Gebze Technical University

Computer Engineering

CSE - 321

Introduction to Algorithm Design

Homework - 2

Yunus ÇEVİK

141044080

Course Teacher: Didem GÖZÜPEK

Course Asistance: Mehmet Burak KOCA

Recursive Equation Solutions

- Substitution Method (Yerine Koyma)
- Iteration Method (Iterasyon Yöntemi)
- Master Theorem (Ana Teorem)

Generic Equation Formula

$$T(n) = \begin{cases} c & n=1 \\ aT\left(\frac{n}{b}\right) + cn & n>1 \end{cases}$$

Result

$$T(n) = \begin{cases} \Theta(n) & a<b \\ \Theta(n \log n) & a=b \\ \Theta(n^{\log_b a}) & a>b \end{cases}$$

Master Theorem

* Divide and Conquer

Formula

Assume that when $T(n) = aT(n/b) + f(n)$, $a >= 1$, $b > 1$ and $f(n)$ asymtotic

$$T(n) = \begin{cases} \text{Case1} \quad \Theta(n^{\log_b a}) & f(n) = O(n^{\log_b a - \varepsilon}) \\ \text{Case2} \quad \Theta(n^{\log_b a} \log n) & f(n) = \Theta(n^{\log_b a}) \\ \text{Case3} \quad \Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon} \log n) \text{ and } af(n/b) < cf(n) \cdot \text{bign} \end{cases} \quad \varepsilon > 0,\ c < 1$$

Regularity Condition

1) Suppose you are choosing between the following three algorithms:

a) Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.

b) Algorithm B solves problems of size n by recursively solving two subproblems of size n-1 and then combining the solutions in constant time

c) Algorithm C solves problems of size n by dividing them into nine subproblems of size n/3, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running time of each of these algorithms (in big-O notation), and which would you choose?

Answer-1

a) $T(n) = 5T(n/2) + O(n)$

$a=5,\ b=2,\ d=1$

Recursive Equation Solution
Master Theorem Case1

$d < \log_b a \Rightarrow 1 < \log_2 5 \Rightarrow 1 < 2.32 \quad \boxed{d \cong 2.32}$

$= O(n^{\log_b a})$
$= O(n^{\log_2 5})$

$O(n^d) \Rightarrow O(n^{2.32}) \Rightarrow \underline{O(n^2)}$ Big-Oh quadratic time. $\cong O(n^2)$

b) $T(n) = 2T(n-1) + O(1)$

$T(n) = 2*2\,T(n-2) + O(1)$

$T(n) = 2*2*2\,T(n-3) + O(1)$

$\vdots$

$\boxed{k=n-1}$ $\quad T(n) = 2^k\,T(n-k) + O(1)$

$T(n) = 2^{n-1}\,T(n-(n-1)) + O(1)$

constant constant

$T(n) = 2^{n-1}\,T(1) + \cancel{O(1)}$

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

Big-Oh exponantial time

c) $T(n) = 9T(n/3) + O(n^2)$

$a=9,\ b=3,\ d=2$

$d = \log_b a \Rightarrow 2 = \log_3 9 \Rightarrow 2 = \log_3 3^2 \Rightarrow 2 = 2.\cancel{\log_3 3}^{1}$

$f(n) = n^{\log_b a}$

Recursive Equation Solutions
Master Theorem

Case2

$2 = 2$

$\Theta(n^{\log_b a}\log n) = \Theta(n^{\log_3 9}\log n)$

$\underline{\Theta(n^2 \log n)}$

2) Solve the following recurrence relations and give a Θ bound for each of them.

