CSE 321 Homework - 2

1) a) $T(n) = 3 \cdot T(n-1) - 2 \cdot T(n-2)$

$\alpha^2 - 3\alpha + 2 = 0$ $\quad T(n) = A \cdot \alpha_1^n + B \cdot \alpha_2^n$

$\alpha \to 2 \quad -2$

$\alpha \to 1 \quad -1$ $\quad T(n) = A \cdot 2^n + B \longrightarrow \boxed{T(n) \in \Theta(2^n)}$

b) $T(n) = T(n/2) + 1$ (backwards substitution)

$T(\frac{n}{2}) = T(n/4) + 1$

$T(\frac{n}{4}) = T(n/8) + 1$

Let's say $\Rightarrow n = 2^k$

$\vdots$

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

$T(n) = T(1) + k = \underbrace{T(1)}_{\text{const}} + \log_2 n \Rightarrow \boxed{T(n) \in \Theta(\log n)}$ for all n because $\log n$ is θ-invariant.

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

$\alpha^2 - 4\alpha + 4 = 0$ $\quad T(n) = \underbrace{(A+Bn)\cdot\alpha^n}_{\text{Homogenous part}} = (A+Bn)\cdot 2^n + \underbrace{Cn+D}_{\text{Non-homogenous part}}$

$(\alpha - 2)^2 = 0$

$\alpha = 2$ $\quad \boxed{T(n) \in \Theta(n 2^n)}$

Forward substitution

g) $T(n) = T(\frac{n}{2}) + n$

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

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

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

$T(\frac{n}{2}) = T(\frac{n}{4}) + \frac{n}{2}$ $\quad \frac{n}{2^k} = 1$

$T(n) = T(\frac{n}{2}) + n$ $\quad k = \log n$

$T(n) = T(1) + 2^1 + 2^2 + 2^3 + \dots 2^k$

$2 \cdot \frac{2^k - 1}{2-1} = 2^{k+1} - 2$

$= 2^{\log n} \cdot 2 - 2$

$\boxed{T(n) = 2n - 2 \in \Theta(n)}$

h) $T(n) = 2T(\sqrt{n}) + 1$ $\quad T(n) = G(m)$

$n = 2^m$ $\quad T(2^m) = 2 \cdot T(2^{\frac{m}{2}}) + 1 = G(m)$

$G(m) = 2 \cdot G(\frac{m}{2}) + 1$

$a = 2$, $b = 2$ $\quad n^{\log_b a} = n$ $\quad f(n) = 1$ $\quad f(n) \in O(n^{1-\varepsilon})$, $\varepsilon = 1$

then $G(m) = T(n) \in \Theta(n^{\log_b a})$

$= \Theta(n)$

d) $T(n) = 4T(n/2) + n^2$ $\quad n^{\log_b a} = n^{\log_2 4} = n^2$

Master theorem $\quad a = 4$, $b = 2$ $\quad f(n) = n^2$

$f(n) \in \Theta(n^2) \to$ so, $\boxed{T(n) \in \Theta(n^{\log_b a} \log n) \in \Theta(n^2 \log n)}$

e) $T(n) = 2 \cdot T(n/2) + \Theta(n)$ (master theorem) $\quad n^{\log_b a} = n^1$

$a = 2$ $\quad b = 2$ $\quad f(n) = \Theta(n)$

if $f(n) \in \Theta(n^1)$, $T(n) \in \Theta(n \log n)$

we can say $f(n) \in \Theta(n^1)$

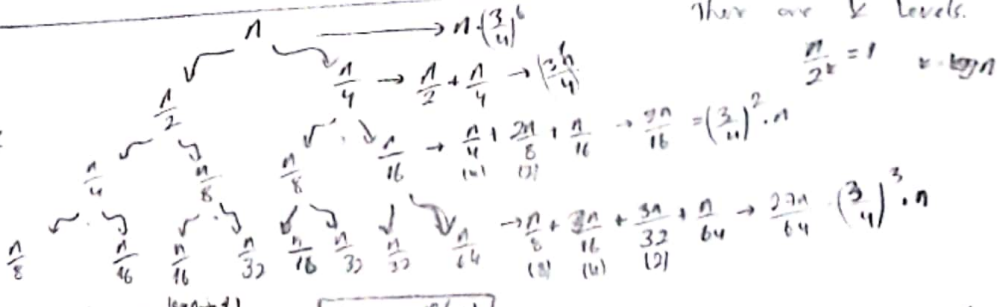
$\Theta(n) \to n$

f) $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + n$

$T(n) = n \cdot \sum_{i=0}^{k} (\frac{3}{4})^i \to 1 + \frac{3}{4} + \frac{9}{16} + \dots (\frac{3}{4})^k$

