

Gebze Technical University
Computer Engineering

CSE 222
2017 Spring

HOMEWORK 02 REPORT

Elif Şeyma ARMAĞAN
151044042

Course Assistant:
Nur Banu ALBAYRAK

Q1:

Give running times of each of the algorithms in proper notations. Explain your answers.

1.

```
for (int i = 0; i < n - 1; i++) {  
    for (int j = i + 1; j < n; j++) {  
        3 simple statements  
    }  
}
```

ANSWER 1.1

Answer
Dıştaki döngü 1 kere döndüğünde içteki döngü $3(n-1)$ ifade çalıştırır.
2. iterasyon için $3(n-2)$ ifade çalışır.
1
en sonda 3 tane ifade çalışır.

$$\begin{aligned} T(n) &= 3(n-1) + 3(n-2) + 3(n-3) + \dots + 3 \\ \text{verilen bir } n \text{ için} \quad &= 3(n-1 + n-2 + n-3 + \dots + 1) \\ &= 3 \left(\frac{n \cdot (n-1)}{2} \right) = 1,5n^2 - 1,5n \end{aligned}$$

$$1,5n^2 - 1,5n \leq cn^2 \quad n > n_0 \quad (\text{Bu ifadeyi doğrulamak } c \text{ ve } n_0 \text{ bulabilirsek } T(n) = O(n^2) \text{ diyebiliriz.})$$

Eğer n değerini 1 girersek denkleminiz 0 olur. 1'den büyük seçeceğimiz tüm değer için ise c 'yi 1,5 alırsanız durumunda oluşan $1,5n^2 - 1,5n \leq 1,5n^2$ eşitliği sağlanmış olur. Bu yüzden c 'yi 1,5 ve n_0 'ı 1 seçebiliriz. Böylelikle $T(n) = O(n^2)$ ifadesi sağlanmış olur.

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

2.

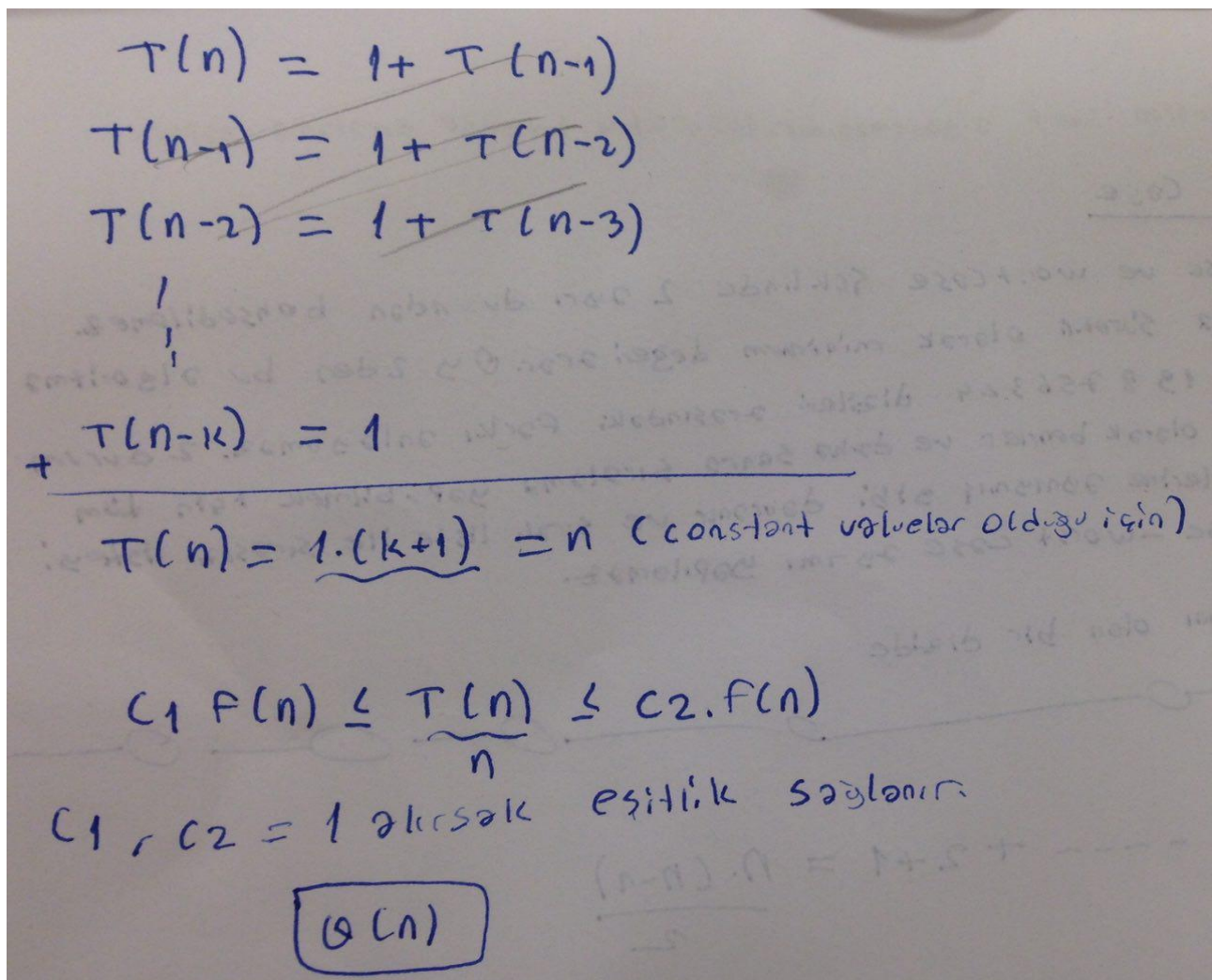
```
public static int length(String str) {  
    if (str == null || str.equals(""))  
        return 0;  
    else  
        return 1 + length(str.substring(1));  
}
```

