Gebze Technical University
Computer Engineering

CSE 222-2018 Spring

HOMEWORK II REPORT

Yunus ÇEVİK
141044080

Course Asistant: Fatma Nur Esirci

1) Prove using only the definitions of asymptotic notations.

a) $n2^n$ is in $O(2^{2n})$

b) $n!$ is in $\Omega(2^n)$

c) $(\log n)$ is in $\Theta(\log_{64} n)$

Answer 1

a Big-O notasyonu için

$T(N) = O(f(N))$ Eğer pozitif sabitler varsa c ve n_0 öyleki

$T(N) \leq c\, f(N) \quad N \geq n_0$

$n2^n \leq c \cdot 2^{2n} \Rightarrow n \cdot 2^n \leq c \cdot 2^n \cdot 2^n$

$n \leq c \cdot 2^n \quad c \geq 1 \quad n_0 \geq 1 \quad \underline{\forall N \geq n_0}$

$1 \leq 1.2^1 \Rightarrow 1 \leq 2$ ✓ $c=1$ $n_0=1$

$2 \leq 2.2^2 \Rightarrow 2 \leq 8$ ✓ $c=2$ $n_0=2$

b) Omega Notasyonu için

$T(N) = \Omega(f(N))$ Eğer pozitif sabitler varsa c ve n_0 öyleki

$T(N) \geq c\, f(N) \quad N \geq n_0$

$n! \geq c.2^n \quad c \geq 1 \quad n_0 \geq 4$

$4! \geq 1.2^4 \quad c=1, n_0=4$

$24 \geq 1.16$ ✓

$5! \geq 2.2^5 \quad c=2, n_0=5$

$120 \geq 64$ ✓

c) Big O notasyonu için

$\log n \leq c_1 \cdot \log_{64} n$

$\log_2 n \leq c_1 \log_{(2^6)} n$

$\log_2 n \leq c_1 \cdot \frac{1}{6} \cdot \log_2 n$

$1 \leq c_1 \cdot \frac{1}{6}$

$c_1 \geq 6 \quad n_0 \geq 1$

Omega notasyonu için

$\log n \geq c_2 \cdot \log_{64} n$

$\log_2 n \geq c_2 \log_{(2^6)} n$

$\log_2 n \geq c_2 \frac{1}{6} \cdot \log_2 n$

$1 \geq c_2 \frac{1}{6}$

$0 < c \leq 6 \quad n_0 \geq 1$

$\boxed{c_1 = 6 = c_2}$

$\underline{c=6}$

$\Theta(\log_{64} n)$ için $c=6$ $n_0 \geq 1$ olması gerekir.

2) Sort the following functions from slowest to fastest in terms of their growth. Using limit estimation.

$(\lg n)^{\lg n}$, $\lg(n!)$, n, $\lg(\lg(n))$, $\lg(2^n)$, $\lg(\lg(\sqrt{n}))$

Answer 2

→ $(\lg n)^{\lg n} \stackrel{?}{=} O(\lg(n!))$

$$\lim_{n\to\infty} \frac{(\lg n)^{\lg n}}{n} \underset{\text{L'hospital}}{=} \frac{\frac{1}{\ln 2}\cdot\frac{1}{n^{\log_2(n)}}\cdot\frac{2n^{\log_2(n)-1}\cdot\ln(n)}{\ln 2}}{1} = \frac{\frac{2\ln(n)}{\ln^2(2)n}}{1} \underset{\text{L'hospital}}{=}$$

$$\Rightarrow \lim_{n\to\infty} \frac{\frac{2\ln(n)}{\ln^2(2)n}}{1} = 0, \text{ böylece } n > (\lg n)^{\lg n}$$

→ $n \stackrel{?}{=} O(\lg(2^n)) \Rightarrow n = \log_2 2^n \Rightarrow n = n\cdot\log_2 2 \Rightarrow n = n$ böylece $n = \lg(2^n)$

