Veri Yapıları HW2

Büşra ARSLAN
131044021

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
    }
}
```

Solution: inner loop $\sum_{j=i+1}^{n} c$ $\quad 1+\cdots+(n-2)+(n-1)$ şeklindedir.

(n-1) executed time

3 simple statements olduğu için;

$3.(1+\cdots+(n-2)+(n-1))$

outer loop executed time : n

$$T(n)=\sum_{i=0}^{n-1}\sum_{j=i+1}^{n}3=\sum_{i=0}^{n-1}3(n-i)$$

$$=3\sum_{i=0}^{n-1}(n-i)$$

$$=3\left(\sum_{i=0}^{n-1}n-\sum_{i=0}^{n-1}i\right)$$

$$=3\left((n-1).n-\frac{(n-1).n}{2}\right)$$

$$=3\left(n^2-n-\frac{(n^2-n)}{2}\right)$$

$$T(n)=3\left(\frac{n^2-n}{2}\right)$$

$$T(n)\in O(n^2)$$

Q1) 2)

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

$\underbrace{\text{return}}_{1}\ \underbrace{1+}_{1}\text{length(str.}\underbrace{\text{substring(1)}}_{1}\text{));}$ — 3 statement

recursive çöz

$$T(n) = \cancel{T(n-1)} + 3$$
$$\cancel{T(n-1)} = \cancel{T(n-2)} + 3$$
$$\cancel{T(n-2)} = \cancel{T(n-3)} + 3$$
$$\cancel{T(n-3)} = T(n-4) + 3$$
$$\vdots \qquad \vdots$$
$$\cancel{T(n-k)} = T(n-k-1) + 3$$

$$T(n) = T(n-k-1) + 3(k+1) \qquad n-k-1 = 1$$
$$k = n-2$$
$$T(n) = T(1) + 3(n-2+1)$$
$$T(1) = 1$$
$$T(n) = 1 + 3(n-1)$$
$$T(n) = 3n-2 = \Theta(n)$$

n arttıkça sürekli yavaşlamaz.

⇒ T(n)'i length'in tamamı olarak düşündüm.
Her substring'de birer eleman olarak ilerler.

| 1 | 5 | 8 | 3 | 4 |

Son elemana gelene kadar bu şekilde ilerler.
Bunu da T(n-1) . . . T(n-k-1) olarak ifade ettim.

Q2) a) What does the function do?
b) Give the best case and worst case running times of the algorithm in Θ notation. Explain your answer.

a)
```
for(int j=0; j<n-1; ++j){
    smallest=j;                      ] 1 statement
    for(int i=j+1; i<n; ++i){
        if( A[i] < A[smallest] )
            smallest=i;              ] 1 statement
    }
    A[j]=A[smallest];                ] 1 statement
}
```

Selection sort algoritması kolay bir algoritmadır. Baştan sıralamayı yaparak ilerler.

Bu kod parçacığı diziyi küçükten büyüğe doğru sıralar.

Bu sıralama selection sort sıralamasıdır, Verimsizdir($O(n^2)$) içteki döngüde en küçük sayı bulunmuş, dıştaki döngüde ise bu işlemin her seferinde yenilenmesi sağlanır. Sıralama yöntemi yalın olduğu ve bazı durumlarda kameli sıralama daha karmaşık olan algoritmalara göre daha iyi sonuç verir.

b) Best Case ve Worst Case durumları incelendiğinde;

Best Case — if durumuna girmezse. Yani dizi sıralıysa.

$$\sum_{j=0}^{n-1}\sum_{i=j+1}^{n} 2 = \sum_{j=0}^{n-1} 2(n-j) = 2\sum_{j=0}^{n-1}(n-j)$$

$$= 2\left(\sum_{j=0}^{n-1} n - \sum_{j=0}^{n-1} j\right)$$