ANSWER 1.2

Best Case

- Verilen stringin boş olması veya null olması durumudur. Bu durumda if statement yalnızca 1 kontrol yapar.

$$T(n) = \theta(1) = \text{constant time}$$


$$\begin{aligned} T(n) &= 1 + T(n-1) \\ T(n-1) &= 1 + T(n-2) \\ T(n-2) &= 1 + T(n-3) \\ &\vdots \\ + T(n-k) &= 1 \end{aligned}$$

$$T(n) = 1 \cdot (k+1) = n \quad (\text{constant value } k \text{ için})$$
$$C_1 f(n) \leq \underbrace{T(n)}_n \leq C_2 f(n)$$

$C_1, C_2 = 1$ alırsak eşitlik sağlanır.

$\theta(n)$

Q2:

```
SOME_FUNCTION (A)
n ← length[A]
for j ← 1 to n - 1
  do smallest ← j
  for i ← j + 1 to n
    do if A[i] < A[smallest]
      then smallest ← i
  exchange A[j] ↔ A[smallest]
```

1. What does the function do?
2. Give the best-case and worst-case running times of the algorithm in Θ notation. Explain your answer.

ANSWER 2.1

Verilen algoritma Selection Sort algoritmasıdır. Verilen n elemanlı bir sayı dizisini küçükten büyüğe sıralar.

ANSWER 2.2

Selection sort için best case ve worst case şeklinde 2 ayrı durumdan bahsedilemez. Çünkü algoritma her iterasyon boyunca sürekli olarak o dizideki minimum değeri arar. Bu yüzden bu algoritma çalışırken 12345678 ile 18675423 dizileri arasındaki farkı anlayamaz. 2 durum için de öncelikle 1 i en küçük eleman olarak belirler ve daha sonra sıralama yapabilmek için tüm diziyi dolaşır. Bütün veri kümelerine aynıymış gibi davranır ve sıralı bir dizi ile sırasız diziyi ayırt edemez. Bu yüzden best case – worst case ayrımı yapılamaz.