→ $(\lg n)^{\lg n} \stackrel{?}{=} O(\lg(\lg(n)))$

$$\lim_{n\to\infty} \frac{(\lg n)^{\lg n}}{\lg(\lg(n))} \underset{\text{L'hospital}}{=} \frac{\frac{2\ln(n)}{\ln^2(2)n}}{\frac{1}{\ln(2)}\cdot\frac{1}{\log_2(n)}\cdot\frac{1}{n\cdot\ln(2)}} = \frac{\frac{2\ln(n)}{\ln^2(2)n}}{\frac{1}{\ln^2(2)n\log_2(n)}} \underset{\text{L'hospital}}{=}$$

$$\Rightarrow \lim_{n\to\infty} \frac{2\ln(n)}{\ln^2(2)n}\cdot\ln^2(2)n\log_2(n) \Rightarrow \lim_{n\to\infty} 2\ln(n)\cdot\log_2(n) = \infty$$ böylece $(\lg n)^{\lg n} > \lg(\lg(n))$

→ $\lg(\lg(n)) \stackrel{?}{=} O(\lg(\lg(\sqrt{n})))$

$$\lim_{n\to\infty} \frac{\lg(\lg(n))}{\lg(\lg(\sqrt{n}))} \underset{\text{L'hospital}}{=} \frac{\frac{1}{\ln(2)}\cdot\frac{1}{\log_2(n)}\cdot\frac{1}{n\cdot\ln(2)}}{\frac{1}{\ln(2)}\cdot\frac{1}{\log_2(\sqrt{n})}\cdot\frac{1}{2\ln(2)n}} \Rightarrow \lim_{n\to\infty} \frac{2\log_2(\sqrt{n})}{\log_2(n)} = 1$$

böylece $\lg(\lg(n)) = \lg(\lg(\sqrt{n}))$

→ $\lg(n!) \stackrel{?}{=} O(n)$

$$\lim_{n\to\infty} \frac{\lg(n!)}{n} \to \log_2(n\cdot n-1\cdot n-2\cdot n-3\cdot \dots \cdot 2\cdot 1) = \log_2(n)+\log_2(n-1)+\dots+\log_2 2+\log_2 1$$

$$= \log_2(n)+\log_2(n)+\dots+\log_2(n) = n\cdot\log(n)$$

$$\log_2(n!) = n\cdot\log_2(n)$$

$$\lim_{n\to\infty} \frac{n\cdot\log_2(n)}{n} \Rightarrow \lim_{n\to\infty} \log_2(n) = \infty$$ böylece $\lg(n!) > n$

Sonuç olarak $\Rightarrow$ $\lg(n!) > n = \lg(2^n) > (\lg n)^{\lg n} > \lg(\lg(n)) = \lg(\lg(\sqrt{n}))$

Yavaştan hızlıya doğru sıralarsak $\lg(\lg(n)) = \lg(\lg(\sqrt{n})) < (\lg n)^{\lg n} < \lg(2^n) = n < \lg(n!)$

3) Solve the following recurrence relation using the substitution method.

$T(n) = 2T(n-1) + 1 \quad n>1, \quad T(n)=1 \quad n=1$

Answer3

$T(1) = 1$ ← Base case

$T(n) = 2T(n-1) + 1$ ← Recursive case

$k=1$ için $T(n) = 2T(n-1) + 1$

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

$k=2$ için $T(n) = 2[2T(n-2)+1]+1 = 2^2T(n-2)+2+1$

- $T(n-2) = 2T(n-2-1)+1 = \underline{2T(n-3)+1}$

$k=3$ için $T(n) = 2^2[2T(n-3)+1]+2+1 = 2^3T(n-3)+4+2+1$

⋮

$k=k$ için $T(n) = 2^kT(n-k) + \sum_{i=0}^{k-1} 2^i$

Note: $\sum_{i=m}^{n} a^i = \frac{a^n - a^m}{1-a}$

$2^kT(n-k) + \frac{2^0-2^k}{1-2} \Rightarrow 2^kT(n-k) + \frac{1-2^k}{-1} \Rightarrow 2^kT(n-k) + (2^k-1)$

$T(n-k-1)$

$n-k-1=0$

$n-1=k$

$\Rightarrow 2^{n-1}T(n-(n-1)) + (2^{n-1}-1) \Rightarrow 2^{n-1}.T(1) + (2^{n-1}-1)$

$\Rightarrow 2^{n-1} + 2^{n-1} - 1 \Rightarrow 2^{n-1}(1+1) - 1 \Rightarrow 2^{n-1}.2^1 - 1$

$\Rightarrow 2^{n-1+1} - 1 \Rightarrow \underline{2^n - 1}$

Sonuç olarak $O(2^n)$ Exponential'dir.