$1 \cdot \left[\frac{(\frac{3}{4})^{k+1} - 1}{\frac{3}{4} - 1}\right]$

$= 4 \cdot \left(1 - (\frac{3}{4})^{k+1}\right) \cdot n = 4 \cdot \left(1 - \frac{3}{4}^{\log n + 1}\right) \cdot n = \boxed{T(n) \in \Theta(n)}$



2) Provide a pseudo code for the following operations on a given BST with n nodes. Derive a recurrence relation for each of your algorithms. Calculate the average-case $\Theta(1)$ complexity of the derived recurrence relations.

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

```
procedure is_balanced(BST):
    if BST is None:                          } Θ(1)
        return True
    left_height = height_of_tree(BST.left)   Θ(n/2)
    right_height = height_of_tree(BST.right) Θ(n/2)
    if abs(left_height - right_height) > 1:  } Θ(1)
        return False
    return is_balanced(BST.left) and is_balanced(BST.right)
           2Θ(n/2) + Θ(1)                    2Θ(n/2) + Θ(1)
```

Master Theorem

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

$a = 2$, $b = 2$, $n^{\log_b a} = n^{\log_2 2} = n$

$f(n) \in O(n^{1-\varepsilon})$, $\varepsilon = 1$

So $T(n) \in \Theta(n)$

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

```
procedure height_of_tree(BST):
    if BST is None:                          } Θ(1)
        return 0
    left_height = height_of_tree(BST.left)   → Θ(n/2)
    left_right = height_of_tree(BST.right)   → Θ(n/2)
    return max(left_height, left_right) + 1
```

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

$a = 2$, $b = 2$, $n^{\log_b a} = n^{\log_2 2} = n$, $f(n) \in O(n^{1-\varepsilon})$, $\varepsilon = 1$

So $T(n) \in \Theta(n)$

3) a) Algorithm A: $T(n) = 5 \cdot T(n/2) + O(n^3)$

b) Algorithm B: $T(n) = 2 \cdot T(n-2) + O(n)$

c) Algorithm C: $T(n) = 3 \cdot T(n/2) + O(n^2)$

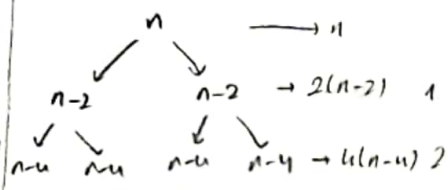
a) $a = 5$, $b = 2$, $f(n) = n^3$

$n^{\log_b a} = n^{\log_2 5} = n^{2.\ldots}$

$f(n) \in \Omega(n^{2.\ldots + \varepsilon})$

$T(n) \in \Theta(f(n)) \in O(n^3)$

b) $T(n) = 2 \cdot T(n-2) + n$



$T(n) = O(n) + 2(n-2) + 2^2 \cdot (n-2^2) + 2^3 \cdot (n-2^3) + \ldots + 2^{k-1} \cdot (n-2^{k-1})$

$a_1 \cdot \frac{r^n - 1}{r - 1} = 1 \cdot (2^k - 1)$

$n = 2^{k-1}$, $k = \log n + 1$

$T(n) = 2^k - 1 = 2^{\log n} \cdot 2 - 1 = 2n - 1 \in O(n)$

c) $a = 3$, $b = 2$, $f(n) = n^2$

$n^{\log_b a} = n^{\log_2 3} = n^{1.\ldots}$

$f(n) \in \Omega(n^{\log_2 3 + \varepsilon})$

So that $T(n) \in O(f(n)) \in O(n^2)$

- I would definitely choose the algorithm B which has the lowest growth rate which is a good example of fast algorithm.

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

$T(n)$ = number of characters printed.

```
Foo(n):
    if n ≤ 1:
        return 1
    else:
        For i in range(n):
            print("a")          } n
        return foo(n/2) + foo(n/2)
               T(n/2)     T(n/2)
```

$$T(n) = 2 \cdot T(n/2) + n \quad \text{for } n > 1$$

$$T(n) = 1 \quad \text{for } n \leq 1$$