Homework #2

Berkan AKIN
7170611073

Question #7

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

$x^2 = 3x - 2$

$x^2 - 3x + 2 = 0$

-2

-1

$(x-2)(x-1)$

$2 \quad 1$

root

$T(n) = c_1 x_1^n + c_2 x_2^n$

$T(n) = c_1(2)^n + c_2(1)^n$

$T(n) = c_1(2)^n + c_2$

$n=1 \; T(1) = 1$

$1 = 2c_1 + c_2$

$n=2 \; T(2) = c_1 \cdot 4 + c_2$

$2 = 4c_1 + c_2$

$1 = 2c_1$

$\frac{1}{2} = c_1$

$c_2 = 0$

$T(n) = \frac{1}{2} 2^n = 2^{n-1}$

$O(2^n)$

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

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

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

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

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

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

$T(4) = T(2) + 1 = 2 + 1 = 3$

$T(8) = T(4) + 1 = 3 + 1 = 4$

$T(16) = T(8) + 1 = 4 + 1 = 5$

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

$2^k = n$

$\log_2 n = k$

$T(n) = \log_2 n + 1$

$O(\log_2 n)$

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

$T(n) - 4T(n-1) + 4T(n-2) = 0$

$r^2 - 4r + 4 = 0$

$(r-2)^2 = 0 \quad r = 2$

$T(n) = (c_1 n + c_2) 2^n$

$3n = cn + n$

$T(n) = (c_1 n + c_2) 2^n + cn + n$

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

$a = 4 \quad b = 2$

$f(n) = n^2 \quad n^2 = \Theta(n^{\log_2 2})$

$a = 4 \quad b = 2$

$c = \log_b a \quad 2 = \log_2 4 \Rightarrow \Theta(n^{\log_2 4} \log n) = \Theta(n^2 \log n)$

master theorem

$\Theta(n^{\log_b a}) \quad T(n) = O(n^{\log_b a - \varepsilon})$

$\Theta(n^{\log_b a} \log^k n) \quad T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

$\Theta(f(n)) \quad T(n) = \Omega(n^{\log_b a + \varepsilon})$

e) $T(n) = 9T(\frac{n}{6}) + T(n)$

$T(n) = 2T(\frac{n}{2}) + \Theta(n) \quad a = 2 \quad b = 2$

$T(n) = \Theta(n) \quad c < 1$

$n^{\log_2 2}$

$g(n) = \Theta(n) \int n$

Case 3 for master theorem

$\Theta(n^{\log_2 2} \log n) = \Theta(n \log n)$

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

$r^2 - r - 1 = 0$ $\quad x = \frac{-1 \pm \sqrt{1+4}}{2}$

$r = \frac{1 \pm \sqrt{5}}{2}$ $\quad \frac{1+\sqrt{5}}{2}$

$T(n) = \left(\frac{1+\sqrt{5}}{2}\right)^n c_1 + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n + Cn + D$

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

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

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

$= T(1) + n/2^0 + n/2^1 + \dots + n/2^k$

$= 1 + n\left(\frac{1}{2^0} + \frac{1}{2^1} + \dots + \frac{1}{2^k}\right)$

$= 1 + n\left(\frac{1 - \left(\frac{1}{2}\right)^{k+1}}{1 - \frac{1}{2}}\right)$ using the sum of geometric series

$= 1 + n\left(1 - \frac{1}{2^k}\right)$

$= 1 + n - \frac{n}{2^{\log_2(n)}} = 1 + n - \frac{n}{n} = n$

$T(n) = T(n/2) + 1$ $\quad n = 2^k$ $\quad T(1) = 1$ $\quad Q(n)$

h) $T(n) = 2T(n/2) + n$ Find a recursion relation

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

$S(k) = 2S(k-1) + 2^k$

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

$= 4S(k-2) + 2^k + 2^k$

$= 4(2S(k-3) + 2^{k-2}) + 2^k + 2^k$

$= 8S(k-3) + 2^k + 2^k + 2^k$

$= 2^3 S(k-3) + 3 \cdot 2^k$ after i steps

$k = 2^i \quad T(1) = 1 \quad i = \log_2(k)$

h) $S(k) = 2^{\log_2(k)} S(1) + k - 1 = 2k - 1$

$S(k) = T(2^k)$ $\quad k = \log_2 n$ $\quad$ Transform

$T(n) = 2\log_2(n) - 1$

$T(n) = 2\log_2 n - 1$ $\quad\quad \Theta(\log_2 n)$

D) is_balanced(node)
if node is null:
return true, 0

left_balanced, left_height = is_balanced(node.left) T(2/n)
right_balanced, right_height = is_balanced(node.right) T(2/n)
balanced = abs(left_height - right_height) <= 1 O(1)
height = max(left_height, right_height) + 1 O(1)
return balanced and left_balanced and right_balanced, height