$\Omega(2^n)$

4) Explain the running time of f(n) using recurrence relation.

```
f(n):
if(n == 1)
    return 1
else
    return f(n-1) + f(n-1)
```

Yandaki recursive fonksiyonda yapılması kesin olan 4 işlem vardır.

1⇒ Base Case de yapılan karşılaştırma

2⇒ Recursive Case de yapılan toplama işlemi

3 ve 4 ise f(n-1) işleminde çıkarma işleminin iki kere yapılmasıdır.

1 + 1 + 1 = 3 + 1 = 4

Answer 4

$f(1) = 1$ ← Base Case

$f(n) = 2f(n-1)+4$ ← Recursive Case

$k=1$ için $f(n) = 2f(n-1) + 4$

$\cdot f(n-1) = 2f(n-1-1)+4 = \underline{2f(n-2)+4}$

$k=2$ için $f(n) = 2[2f(n-2)+4]+4 = 2^2 f(n-2) + 8 + 4$

$\cdot f(n-2) = 2f(n-2-1)+4 = \underline{2f(n-3)+4}$

$k=3$ için $f(n) = 2^2[2f(n-3)+4] + 8 + 4 = 2^3 f(n-3) + 16 + 8 + 4$

$\vdots$

$k=k$ için $f(n) = 2^k f(n-k) + 4 \cdot \sum_{i=0}^{k-1} 2^i$

Note : $\sum_{i=m}^{n} a^i = \frac{a^m - a^n}{1-a}$

$2^k f(n-k) + 4 \cdot \left(\frac{2^0 - 2^k}{1-2}\right) \Rightarrow 2^k f(n-k) + 4 \cdot \left(\frac{1-2^k}{-1}\right) \Rightarrow 2^k f(n-k) + 4(2^k - 1)$

T(n-k-1)

n-k-1 = 0

n-1 = k

$\Rightarrow 2^{n-1} f(n-(n-1)) + 4(2^{n-1} - 1) \Rightarrow 2^{n-1} f(1) + 4(2^{n-1} - 1)$

$\Rightarrow 2^{n-1} + 4 \cdot 2^{n-1} - 4 \Rightarrow 2^{n-1}(1+4) - 4 = 2^{n-1}(5) - 4$

$\Rightarrow \underline{5 \cdot 2^{n-1} - 4}$

Sonuç olarak $O(2^n)$ Exponential'dir.

$\Omega(2^n)$

5) What does do UnknownFunction? Analize the running time of UnknownFunction using proper asymptotic notations.

```
UnknownFunction(n):
i = 0                          ——————> Q(1)
while (n %2 == 0)              ┐
    n = n/2                    │  Döngü verilen n değerinin asal çarpanlarından
                               │  2'nin derecesi kadar dönmektedir. Her döngüye
    i++                        │  girdiğinde n değeri logaritmik olarak azaldığından
                               ┘  bu while döngüsü Q(log n)'dir.
return i                       ——————> Q(1)
```

Answer 5)

UnknownFunction fonksiyonu, verilen sayının asal çarpanları arasındaki 2'nin k. dereceden kuvvetlerini belirler. 2^k'li değer, iki tabanına sahip logaritma ile ifade edilebilir ve fonksiyon, çalışma döngüsünü k kez tekrarlar. Girilen değer döngünün şartına bağlı olarak çift sayı olup olmadığını belirleyip, eğer çift sayı ise her k. iterasyonda $\log_2 2^k$ gibi bir logaritmik azalışa uğrar. Sonuç olarak verilen n sayısının asal çarpanlarından 2'nin derecesini return eden bir fonksiyondur.

