1. **Fill in the Blank** (5000pts)
   Note: Items 1-4 deal with Red-Black properties.

   1. Every node is either red or black
   2. Color-wise, the root and leaves are black
   3. If a node is red, then its parent is black
   4. All simple paths from any node x to a descendant leaf have the same number of black nodes
   5. The pop operation is used to remove an element from a stack
   6. The element removed from a queue is the first-most element added to the queue
   7. Dijkstra's algorithm produces shortest paths between nodes.
   8. If the next empty location is found in a squared number sequence, then you are using quadratic probing.

2. **Matching** (30,000 points)

a.	Comments	(D) Accessed by a parameter to main.
b.	Block	(C) In Java, integer, floating-point, Boolean, and character.
c.	Primitive Types	(B) A sequence of statements within braces.
d.	Command-line argument	(A) Make code easier for humans to read but have no semantic meaning.
e.	Null reference	(E) The value of an object reference that does not refer to any object.
f.	Mergesort	(G) A recursive instance that can be solved without recursion.
g.	Base Case	(F) A divide-and-conquer algorithm that obtains an O(N log N) sort.
h.	Bellman–Ford algorithm	(I) An alg that is used to solve the positive-weighted, shortest-path proble
i.	Dijkstra's algorithm	(H) An alg that is used to solve the negative-weighted, shortest-path probl

3. (A Single Point) Fill in the runtimes of each sorting algorithm.

Sorting Algo	Big Oh	Theta
Selection Sort	O(n^2)	Yes
Bubble Sort	O(n^2)	Yes
Merge Sort	O(n^2)	Yes
Heap Sort	O(nlogn)	No
Quick Sort	O(nlogn)	No

4. (1600 pts) Give the expected Big-O runtime for the following operations on using the given data structures.

   Red-Black Tree: search, min, max, successor, and predecessor
   O(logn)

   Bubble Sort: Best, worst, and average case
   n, n^2, and n^2

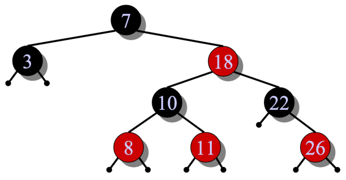
   Selection Sort: Best, worst, and average case
   All n^2

   Heapsort: Best, worst, and average case
   n (if all keys are distinct, nlogn), nlogn, nlogn

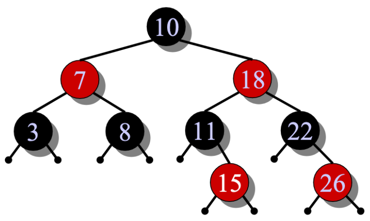
5. (12 pts) **Red-Black** Below is the adjacency matrix for a Red-Black tree. The outer edges indicate the value at a given node (the first value is the root, the next two are the node's children (left to right), and so on in an inorder traversal).
   Draw the red-black tree below, indicating black nodes as boxes and red nodes as circles.

$$\begin{bmatrix} & 7 & 3 & 18 & 10 & 22 & 8 & 11 & 26 \\ 7 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 18 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 10 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 22 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 8 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 26 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

sorry, this one was a little bit of a disaster



Now, insert a node x=15 into the tree. *Idea:* Recolor, moving violation up the tree. Right-Rotate(18), Left-Rotate(7), and recolor.

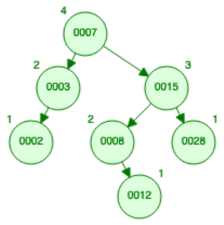


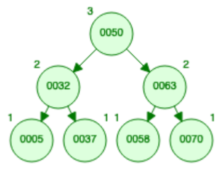
6. **Trees** (5! points)Perform a preorder traversal on this tree
   Remember: Preorder is root, left, right

```
         a
        / \
       c   g
      / \ / \
     x  m p  q
    /     \
   y       w
```

   a,c,x,y,m,g,p,w,q

7. **AVL** (4! points) Perform the AVL instructions listed below. Perform correct rotations.





8. **AVL** (6? points) Do that again. Perform correct rotations.
   Here, make 2 the parent of 1 and 14 and the left child of 32.

9. (10 or 18 points) **Heaps**

   (a) Describe the structure of a min-heap, how do each nodes compare to their children?
   The children in a min-heap will be larger than their parent.

   (b) Fill the runtimes of min-heap operations

GetMin()	O(1) : returns the root element
RemoveMin()	O(logn) : remove min element (remove the root AND maybe heapify)
Insert()	O(logn) : add a new key, may have to traverse up to fix any errors

   (c) Fill in the array index locations of each node

The Root	A[0]
Node i's Left Child	A[(2*i) +1]
Node i's Right Child	A[(2*i)+2]
Node i's Parent	A[(i-1)/2]

10. **Hash Slinging Slasher** (π points) Insert the following into a hash table using linear probing with a hash function H(x) = k mod 7. $[76, 93, 40, 47, 10, 55]$

    You should end up with this array: $[47, 55, 93, 10, -, 40, 76]$

    What is quadratic probing? An open addressing method to avoid collisions. Uses a quadratic function to find the next available space for an item.

11. **Code** (2^e points) Describe the runtime of the following psedocode functions.

```
def ILoveOwls(n):
      result = 0
      while n > 1:
            n //= 2
            result += 1
      return result
```

This runs in O(logn) time. The value of n is halved on each iteration of the loop. If n = 2^x, then logn = x.

```
def IReallyLoveOwls(n)
      result = 0
      for i in range (n):
            for j in range (i, n)
                  for k in range (n):
                        result += 1
      return result
```

O(n^3) since we run through 3 for loops, each of size n.