Homework 6

Q1) A binary search tree is a balanced tree when the height of the tree is O(log n). The nodes of the binary tree are evenly distributed meaning that the tree is not skewed. The left and right nodes are almost equal in number and the search, insertion and deletion operations can be done in O(log n) time.

Q2) In a perfectly balanced binary tree, the time complexity is O(log n). Here's why: Firstly, every level of the tree is populated meaning that each node has two children. Due to this balance, you eliminate half of the remaining nodes as you traverse the tree at each step. This means that the number of nodes grows logarithmically at each step.

Q3) A binary search tree can have a worst-case scenario of O(n) when the tree is severely unbalanced turning it into a linked list. For example: if we add only nodes greater than the original root node, then only right nodes will be present and traversing the tree means that it will only visit right nodes in an order meaning that the tree is basically the linked list with height 'n' where n is the number of nodes.

Q4) The recurrence relation for a binary search can be expressed as follows:

T(n) = T(n/2) + f(1)

a=1 which represents the number of subproblems created at each recursive step. When you split the binary tree problem into two subproblems, you look at the middle element hence one subproblem at this stage.

b=2 as b is the factor by which the input size is reduced at each step. So since the number of nodes are getting halved at each step, b is 2.

f(n)=f(1) because the time needed to compare the element that is being searched with the middle element is constant.

Q5)

T(n) = T(n/2) + f(1)

Using substitution:

T(n) = 2^k

1 = 1/2 + f(1) / 2^k

Re-arranging the equation.

2^k / f(1) = 2

Take log on both sides.

k = log2(2) = 1

Θ (2^k)

Θ (log n)

Q6)

$T(n) = T(n/2) + f(1)$

a=1 as only sub-problem is created at each stage

b=2 as input size is halved at each stage

d=0 as time to make comparison is constant

f(n) = 1 (constant)

n^(log_b(a)) = n^(log_2(1)) = n^0 = 1 (constant)

Using Master's Theorem Case 2:

T(n) = Θ (n^d * log^(k+1)(n))

T(n) = Θ (n^0 * log^1(n)) = Θ(log(n))