Name(s)_____ _____

Algorithms, Assignment 4: Heaps, Search Trees, and Hash Tables

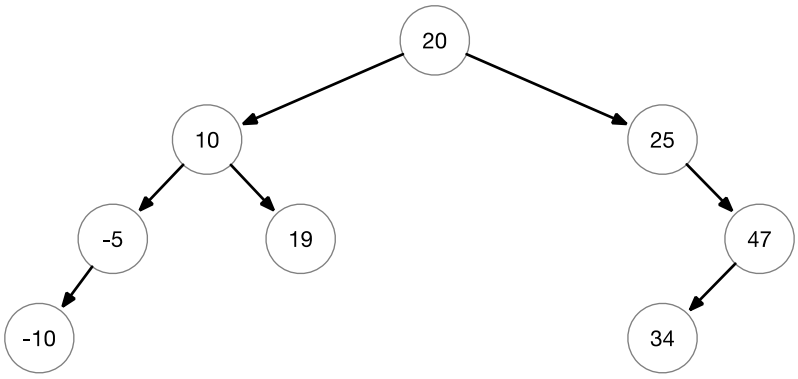
1. Answer the following questions about heaps.
   a. Why is a heap implemented using an array instead of pointers (for example, think about how linked lists and binary search trees are implemented)?

   b. Where can you find the second smallest element in a min-heap?

   c. Describe an algorithm for finding the largest element in a min-heap.

   d. What is the running time of your algorithm for finding the largest element in a min-heap?

2. Answer the following questions about trees.
   a. Turn the following binary search tree into a valid Red-Black tree.
      - Do not change any current nodes values.
      - Add as few nodes as possible.
      - Indicate black nodes by circling the node (including new nodes) and do nothing to red nodes.



   b. Why is it important to balance a binary search tree?

   c. How does a binary search tree differ from a heap?

   d. Describe an application in which you would prefer a binary search tree to a sorted array.

3. Answer the following questions.
   a. What is the probability of a collision when inserting a new object into a hash table if n=10 and the table already contains one object?

   b. What is the probability of a collision when inserting a new object into a hash table if n=10 and the table already contains four objects (do not take into account collisions from the previous insertions)?

   c. How many objects can you insert into a hash table before you have a greater than 50% chance to have a **collision between any two objects** if n=100? Assume all hash values are equally likely. Note: this problem is fundamentally different than **(a)** and **(b)**. See the assignment description for a link to an equation you can use.

   d. Fill in the following table given the hash function below and assuming that your hash table has **10 buckets** (use linear probing to handle any collisions). You should run this Python code to get your results.

```
def djb2(s):
    value = 5381 # some prime number
    magic = 33   # magic number that works well
    for c in s:
        value = value * magic + ord(c)
    return value & 0xFFFFFFFF
```

String	'Hello!,'	'CSCI140	'Algorithms'	'Class'
Hash Value				
Bucket Index				

4. Insert the following strings into the three different hash tables:

	Dijkstra	Kosaraju	Turing	Lovelace	Knuth	Backus	Neumann	Shannon	Church	Chomsky
Hash	3471236513	651802309	3562396926	442506320	227180719	2833704798	1813629751	4069740186	2881795522	602694147
Hash % 10	3	9	6	0	9	8	1	6	2	7
Hash % 17	2	5	6	9	12	9	0	9	11	15
Hash % 15	8	4	6	5	4	3	1	6	7	12

Some examples are provided for you. You should insert the objects from left (starting with 'Dijkstra') to right. (Note: these are really hash **`sets`** rather than **`dictionaries`**.) For each bucket, indicate the total number collisions experienced when inserting the item(s) in the space to the left.

I. A separate chaining hash table with 10 buckets. Insert at the **tail** of each linked list.

II. An open addressing hash table using linear probing with 17 buckets.

III. A hash table similar to that in Python 3.6+. Use an indices array with 15 buckets and use linear probing to insert into this array (see the amazing comments in dictobject.c to see collisions are actually handled—it is not too different than what we have discussed in class).

I. Separate Chaining

0	
1	
2	
3	
4	
5	
6	Turing → Shannon
7	
8	
9	

II. Open Addressing

0	
1	
2	
3	
4	
5	
6	Turing
7	
8	
9	
10	
11	Shannon
12	
13	
14	
15	
16	

III. Python 3.6 (ish)

Indices Array

0	
1	
2	
3	
4	
5	
6	2
7	
8	
9	7
10	
11	
12	
13	
14	

Entries Array

0	
1	
2	Turing
3	
4	
5	
6	
7	Shannon
8	
9	

5. In class we showed that maintaining the red-black tree invariants guarantees that the height of a red-black tree with n nodes is never more than $2\lg(n+1)$. Consider another balanced binary search tree which maintains the following invariant:

   For any node x, the heights of the left and right subtrees of x differ by at most 1.

   We'll call these "1-off" trees.

   Prove by induction that a 1-off tree with height h has at least $f(h)$ nodes, where $f(h)$ is the h^{th} Fibonacci number. Recall that

   - $f(n) = f(n-1) + f(n-2)$ and
   - $f(0) = f(1) = 1$

   When considering your base case, you will find it helpful to consider the cases when $h = 0$ and $h = 1$. Note, we discussed inductive proofs for Quicksort and Dijkstra's Algorithm.