$T(n) = 2T(n/2) + O(1)$ we use master Theorem

$O(1) < n^{\log_2 2}$ => we use case 1. of master theorem

$O(n^{\log_2 2}) = O(n)$

1)
```
height_of_tree(node)
  if node is null:
      return 0

  left_height = height_of_tree(node.left)    T(n/2)
  right_height = height_of_tree(node.right)  T(n/2)

  height = max(left_height, right_height)+1   O(1)
  return height
```

$T(n) = 2T(n/2) + O(1)$ we use master theorem

$a = 2 < O(2^{\log_2 2})$ $\Rightarrow$ $O(n^{\log_2 2})$ $\Rightarrow$ $O(n^{1-\log_2 2}) = O(n)$

3.a) $T(n) = 5T(n/2) + O(n^3)$ Algorithm recursions Relation exists

we use master theorem

$T(n)$ $O(n^{\log_2 5})$ O

$n^3 > O(n^{\log_2 5})$ we select 3 case so

$O(T(n)) = O(n^3)$

4) $T(n) = 2T(n-2) + O(1)$ $T(0) = 1$ $n = 2k$

$T(2k) = 2T(2k-2) + O(1)$

$T(2k-2) = 2T(2k-4) + O(1)$

$T(2k-4) = 2T(2k-6) + O(1)$

$T(2k-6) = 2T(2k-8) + O(1)$

$T(0) = 1$

$T(2k) = 2(2(2(2T(2k-8) + O(1)) + O(1)) + O(1)) + O(1)$

$= 2^2 T(2k-4) + 3O(1)$

$= 2^3 T(2k-6) + 2^2 O(1) + 3O(1)$

$= 2^4 T(2k-8) + 2^3 O(1) + 2^2 O(1) + 3O(1)$

$= 2^k T(0) + 2^{k-1} O(1) + 2^{k-2} O(1) + \dots + 2^2 O(1) + 3O(1)$

$= 2^k T(0) + 2^{k-1} O(1) + 2^{k-2} O(1) + \dots + 2^2 O(2^{k-2}) + 3O(2^k)$

$= 2^k + \sum_{i=1}^{k} 2^{k-i} O(2^i) + 3O(2^k)$

$= 2^k + O(2^k) \cdot (2^k - 1) + 3O(2^k)$

$= 2^k + 2^k O(2^k) - O(2^k) + 3O(2^k)$

$= 2^k (1 + O(2^k)) + 2O(2^k)$

$= O(2^k) + O(2^k)$

$= O(2^k)$ $\Rightarrow$ $T(n) = O(2^k)$ $\Rightarrow$ $O(2^{\frac{n}{2}})$ $n = 2^k$

c) $T(n) = 3T(\frac{n}{2}) + O(n^2)$

we can use master theorem.

$T(n) = O(n^2)$ case 3.

$T(n) \geq O(n \log n)$ $\Rightarrow$ $O(T(n)) = O(n^2)$

Summary

$O(n) = O(n^3)$

$G(n) = O(2^{\frac{n}{2}})$

$C(n) = O(n^2)$

I select $C(n)$ algorithm because more efficient than other algorithm.

4) There are several algorithms for the maximum cardinality matching problem. Some of these algorithms have worst-case, best-case and average-case time complexity.

These algorithms is below

Simulation: Attempts to find a matching using augmenting paths. It has a worst-case time complexity of $O(V*E)$, where V is the number of nodes and E is the number of edges. This algorithm can be inefficient, especially for large graphs.

Edmond-Karp Algorithm (BFS-based augmenting path): This algorithm finds augmenting paths using breath first search (BFS) It has a worst-case time complexity of $O(V*E^2)$. In the best case the complexity can be better. For example, when there are few matches or few augmenting paths in the graph)

Hopcroft-Karp Algorithm: This algorithm has a better time complexity of $O(E*\sqrt{V})$ This algorithm is often more effective for larger graphs.

Blare Algorithm: This algorithm combines a worst-case time complexity of $O(V^2*E)$ but it often performs better in real applications. In the best case it can have a complexity of $O(E*V*\log(V))$

* I select to Hopcroft-Karp Algorithm. Because better than other. Pseudo code is below

```
Function HopcroftKarp(Graph G)
    For each vertex v in G.leftNodes do
     Pair[v] = NULL
    MaxMatching = 0
    while(BFS()) do
        For each vertex v in G.leftNodes do
            If (Pair[v] == NULL and DFS(v)) then
              maxMatching = maxMatching + 1
    return maxMatching
```

$O(E*V)$ = $O(E)$ → $O(E)$

5) Foo(n)

```
if n == 7
    return 7
else
    for i in range(n):
        print("0")
    return Foo(n/2) + Foo(n/2)
```

$T(n/2) \quad + T(n/2)$

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

we use master theorem

$a = 2$

$b = 2$

$T(n) = n \qquad n^{\log_2 2}$

$n \in \Theta(T(n)) \qquad$ case 2

$\Theta(n^{\log_2 2} \cdot \log n) = \Theta(n \cdot \log n)$