Ahmet Tuğkan Ayhan
1901042692

CSE321 - Homework 2

1)
$$T(n) = \begin{cases} \Theta(n^{\log_b a}) \rightarrow \text{case 1}, & f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta(n^{\log_b a} . \log n) \rightarrow \text{case 2}, & f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) \rightarrow \text{case 3}, & f(n) = \Omega(n^{\log_b a + \varepsilon}) \\ & \text{control}: a.f(n/b) < c.f(n) \end{cases} \quad \begin{matrix} \varepsilon > 0 \\ c < 1 \end{matrix}$$

---

a) $T(n) = 16T(\frac{n}{4}) + n!$

* $a : 16$
  $b : 4$
  $f(n) : n!$

* $n^{\log_4 16} \Leftrightarrow n!$

* $n^2 < n!$ (case 3)

* We need to test $a.f(n/b) < c.f(n)$

* $16.(\frac{n}{4})! < c.n!$, $c = \frac{1}{2}$

* for $n=8$, $c=\frac{1}{2} \rightarrow 16 \cdot 4! < \frac{1}{2} \cdot 8! \rightarrow 768 < 40,320$, so it satisfies for sufficiently large n values

* Case 3: $T(n) = \Theta(f(n)) \Rightarrow T(n) = \Theta(n!)$

---

b) $T(n) = \sqrt{2}\, T(\frac{n}{4}) + \log n$

* $a : \sqrt{2}$
  $b : 4$
  $f(n) : \log n$

* $n^{\log_4 \sqrt{2}} \Leftrightarrow \log n$

* $n^{\frac{1}{4}} > \log n$ (case 1)

* Case 1: $T(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\frac{1}{4}})$

---

c) $T(n) = 8T(\frac{n}{2}) + 4n^3$

* $a : 8$
  $b : 2$
  $f(n) : 4n^3$

* $n^{\log_2 8} \Leftrightarrow \cancel{4}n^3$

* $n^3 = n^3$ (case 2)

* Case 2: $T(n) = \Theta(n^{\log_b a} \log n) \Rightarrow T(n) = \Theta(n^3 . \log n)$

d) $T(n) = 64T(\frac{n}{8}) - n^2 \log n$

* $a : 64$
  $b : 8$
  $f(n) : -n^2 \log n$

★ This algorithm cannot be solved by using moster theorem. Becouse f(n), which is the combination time, is not positive. Cost of an algorithm cannot be negative.

---

e) $T(n) = 3T(\frac{n}{3}) + \sqrt{n}$

* $a : 3$
  $b : 3$
  $f(n) : \sqrt{n}$

* $n^{\log_3 3} \Leftrightarrow \sqrt{n}$

* $n > \sqrt{n}$ (case 1)

* Case 1: $T(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n)$

---

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

* $a : 2^n$
  $b : 2$
  $f(n) : -n^n$

★ This algorithm cannot be solved by using moster theorem. Becouse, f(n), which is the combination time, is negative and "a" is not a constant.

---

g) $T(n) = 3T(\frac{n}{3}) + \frac{n}{\log n}$

* $a : 3$
  $b : 3$
  $f(n) : n / \log n$

★ This algorithm cannot be solved by using moster theorem becouse the difference between f(n) and $n^{\log_b a}$ must be polynomial time but this is not the case for this algorithm.

2) $T(n) = a.T(\frac{n}{b}) + f(n)$

a : Number of subproblems
b : Subproblem size
f(n) : cost / time

---

a) Algorithm X $\Rightarrow T_x(n) = 9T(\frac{n}{3}) + n^2$

b) Algorith Y $\Rightarrow T_y(n) = 8T(\frac{n}{2}) + n^3$

c) Algorithm Z $\Rightarrow T_z(n) = 2T(\frac{n}{4}) + \sqrt{n}$

Which to choose?

