CSE 321 HOMEWORK 2

1) Solve the following recurrence relations and provide a Θ bound for each of them. You must use backward substitution, forward substitution or the Master's Theorem at least once to solve the following relations.

a) $T(n) = 3 * T(n-1) - 2 * T(n-2)$ => (forward)

$T(1) = 1 \quad T(2) = 4$

$T(3) = 3 * T(2) - 2 * T(1) = 4$

$T(4) = 3 * T(3) - 2 * T(2) = 8$

$T(5) = 3 * T(4) - 2 * T(3) = 16$

$T(n) = 2^{n-1}$ → guess

$T(n) = 2^{n-2}$

$T(n) = 2^{n-3}$ → $2^{n-1} = 3(2^{n-2}) - 2(2^{n-3})$

$T(n-1) \quad T(n-2)$

$T(n) = 3(2^{n-2}) - 2^{n-2}$

$2^{n-1} = 2^{n-2}(3-1) = 2^{n-1}$ ✓

$\in \Theta(2^n)$

b) $T(n) = T(n/2) + 1$ => (backward)

skip

1 $T(n) = T(n/2) + 1 \qquad n = 2^k \qquad k = \log_2 n$

2 $T(n) = T(n/4) + 2$

3 $T(n) = T(n/8) + 3$

$T(n/2) = T(n/4) + 1$

$T(n/4) = T(n/8) + 1$

k $T(n) = T(n/2^k) + k = T(1) + \log_2 n = \in \Theta(\log n)$

c) $T(n) = 4 * T(n-1) - 4 * T(n-2) + 3n$ => (backward)

Let's look the characteristic equation: $r^n = 4r^{n-1} - 4r^{n-2} + 3n$

(divide by r^{n-2})

$r^2 = 4r - 4 + \frac{3n}{r^{n-2}}$

$An + B = 4(A(n-1) + B) - 4(A(n-2) + B) + 3n$

$T(n) = T_h(n) + T_p(n)$

$r^2 = 4r - 4$

$r^2 - 4r + 4 = 0$ r_1 and $r_2 = 2$ $\quad T_h(n) = c_1 \cdot 2^n + c_2 \cdot n \cdot 2^n$

$T_h(n) = c_1 \cdot r_1^n + c_2 \cdot r_2^n$

$T_p(n) = An + B$

$T_p(n) = An + B$

$An + B = 4An - 4A + 4B - 4An + 8A - 4B + 3n$

$An + B = 4A + 3n \Rightarrow A = 3, \; B = 12$

$T_p(n) = An + B$

most significant growth

$T(n) = c_1 \cdot 2^n + c_2 (n \cdot 2^n) + 3n + 12 \in \Theta(n \cdot 2^n)$

d) $T(n) = 4 * T(n/2) + n^2 \quad (aT(n/b) + f(n))$ => (master's theorem)

$a = 4, \; b = 2, \; f(n) = n^2 \Rightarrow d = 2$

$\Theta(n^d \cdot \log n)$ if $a = b^d$ ✓ → $T(n) = \Theta(n^2 \log n)$

e) $T(n) = 2 * T(n/2) + O(n) \quad (aT(n/b) + f(n))$ => (master's theorem)

$a = 2, \; b = 2, \; O(n) = n \Rightarrow d = 1$

$\Theta(n^d \cdot \log n)$ if $a = b^d$ ✓ → $T(n) = \Theta(n \log n)$

f) $T(n) = T(n/2) + T_b(n/4) + n$ => (backward)

$T(n) = T_b(n/2) + T_b(n/4) + n$

$T(n/2) = T(n/4) + T(n/8) + n/2$

$T(n) = T(n/4) + T(n/8) + T(n/4) + n/2 + n$

$T(n/4) = T(n/2) + T(n/4) + n/4$

$= 2T(n/4) + T(n/8) + 3n/2$

$T(n) = 2(T(n/8) + T(n/16) + n/4) + T(n/8) + 3n/2$

$= 3T(n/8) + 2T(n/16) + 2n$

$T(n) = k \cdot T(n/2^k) + 2^{k+1} \cdot T(n/2^{k+1}) + \frac{(k+1)n}{2}$

if $n/2^{k+1} = 1 \quad k = \log_2(n/2)$

$T(n) = \log_2(n/2) T(2) + \log_2 n \, T(1) + \underbrace{\frac{n \cdot \log_2 n}{2}}_{\text{significant operation}}$

therefore $T(n) \in \Theta(n \log n)$

g) $T(n) = T(n/2) + n$ (forward) $T(1)=1$ $T(2)=3$ $\Rightarrow$ $T(n) = \Theta(2^{k+1}+1)$ → guess

$(n=2^k \cdot T(2^{k-1}) + 2^k)$

$T(2^k) = T(2^{k-1}) + 2^k$

$T(2) = T(1) + 2 = 3$

$T(4) = T(2) + 4 = 7$

$T(8) = T(4) + 8 = 15$

$T(2^k) = 2^{k+1} - 1$

$T(2^{k-1}) = 2^k - 1$

$2^{k+1} - 1 = 2^k - 1 + 2^k$

$2^{k+1} - 1 = 2^{k+1} - 1$ ✓

$2^{\log_2 n - 1} = n/2$ - $\in \Theta(n)$

$2^k = n$

$k = \log_2 n$

$\log_2 n - 1 = \log_2 n - \log_2 2$

$= \log_2 n/2$

$= \Theta(n)$

h) $T(n) = 2T(\sqrt{n}) + 1$ (master) $\Rightarrow$ Let's suppose $m = \log n$, $n = 2^m$