a) $T(n) = 2T(n/3) + 1$

b) $T(n) = 5T(n/4) + n$

c) $T(n) = 7T(n/7) + n$

d) $T(n) = 9T(n/3) + n^2$

e) $T(n) = 8T(n/2) + n^3$

f) $T(n) = 49T(n/25) + n^{3/2} \log n$

g) $T(n) = T(n-1) + 2$

h) $T(n) = T(\sqrt{n}) + 1$

Answer 2

a) $T(n) = 2T(n/3) + 1$

$a = 2,\ b = 3,\ f(n) = 1 = n^0$

$n^{\log_b a} \Rightarrow n^{\log_3 2}$ $\qquad f(n) < n^{\log_3 2}$

$n^0 < n^{\log_3 2}$

Recursive Equation Solutions
Master Theorem
Case 1

$\Theta(n^{\log_b a}) = \Theta(n^{\log_3 2})$

b) $T(n) = 5T(n/4) + n$

$a = 5,\ b = 4,\ f(n) = n$

$n^{\log_b a} \Rightarrow n^{\log_4 5}$ $\qquad f(n) < n^{\log_4 5}$

$n^1 < n^{\log_4 5}$

Recursive Equation Solutions
Master Theorem Case 1

$\Theta(n^{\log_b a}) = \Theta(n^{\log_4 5})$

c) $T(n) = 7T(n/7) + n$

$a = 7,\ b = 7,\ f(n) = n$

$n^{\log_b a} \Rightarrow n^{\log_7 7} = n$ $\qquad f(n) = n^{\log_b a}$

$n = n$

Recursive Equation Solutions
Master Theorem
Case 2

$\Theta(n^{\log_b a} \log n) = \Theta(n^{\log_7 7} \log n)$

$\Theta(n \log n)$

Answer2

d) $T(n) = 9T(n/3) + n^2$

$a = 9,\ b = 3,\ f(n) = n^2$

$n^{\log_b a} \Rightarrow n^{\log_3 9} = n^{2\log_3 3} = n^2 \quad f(n) = n^{\log_b a}$

$n^2 = n^2 \quad \Theta(n^{\log_b a}\log n) = \Theta(n^{\log_3 9}\log n)$

$\underline{\Theta(n^2 \log n)}$

Recursive Equation Solutions
Master Theorem
Case 2

e) $T(n) = 8T(n/2) + n^3$

$a = 8,\ b = 2,\ f(n) = n^3$

$n^{\log_b a} \Rightarrow n^{\log_2 8} = n^{3\log_2 2} = n^3 \quad f(n) = n^{\log_b a}$

$n^3 = n^3 \quad \Theta(n^{\log_b a}\log n) = \Theta(n^{\log_2 8}\log n)$

$\underline{\Theta(n^3 \log n)}$

Recursive Equation Solutions
Master Theorem
Case 2

f) $T(n) = 49T(n/25) + n^{3/2}\log n$

$a = 49,\ b = 25,\ f(n) = n^{3/2}\log n$

$n^{\log_b a} = n^{\log_{25} 49} = n^{\log_{5^2} 7^2} = n^{2/2 \log_5 7} = n^{\log_5 7}$

$f(n) > n^{\log_b a} \Rightarrow n^{3/2}\log n > n^{\log_5 7}$

Recursive Equation Solutions
Master Theorem
Case 3

$af(n/b) < c f(n)$

$49\dfrac{n^{3/2}\log n}{25} < c\, n^{3/2}\log n \qquad \dfrac{49}{25} < c$

$\underline{\Theta(f(n)) = \Theta(n^{3/2}\log n)}$

g) $T(n) = T(n-1) + 2 \qquad T(1) = 1$

$T(n) = T(n-2) + 2+2$

$T(n) = T(n-3) + 2+2+2$

$\vdots$

$\boxed{k = n-1} \quad T(n) = T(n-k) + 2k$

Substituting Equations $n \to n-1$

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

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

$T(n) = 2n - 1$

$\underline{T(n) = O(n)}$

h) $T(n) = T(\sqrt{n}) + 1$

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

$T(n) = T(n^{1/4}) + 1+1$

$T(n) = T(n^{1/8}) + 1+1+1$

$\vdots$

$\boxed{n^{1/2^k} = 2} \quad T(n) = T(n^{1/2^k}) + 1k$

$n^{1/2^k} = 2$

$\log n^{1/2^k} = \log_2 2 \Rightarrow 1/2^k \log n = 1$

$\log n = 2^k$

$\log(\log n) = \log_2 2^k \Rightarrow \boxed{\log(\log n) = k}$

not important

$T(n) = \cancel{T(2)} + \log(\log n)$

$\underline{T(n) = O(\log(\log n))}$

3) Given an array of n elements, and you notice that some of the elements are dublicates; that is, they appear more than once in the array. Show how to remove all dublicates from the array in time O(n log n).

