Avik Bag
Professor Peysakhov
CS 260 – Assignment #3
12th July, 2015

Written Problems

Question 1

```
def fibonacci(n):
        if n < 0:
                print "Invalid Index"
        elif n == 0:
                return 1
        elif n == 1:
                return 1
        else:
                return fibonacci(n-2)+fibonacci(n-1)

def fibonacci_memo(n):
        if n < 0:
                print "Invalid Index"
        elif n == 0:
                return memo[0]
        elif n == 1:
                return memo[1]
        else:
                if memo[n] != 1:
                        return memo[n]
                else:
                        memo[n] = fibonacci_memo(n-2)+fibonacci_memo(n-1)
                        return memo[n]
```

The time complexity for fibonacci() is $O(2^n)$ because of the branching recursive function. But the time complexity for the function fibonacci_memo() is O(1) as the time required to run it is constant throughout no matter what the size of the input is. This is because the function doesn't require finding the values at the previous two indices all the time whenever the function in invoked. Every time the Fibonacci value at that index is calculated; it is stored into a global array. The last two values are obtained by accessing the last two indeces in the within the global array. This makes the run time as fast as the access time, which is O(1).

Question 2

For a function with no bounds, I would say that the size of the array could be increased to as much as the memory allows. After the maximum possible size is reached, the non-memoised function can be used. The run time until the maximum size of the array is reached will be at O(1) and any values after that would be at $O(2^n)$.

Question 3

a) Nodes: D, M, N, F, J, K, L
b) Node A
c) Node A
d) Nodes: F, G, H
e) Nodes: B, A
f) Nodes: I, M, N
g) Right sibling of D is E, F, G, H
   Right sibling of E is F, G, H
h) Nodes D, E, F is to the left of the node G and node H is to the right of node G
i) Depth of Node C is 1
j) Height of Node C is 2

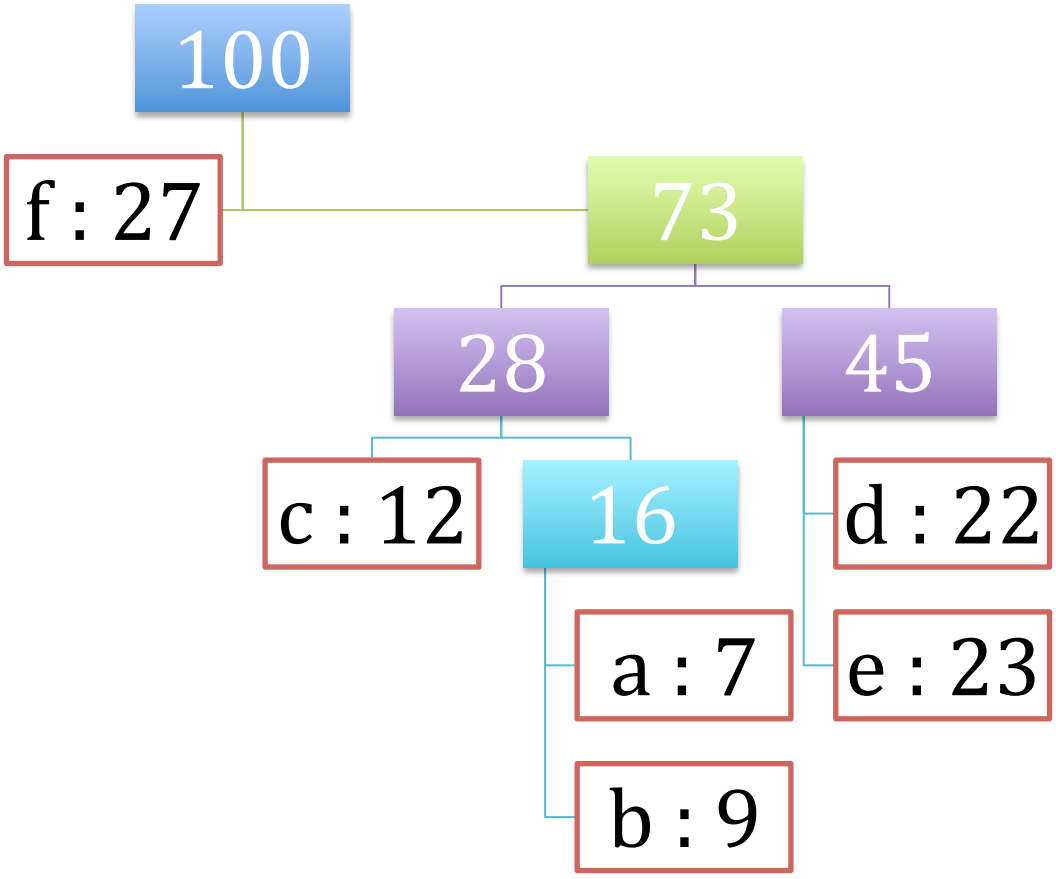
Question 4

There are **3** different paths of length 3 starting from the root Node A.

Question 5

	preorder(n) < preorder(m)	inorder(n) < inorder(m)	postorder(n) < postorder(m)
n is to the left of m	✔	✔	✔
n is to the right of m			
n is a proper ancestor of m	✔		
n is a proper descendant of m		✔	✔

Question 6



By using the Huffman code algorithm, these are the corresponding codes to the letters based on their frequency of occurrence.

Letters	A	B	C	D	E	F
Frequency	7	9	12	22	23	27
Huffman Code	1010	1011	100	110	111	0

Question 7

Greater the depth of the leaf, the lower the probability of occurrence of that character is. This is because the optimized Huffman code is designed to allow access to the leaves with a high probability in comparison to the other leaves. Thus, a leaf with a smaller depth value will require a shorter path to access the value at the leaf as compared to a leaf with a greater depth value for the leaf. Therefore, since leaf with symbol A has a greater depth than the leaf with symbol B, the probability of symbol A is lesser than that of the symbol B.