Fonksiyon $T(n) = Q(1) + Q(\log n) + Q(1)$

Fonksiyon Best Case => $T_B(n) = \Omega(1)$

Fonksiyon Worst Case => $T_W(n) = O(\log n)$

6) Write Insertion sort with pseudocode Explain analyze of your algorithm worst-case, best-case and average-case using proper asymptotic notations.

Answer 6)

Insertion Sort - Pseudocode

Input => n elemanlı A[1..n] arrayi

Ouput => A[1..n] azalmayan sıraya göre sıralanır.

```
1. for i ← 2 to n                          ┐
2.     x ← A[i]                            │
3.     j ← i-1                             │
4.     while (j>0) and (A[j]>x)            │ maximum ((n-1) x (n-1))/2
5.         A[j+1] ← A[j]                   │
6.         j ← j-1                         │
7.     end while                           ┘
8.     A[j-1] ← x                          n-1
9. end for                                 ┘
```

Best-Case

- A arrayi sıralanmış bir şekilde ise sadece n-1 kere for döngüsü döner.
- İçerde bulunan while döngüsü çalışmayacağından hesaplanmaz.

$T_b(n) = O(n)$ linear

Worst-Case

- A arrayi ters sıralandığında worst-case durumu meydana gelir. İçerde bulunan while döngüsü n-1 defa karşılaştırma yapmalıdır.

$$T_w(n) = (n-1) + \dots + 2 + 1$$
$$= (n(n-1))/2$$
$$= O(n^2) \quad \text{Quadratic}$$

Avarage-Case

- İçerde bulunan while döngüsü yaklaşık olarak (n-1)/2 karşılaştırma yapmalıdır.

$$T_{avg}(n) = (n-1)/2 + \dots + 2/2 + 1/2$$
$$= n(n-1)/4$$
$$= O(n^2) \quad \text{Quadratic}$$

7) Calculate the running time of the Test function. Show your calculation in paper asymptotic notations.

```
Test(n):
for(int i=1; i<2n; i+=4i)
  for(int j=n; j>0; j--)
    if(i*j == target)
      target = checkFunc(n); → Q(nlogn)
    else
      print(); → Q(1)
```

Q(n) Q(logn)

i+=4i den for döngüsü $5^0, 5^1, 5^2, 5^3, 5^4$ şeklinde 1, 5, 25, 125, 625 arttığından bu döngü $\log_5 n$ sürede yapar. Ben bunun $\log_2 n$ olarak kabul ettim

```
checkFunc(n){
foo(n); → Q(n)
if(n==1) → Q(1)
  return 1;
else
  return checkFunc(n/2);
}
```

Q(nlogn)

```
foo(n){
for(int i=0; i<n; i++)
  print(); → Q(1)
}
```

Q(n) Qn Q(n)

Test(n) ⇒ worst-case $Q(n^2 \log n^2)$

best-case $Q(n \log n)$

* Assume that arithmetic operators and print() can be done in constant time

Answer 7

foo(n) ⇒ foo fonksiyonu for döngüsünde n kadar n defa döneceğinden worst-case de $Q(n)$ yani linear time, n=1 durumunda best-case de $Q(1)$ yani constant time çalışır.

checkFunc(n) ⇒ Bu fonksiyon içinde bulunan foo(n) fonksiyonu Q(n) zamanda olursa ve base-case $T(1)=1$, recursive-case $T(n) = 1 + T(n/2)$ olursa checkFunc(n) şu şekilde hesaplanır.

$n = 2^k$

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

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

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

$\vdots$

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

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

$2^k = n$

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

$T_w(n) = n \log n + \log n + 1 \in Q(n \log n)$ $T_b = Q(1)$

Test(n) ⇒ Test fonksiyonunda target değerini ilk olarak 1 kabul edersek Test(n) fonksiyonunun worst-case

$T_w(n) = \log_2 n \cdot n \cdot n \log_2 n = (n \log n)^2$ $T_w(n) = Q(n^2 \log n^2)$

best-case ⇒ $T_b(n) = 1 \cdot n \cdot \log n = n \log n$ $T_b(n) = Q(n \log n)$