Answer 3

Öncelikle dizinin elemanlarını O(n logn) complexity algoritmasına uygun bir yapıda sıralamalıyız. Bu yapıya uygun olan "Merge Sort" sılama algoritmasını kullanabiliriz. Sıralama işleminin ardından, sıralanmış dizi elemanları gezinilerek (traversin...) yinelenen (dublicate) elemanlar kaldırılmalıdır. Yinelenen elemanların kaldırıldığı dizi ekrana çıktı olarak verilebilir veya yeni bir diziye kopyalanılarak başka işlemlerde kullanılmak üzere hazırlanabilir.

```
function mergesort (variable a as array)
    if( n equal to 1)
        return a
    variable M as array = a[0] ... a[n/2]
    variable N as array = a[n/2+1] ... a[n]
    M asign mergesort(M)
    N asign mergesort(N)
    return merge(M,N)
end function

function merge(variable a as array, variable b as array)
    variable c as array
    while ( a and b have elements)
        if( a[0] > b[0])
            add b[0] to the end of c
            remove b[0] from b
        else
            add a[0] to the end of c
            remove a[0] from a
    while (a has elements)
        add a[0] to the end of c
        remove a[0] from a
    while (b has elements)
        add b[0] to the end of c
        remove b[0] from b
    return c
end function
```

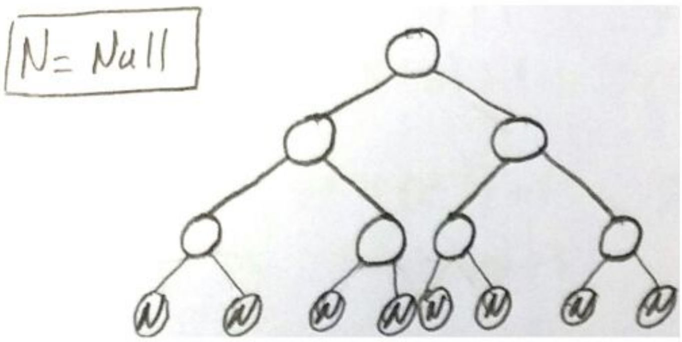
```
function remove dublicate (variable a as array)
    variable b as array, variable c as array
    b asign mergesort(a), k asign 0
    for(i asign 0; i from b legnth-1; i increase)
        if(b[i] equal to b[i+1])
            c[k] add b[i]
            remove b[i+1] from b
            k increase
        else
            c[k] add b[i]
            remove b[i] from b
    return c
end function
```

Yukarıda sıralanmış bir diziden aynı olan elemanların silinerek yeni bir diziye atılmasının pseudo kodudur.

4) Consider the task of searching a sorted array $A[1, \ldots, n]$ for a given element x: a task we usually perform by binary search in time $O(\log n)$. Show that any algorithm that accesses the array only via comparisons (that is, by asking questions of the form "is $A[i] \leq z$?"), must take $\Omega(\log n)$ steps.

Answer 4

Binary Search algoritmasının alt sınırını ispatlamak için, ikili arama ağacının her iç düğümü bir karşılaştırmadır. Her bir yaprağın (leaf) bir permütasyon olmasındansa, alt sınırın sınıflandırılmasında olduğu gibi her bir yaprak dizinin bir elemanıdır. Dizinin her elemanı en az bir yaprak olmalı çünkü bu algoritma bir yaprak tarafından temsil edilmeyen bir yerde bir eleman bulamaz. Ayrıca dizide n eleman olması için ağacın en az n yaprak içermesi gerekir. Böylece, ağacın derinliği $\Omega(\log n)$'dir. Bu nedenle bu tür bir algoritma $\Omega(\log n)$ karşılaştırması almalıdır.



Örneğin: n=7 elemanlı olsun
A ağacını A dizisi olarak ele alırsak $A[1 \ldots 7]$ toplam 7 yaprak (leaf) vardır.

$A[i] \leq z$ yapısın sorulara cevap verecek bir yapıdadır.

5) How many lines, as a function of n (in $\Theta(\cdot)$ form), does the following program print? Write a recurrence and solve it. You may assume n is a power of 2.

```
function f(n)
  if n>1
    print_line("still going")   (Θ(1))
    f(n/2)                      (T(n/2))
    f(n/2)                      (T(n/2))
```

Answer 5

$T(n) = 2 * T(n/2) + 1$ $\quad$ $n = 2^k$ $\quad$ $k=4$

$\boxed{n=16}$

$T(1) = 0$

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

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

$T(8) = 2 * T(4) + 1 = 7$

$T(16) = 2 * T(8) + 1 = 15$ $\quad$ $\boxed{n-1 = 16-1 = 15}$

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

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

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

$\boxed{n=2^k}$ $\quad = 2^k - 1$

$= n - 1$

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