13
```

```
              ~~2~~
          /       \
        3           4
      /  \        /   \
    6     7      5     8
  / \    / \    / \    /
15  14  12  9  10  11  13
```

```
              13
          /       \
        3           4
      /  \        /   \
    6     7      5     8
  / \    / \    / \
15  14  12  9  10  11
```

```
              3
          /        \
       **13**          4
        /  \          /    \
     6      7        5       8
   /  \     /  \     /  \
15  14  12   9   10  11
```

```
              ~~3~~
          /        \
         6              4
        /  \          /    \
     13      7       5       8
   /  \     /  \     /  \
15  14  12   9   10  11
```

```
             **11**
          /          \
         6              4
        /  \          /    \
     13      7       5       8
   /  \     /  \     /
15  14  12   9   10
```

```
             4
          /        \
         6             **11**
        /  \          /    \
     13      7       5       8
   /  \     /  \     /
15  14  12   9   10
```

```
             4
          /        \
         6              5
        /  \          /    \
     13      7      **11**      8
   /  \     /  \     /
15  14  12   9   10
```

```
                 4
             /          \
           6               5
         /   \            /    \
      13     7       10       8
     /  \     /  \       /
   15  14  12   9   11
```

5. **Weiss Exercise 6.8**
    a. By definition, the root element of a minHeap is always smaller than its children. If an element has children (i.e. is not a leaf), then it has values in the heap that are larger than it (e.g. its children at the very least). As such, the maximum needs to be an element with no children (i.e. a leaf).

    b. When N=1 (there is just a root node), there is 1 leaf (ceiling(N/1) = 1). Whenever you add a node to the heap, either the number of leaves stays the same (leaf gains a child, meaning you lose 1 leaf to becoming a normal node but gain a leaf, its child) OR the number of leaves increases by 1 (half full node becomes a full node, meaning you do not lose any leaves and gain a leaf, the new child). Because a sequence of inserts will always alternate between the two, we see that the number of leaves will always be equal to either ((N/2)+1) OR (N/2), which is the same as ceiling(N/2).

    c. A heap is only restricted by the heap property, meaning that the root must be smaller than its children. As shown in (a), this means that the maximum must be in the leaves. However, there is no other restriction other than elements being inserted from left-to-right (O(n log n) build) OR arbitrarily (O(n) build). As both of these methods of insertion are arbitrary, they can create valid permutations of the leaves in which any of them could contain the maximum and be a valid heap. Because all these permutations are possible, all leaves must be searched in order to find the maximum.