★ I would choose Algorithm Z because this one has the lowest running time complexity, Also it divides problem into less amount of subproblems and this is also good for the memory. Finally cost of these subproblems are lower than other algorithms

Running Times

a) $T_x(n) = 9T(\frac{n}{3}) + n^2$

* a : 9
  b : 3
  f(n) : n^2

* $n^{\log_3 9} \Leftrightarrow n^2$

* $n^2 = n^2$ (case 2)

* Case 2 : $T_x(n) = O(n^{\log_b a} . \log n) \Rightarrow T_x(n) = O(n^2 \log n)$

b) $T_y(n) = 8T(\frac{n}{2}) + n^3$

* a : 8
  b : 2
  f(n) : n^3

* $n^{\log_2 8} \Leftrightarrow n^3$

* $n^3 = n^3$ (case 2)

* Case 2 : $T_y(n) = O(n^{\log_b a} . \log n) \Rightarrow T_y(n) = O(n^3 . \log n)$

c) $T_z(n) = 2T(\frac{n}{4}) + \sqrt{n}$

* a : 2
  b : 4
  f(n) : $\sqrt{n}$

* $n^{\log_4 2} \Leftrightarrow \sqrt{n}$

* $\sqrt{n} = \sqrt{n}$ (case 2)

* Case 3 : $T_z(n) = O(n^{\log_b a} . \log n) \Rightarrow T_z(n) = O(n^{\frac{1}{2}} . \log n)$

......../......../........

3) Array = [1,2,3,4,5,6,7,8]

a)

i) Merge Sort Max. Comparison : [5,1,7,3,6,2,8,4] 17 comparison

ii) Merge Sort Min. Comparison : [1,2,3,4,5,6,7,8] 12 comparison

b)

pivot

i) Quick Sort Max. Swap : [8,7,6,5,4,3,2,1]

→pivot

ii) Quick Sort Min. Swap : [2,3,4,5,6,7,8,1]

Explanation

a)

→17 comparison

i) To make maximum amount of comparison first we need to make worst scenario for first divide which is [1,3,5,7] and [2,4,6,8]. If we do like that every element in the subarrays will be compared. Then we do the same for rest of it. [1,5], [3,7] and [2,6], [4,8]. Finally we swap the last subarrays and merge them back. [5,1], [7,3] and [6,2], [8,4] → [5,1,7,3] and [6,2,8,4] → [5,1,7,3,6,2,8,4]

ii) To make minimum amount of comparison we need to do exact opposite what we did in first case, First we divide into two sorted arrays [1,2,3,4] and [5,6,7,8]. We do this until we get [1], [2], [3], [4], [5], [6], [7], [8]. We merge them back [1,2], [3,4], [5,6] [7,8] → 4 comparison, [1,2,3,4] and [5,6,7,8] → 4 comparison. Finally [1,2,3,4,5,6,7,8] → 4 comparison. Total → 12

b)

i) Max swap : 35

ii) Min swap : 7

4)

```
algorithm (left, right)                      T(n)
    mid = (left + right) / 2                 Θ(1)
    if A[mid] == 0                           Θ(1)
        return mid
    else
        if A[mid] > 0                        Θ(1)
            right = mid                      Θ(1)
            algorithm (left, right)          T(n/2)
        else
            left = mid                       Θ(1)
            algorithm (left, right)          T(n/2)
```

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

Recurrence Relation : $T(n) = T(\frac{n}{2}) + 1$

* $a : 1$ * $n^{\log_2 1} \Leftrightarrow 1$

$b : 2$

$f(n) : 1$ * $1 = 1$ (case 2)

* Case 2 : $T(n) = \Theta(n^{\log_b a} \cdot \log n) \Rightarrow T(n) = \Theta(\log n)$

5)

a) algorithm (box , gift) $T(n)$

```
algorithm ( box , gift)                                   T(n)
    giftSize = length(gift) ........................... Θ(1)

    if giftSize == 1 .................................. Θ(1)
        return box

    for i=1 to giftSize ............................... Θ(n)
        if box[i] == gift[0] .......................... Θ(1)
            swap (box[i], box[0]) ..................... Θ(1)

    left = algorithm (box[0:1], gift[0,1]) ............ T(n/2)
    right = algorithm (box[1:giftSize], gift[1:giftSize]) ... T(n/2)

    box = left + right ................................ Θ(1)

    return box
```

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

* $a: 2$
  $b: 2$
  $f(n): n$

* $n^{\log_2 2} \Leftrightarrow n$

* $n = n$ (case 2)

* Case 2 : $T(n) = \Theta(n^{\log_b a} . \log n)$

* $\boxed{T(n) = \Theta(n . \log n)}$