Selection sort n tane elemanı olan bir dizide ilk en küçük elemanı bulabilmek için (n-1) kere döner. 2. en küçük elemanı bulabilmek için (n-2) tane karşılaştırma yapar. Bu döngü 3. için (n-3) diye tekrarlar.

$$(n-1) + (n-2) + (n-3) + \dots + 2 + 1 = \frac{n \cdot (n-1)}{2}$$

$$0,5n^2 - 0,5n \leq cn^2 \quad n > n_0 \quad (c \text{ ve } n_0 \text{ bulabilirsek})$$

$$T(n) = O(n^2) \text{ diyebiliriz.}$$

eğer n_0 değerini 2 alırsak ve c değeri için 0,5 seçersek eşitlik sağlanmış olur.

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

Q3:

Prove that the running time of an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$.

ANSWER 3

Algoritmanın çalışma süresinin $T(n)$ olduğunu varsayalım. Eğer $T(n) = \Theta(g(n))$ ise

$$0 \leq c_1 \cdot g(n) \leq T(n) \leq c_2 \cdot g(n) \quad n \geq n_0 \text{ için}$$

$0 \leq T(n) \leq c_2 \cdot g(n) \quad n \geq n_0$ için eşitliği üstteki eşitlikten elde edilir.

$T(n)$ $\Theta(n)$ 'in üst sınıridir. Worst case $\Theta(n)$ dir.

$$0 \leq c_1 \cdot g(n) \leq T(n) \quad n \geq n_0 \text{ için} \quad T(n) = \Omega(g(n))$$

$T(n)$ $\Omega(n)$ 'in alt sınıridir. Best case $\Omega(n)$ dir.

Q4:

1. Express insertion sort as a recursive procedure.
2. Write a recurrence for the worst-case running time of the procedure. Explain your answer.
3. Solve the recurrence. Give detailed answer.

ANSWER 4.1

1 den n e kadar olan sayıları içeren bir diziyi $[1 \dots n]$ recursive olarak sıralamak için önce $n-1$ lik kısmının recursive olarak sıralandığını varsayınız. Daha sonra geriye kalan n . sıradaki o elemanı zaten recursive olarak sıralanmış dizide bir yere yerleştirmeye çalışırız. Kod ise aşağıdaki gibidir.

```
public static int[] RecursiveInsertionSort(int[] array, int n)
{
    int i;

    if (n == 1)
        return array;

    if (n > 1)
        RecursiveInsertionSort(array, n - 1);

    else
    {
        int k = array[n];
        i = n - 1;
        while (i >= 0 && array[i] > k)
        {
            array[i + 1] = array[i];
            i = i - 1;
        }
        array[i + 1] = k;
    }
    return array;
}
```


ANSWER 4.2

$$T(n) = \begin{cases} O(1) & \text{if } n == 1 \\ T(n-1) + O(n) & \text{if } n > 1 \end{cases}$$

$T(n-1)$ = Recursive olarak $(n-1)$ tane elemanı sıralama işlemi

$O(n)$ = Recursive olarak sıralanmış arrayde son kalan elemmanı uygun bir yere koymak için geçen süre

Best Case

- Verilen dizide yalnızca 1 eleman olması durumudur. Bu durumda dizimiz zaten sıralıdır. İlk if statement ına girer ve constant time da çalışır.

Worst Case

- Verilen dizinin ters yönde sıralı olması durumudur.
- İlk başta fonksiyon $(n-1)$ için tekrar çağrılır.
- İçte dönen while döngüsü her n için $(n-1)$ kez döner.

$T(n) = T(n-1) + n$ bizim recurrence ımız olur.

