Name:

1) Recursion (10pts)

Consider the following iterative routine that takes a number and continually divides the number in half until it reaches zero. (32, 16, 8, 4, 2, 1)

```
i = 32
while i > 0:
      print i
      i = i // 2
```

Create a recursive function called `half_value` that behaves in a similar manner, but without any control blocks such as a `for` loop or `while` loop.

2) Prefix Notation (10pts)

a) Represent the following arithmetical expression, which is currently in typical infix form, in prefix (Polish) notation:

```
(3 + 3) / ( 8 + 4) - 10
```

b) Show your work for parsing that expression you wrote above in (a) by using a stack. Draw as many stages as necessary to describe the process of parsing and evaluating that expression.

3) JSON (10pts)

You are creating a web service that returns information about molecules to chemistry students. When someone does a query such as 'water' or 'ammonia' a response is structured that represents the information regarding that molecule. Specifically, you must include the name of the molecule and a list of constituent elements that are in the molecule such as hydrogen and oxygen. For each element listed, the quantity, atomic number, atomic mass, and the name of the element is specified.

For this example, create a JSON string that represents all the necessary information listed above for 'Ammonia'- which has 3 hydrogen atoms (atomic number 1, atomic mass: 1.007) and 1 nitrogen atom (atomic number 7, atomic mass: 14.007). There might be different ways of structuring your JSON string, but as long as all the data is structured fully and coherently you will receive credit.

4) Heaps and Priority Queues (15pts)

a) Draw the resulting max heap as a tree (not as an array) after the following operations are performed on an initially empty heap. Show your work for partial credit.

```
h = Heap()
h.insert(30)
h.insert(500)
h.insert(70)
h.insert(250)
h.remove()
h.insert(10)
h.remove()
```

b) Assume a min heap is represented as an array in memory called 'A'. Write the following insert function that will place the number in the array and 'reheapify' the values so that everything is in the correct position. (Hint: you will need to 'bubble up' the value).

```
A  = [(some numbers go here)]

def insert(value):
     # write your code here

insert(50)
```

5) Breadth-First Searching (10pts)

You are given a 2 dimensional matrix called an 'adjacency matrix'. The adjacency matrix works simply by describing if there's a connection between two nodes. If there's a one in a cell, they are connected, if a zero, they are not connected. The top heading row and the far left column in bold represents the node numbers. You will use the adjacency matrix shown below for this problem. For example, nodes 1 and 2 are connected because there's a one in the cell that links them. However, nodes 6 and 1 are not connected since there's a zero in the cell that links them.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	1	1	1	0	0	0	0	0	0
2	1	0	0	0	1	1	1	0	0	0
3	1	0	0	0	0	0	0	1	1	0
4	1	0	0	0	0	0	0	0	0	1
5	0	1	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0	0
7	0	1	0	0	0	0	0	0	0	0
8	0	0	1	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0	0
10	0	0	0	1	0	0	0	0	0	0

First, draw the graph structure with nodes and edges. Assume you start at node 1 and must travel to node 7. What is the path that's returned by a breadth first search?

6) A* Pathfinding (10pts)

a) Why is the pathfinding algorithm A* used in some cases instead of simply using a breadth-first-search?

b) Describe in your own words how the A* pathfinding algorithm works. Include as much detail as possible. (Hint, mention details about the g and the h values, and how the path is built).

7) Trees (10pts)

Consider a tree has a branching factor of 3 and a height of 4. How many total nodes are in the tree (assuming the tree is fully filled). Show your math for full credit.

8) Sorting (10pts)

a) List the time complexity of sorting a general list of numbers using the following algorithms (assume the worst case in Big O notation)? Describe simply why for each of these cases.

1) Insertion sort
2) Merge sort
3) Heap sort

b) In the case of insertion sort, what particular arrangement of data as input would make it run very quickly? Would particular arrangement of data as input would make it run very slowly? Why?

9) Abstract Data Types (15pts)

For the following scenarios, list the name of the data structure implementation that would be suitable in this particular case and why. You may name any data structure you learned in the beginning or end of the course.

a) Sally is receiving a lot of calls from customers needing jobs done. She wants to place these work orders into some data structure so that they are serviced in the order she received them. The task is eventually delegated to a worker who can service that task. What data structure should she use? Why?

b) Ben is using a queue for his breadth-first search algorithm. He decided he wants to make the algorithm instead search depth-first. What data structure would he replace the queue with? Why?

c) Taylor needs to store the a collection of telephone numbers that are retrievable by name. She wants the data structure to work quickly in most cases, often returning a telephone number without much searching at all. What data structure would he replace the queue with? Why?