$T(2^m) = 2T(2^{m/2}) + 1$

Suppose: $S(m) = T(2^m)$

$S(m) = 2S(m/2) + 1$

$a=2$ $b=2$ $d=0$

$m^{\log_b a} <= a > b^d$

$T(n) = \Theta(\log n)$

$T(2^m) = \Theta(\log n)$

$m^{\log_2 2} <=$

---

② Provide pseudo code for the following operations on a given binary search tree (BST) with n nodes. Derive a recurrence relation for each of your algorithms. Calculate the average-case $\Theta()$ complexity of the derived recurrence relations.

a) is_balanced(BST): This function checks whether the given binary search tree is balanced or not

```
def is_balanced(root):
    if root is None:
        return True
    left_height = height_of_tree(root.left)
    right_height = height_of_tree(root.right)
    if (abs(left_height - right_height) <= 1) and is_balanced(root.left) is True and is_balanced(root.right) is True:
        return True
    return False
```

b) height_of_tree(BST): This function returns the height of the given binary search tree.

```
def height_of_tree(root):
    if root is None:
        return 0
    return max(height_of_tree(root.left), height_of_tree(root.right)) + 1
```

recurrence relation of height_of_tree is $T(n) = \underbrace{2T(n/2)}_{\text{left tree + right tree}} + \underbrace{\Theta(1)}_{\text{return 0}}$ $\Rightarrow$

$T(n) = 2[2T(n/4) + \Theta(1)] + \Theta(1)$

$= 2^2(T(n/2^2)) + 2\Theta(1) + \Theta(1)$

$= 2^2 \cdot (2T(n/2^3) + \Theta(1)) + 2\Theta(1) + \Theta(1)$

$= 2^3 T(n/2^3) + 2^2\Theta(1) + 2\Theta(1) + \Theta(1)$

recurrence relation of is_balanced is $T(n) = 2T(n/2) + 2\Theta(n) + 1$

$T(n) = a \cdot T(n/b) + f(n)$

by applying master's theorem: $a=2$ $b=2$ $d=1$

$\Theta(n^d \log n)$ $a = b^d$ $\Rightarrow$ $\Theta(n \log n)$

$T(n) = 2^k T(n/2^k) + 2^{k-1}\Theta(1) + 2^{k-2}\Theta(1) + \dots + \Theta(1)$

$n/2^k = 1$ $k = \log_2 n$

$T(n) = 2^{\log_2 n} T(1) + n/2\,\Theta(1) + n/4\,\Theta(1) + \dots$

$= n \cdot T(1) + n/2\,\Theta(1) + n/4\,(\Theta(1))$

$= \Theta(n)$

③ Suppose you are choosing between the following three algorithms.
a) Algorithm A divides a problem with size n into five subproblems that are one-half the size, solves each one recursively, and then combines the result in cubic time.
b) Algorithm B solves a problem with size n by recursing two subproblems of size n-2 recursively and then integrating the solutions in linear time.
c) Algorithm C addresses issues of size n by dividing them into three subproblems, each half the size, solving each subproblem recursively, and then combining the solutions in $O(n^2)$ time.
What is the running time of each algorithm (in terms of big-Oh notation), and which one would you choose? Provide a detailed explanation for your choice.

a) $T(n) = 5T(n/2) + O(n^3)$ → half size; 5 subproblems; → cubic time

by applying master's theorem $a=5 \quad b=2 \quad d=3$
$\Theta(n^d)$ if $a<b^d$ ✓ $\in O(n^3)$

b) $T(n) = 2T(n-2) + O(n)$ → n-2 size; 2 subproblems; → linear time

by applying master theorem for decreasing functions:
if: $T(n) = a \cdot T(n-b) + f(n)$
$2 \cdot T(n-2) + O(n)$
$a=2$ $b=2$ $d=1$

if $a>1 \Rightarrow \Theta(n^d \cdot a^{n/b})$
$\in \Theta(n \cdot 2^{n/2})$

i choose c algorithm: (c) $T(n) = 3T(n/2) + O(n^2)$ → half size; 3 subproblems; → quadratic time
growth rate is better than the others

by applying master's theorem $a=3 \quad b=2 \quad d=2$
$\Theta(n^d)$ if $a<b^d$ ✓ $\in O(n^2)$

---

④ The maximum cardinality matching problem in bipartite graphs involves finding the largest possible set of pairwise non-adjacent edges in a given bipartite graph. A bipartite graph is a graph in which the set of vertices can be divided into two disjoint sets, A and B, such that all edges connect a vertex from set A to a vertex in set B (and vice versa).
Provide a polynomial-time algorithm to compute a maximum cardinality matching in bipartite graphs and analyze the worst-case, best-case and average-case time complexity of the algorithm.

This problem can solve by Hopcroft-Karp algorithm. The worst-case time complexity of the algorithm is $O(E\sqrt{V})$. (E: edge, V: vertex)

1) Worst-case: $O(E\sqrt{V})$
2) Best-case: The algorithm may terminate early if the initial matching is already maximum. In this case, the time complexity would be lower but it still depends on the specific characteristics of the input graph.
3) Average-case: Depends on distribution of graphs.

⑤ Write a recurrence relation to calculate the number of characters printed when the following function is called with input n.

```
foo(n):
    if n <= 1:
        return 1
    else:
        for i in range(1):
            print("a")
        return foo(n/2) - foo(n/2)
```

recurrence relation is: $2T(n/2) + n$ → half size; 2 subproblems; → print "a" n times

by applying master's theorem: $a=2 \quad b=2 \quad d=1$
$\Theta(n^d \log n) \quad a=b^d$ ✓
$\Theta(n \log n)$