ANSWER 4.3

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= n + (n-1) + T(n-2) \\ &= n + (n-1) + (n-2) + T(n-3) \\ &\vdots \\ &= n + (n-1) + \dots + T(1) \\ &= \frac{n \cdot (n-1)}{2} \\ &= O(n^2) \\ T(n) &= O(n^2) = O(n^2) \end{aligned}$$

Q5:

Indicate giving detailed explanation whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$ or $f(n) = \Theta(g(n))$ for the following:

1. $f(n) = n^{0.1}$, $g(n) = (\log n)^{10}$
2. $f(n) = n!$, $g(n) = 2^n$
3. $f(n) = (\log n)^{\log n}$, $g(n) = 2^{(\log_2 n)^2}$

ANSWER 5.1

$f = \log(n^{0.1}) = \frac{1}{10} \cdot \log n$ $g = \log((\log n)^{10}) = 10 \cdot \log \log n$

g fonksiyonu f 'e göre daha yavaş olduğu için alttan sınırlar.

Böylelikle $(\log n)^{10} \leq n^{0.1}$ $n \geq 1$ denebilir.

$f(n) = \Omega(g(n))$ dir.

ANSWER 5.2

Deris kitabımızda verilen kıyaslama tablosu şu şekildedir.

$1 < \log n < n < n \log n < n^2 < n^3 < 2^n < n!$ (en karmaşık (en kötü) (yavaştan hızlıya))

Bu tabloya göre $n!$, 2^n 'e göre daha hızlı büyür

$f(n) = n!$, $g(n) = 2^n$

$g(n)$ Fonksiyonu $f(n)$ 'e göre daha yavaş büyüdüğü için alttan sınırlar.

$2^n < n!$ $n \geq 4$ için

$f(n) = \Omega(g(n))$ dir.

ANSWER 5.3

$F(n) = (\log n)^{\log n}$ şu şekilde yazılabilir $n^{\log \log n}$

$g(n) = (2^{\log_2 n})^{\log_2 n} = n^{\log_2 n}$ şeklinde yazılabilir.

$F(n)$ Fonksiyonu $g(n)$ Fonksiyonundan daha büyüktür (üsleri karşılaştırsak.)

$g(n)$ Fonksiyonu $F(n)$ Fonksiyonunu üstten sınırlar.

$F(n) \leq g(n) \quad n > 1$

$F(n) = O(g(n))$ dir.

Q6:

Rearrange your Homework 1 code using **Array**, **Array List**, and **Linked List** structures. Analyze and compare their performances. Add detailed information about your test and performance analysis method. Also, write detailed analysis results in your report.

```
"C:\Program Files\Java\jdk1.8.0_102\  
Testing LinkedList with 15000 input.  
  
ms: 3210  
second: 3  
  
Process finished with exit code 0
```

```
"C:\Program Files\Java\jdk1.8.0_102\bin\java" ...  
Testing ArrayList with 15000 input.  
  
ms: 560  
second: 0  
  
Process finished with exit code 0
```

```
"C:\Program Files\Java\jdk1.8.0_102\bin\java" ...  
Testing MyArray with 15000 input.  
  
ms: 2410  
second: 2  
  
Process finished with exit code 0
```

Burada veri yapılarını test etmek için Java'nın Random kütüphanesini kullanarak rastgele isimler ve ID'ler oluşturup 15000 uzunluğunda bir liste oluşturdum. Her bir veri yapısı için sonuçlar ekran görüntülerinde görünmektedir.

LinkedList çok daha uzun sürmüştür çünkü bu işlemler süresince elemanlara erişim çok gereklidir. LinkedList bir elemana ulaşmak için bütün listenin üstünden geçmek zorunda olduğu için böyle kötü çalışmaktadır.

ArrayList ise elemanlara direkt eriştiği için çok hızlı çalışmaktadır.

Benim yazdığım MyArray ise LinkedList'den iyi ama ArrayList kadar iyi değil çünkü LinkedList'in yaptığı gibi liste üstünde dolaşmadan elemana ulaşma imkanına sahip ancak ArrayList kadar optimize çalışmıyor.

Elemanlara direkt erişimin gerektiği uygulamalarda ArrayList gibi direkt erişim sağlayan veri yapıları programı çok hızlandırabilir.