$$= 2(n.(n-1)) - \cancel{2}\left(\frac{(n-1).n}{\cancel{2}}\right)$$

$$= 2n^2 - 2n - n^2 + n$$

Best case $O(n^2) \leftarrow T(n) = n^2 - n$

Bu durumda selection sort sıralamamızın base case ve worst case'i $O(n^2)$ bulunur.

Worst Case — if durumuna girerse. Yani dizi sıralı değilse.

$$\sum_{j=1}^{n-1}\sum_{i=j+1}^{n} 3 = \sum_{j=1}^{n-1} 3(n-j) = 3\left(\sum_{j=1}^{n-1} n - \sum_{j=1}^{n-1} j\right)$$

$$T(n) = 3(n^2-n) - 3\frac{(n^2-n)}{2} = \frac{3(n^2-n)}{2}$$

worst case $\leftarrow O(n^2)$

Q3) If $T(n) = \Theta(g(n))$, $0 \le c_1 g(n) \le T(n) \le c_2 \cdot g(n)$; $n \geq n_0$

- $T(n) = O(g(n)) \Rightarrow 0 \le T(n) \le c_2 \cdot g(n)$ for $n \geq n_0$

  $T(n)$'in üst sınırı $O(n)$'dir.

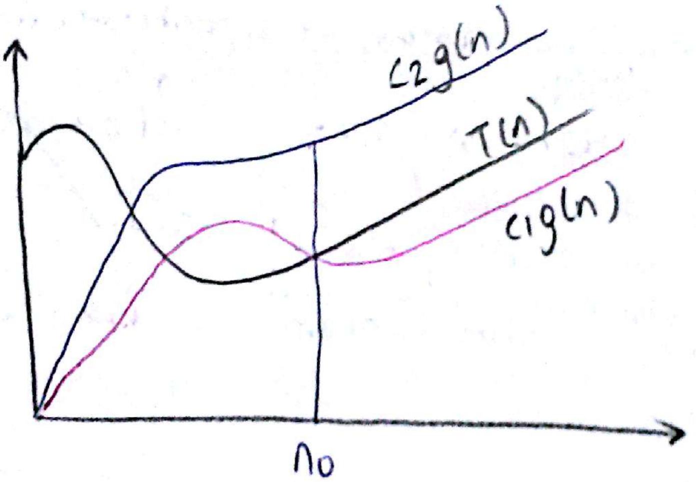
  Worst case running time of the algorithm is $O(n)$.

  $O(n)$ worst case durumunu ifade eder çalışma süresinde.

- $T(n) = \Omega(g(n)) \Rightarrow 0 \le c_1 g(n) \le T(n)$ for $n \geq n_0$

  $T(n)$'in alt sınırı $\Omega(n)$'dir.

  Best case running time of the algorithm is $\Omega(n)$.



θ-notation

Lower Bound $\le$ Running Time $\le$ Upper Bound

$T(n) = O(g(n))$ and $T(n) = \Omega(g(n))$

$\Rightarrow 0 \le T(n) \le c_2 g(n)$, $n >= n_1$

$0 \le c_1 g(n) \le T(n)$, $n >= n_2$

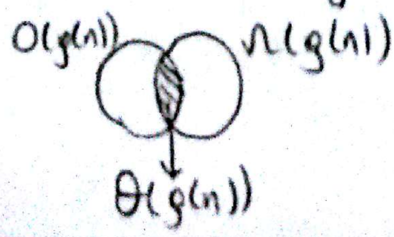
$\Rightarrow n_0 = \max(n_1, n_2)$ bunun sonucunda;

$0 \le T(n) \le c_2 \cdot g(n)$ $n \geq n_0$ and

$0 \le c_1 g(n) \le T(n)$ $n \geq n_0$

$\Rightarrow 0 \le c_1 g(n) \le T(n) \le c_2 g(n)$ $n \geq n_0$

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



Bu denklemde göre worst case $O(g(n))$ ve best case $\Omega(g(n))$ olan bir fonksiyonun çalışma zamanı $\Theta(g(n))$'dir. //

Ç4) 1) Bu soruda www.algoqueue.com sitesinden yararlanılmıştır.

```
public static int[] insertionSort (int[] array, int m)
{
        int i=0;
        if(m-1) {
T(n-1) <---- insertionSort (array, m-1);
        }
        else {
            int k = array[m];
            i=m-1;
            while (i>=0 && array[i]>k) {
                array[i+1] = array[i];
                i=i-1;
            }
            array [i+1]=k;
        }
        return array;
}
```

2) $T(n) \begin{cases} m=1 \rightarrow O(1) \\ m>1 \rightarrow \underbrace{T(n-1)}_{\text{recursive kol için}} + O(n) \end{cases}$ Best Case

karşılaştırmalar için

Sıralarken sıralamada uygun konumu bulmak için $O(n)$ süresi.

$$T(n) = T(n-1) + n$$
$$= T(n-2) + n + (n-1)$$
$$= T(n-3) + n + (n-1) + (n-2)$$
$$\vdots$$
$$= T(1) + n + (n-1) + (n-2) + \dots + 2$$
$$= \frac{n(n-1)}{2} + \theta(1) = \theta(n^2) \Rightarrow T(n) = O(n^2)$$

Best Case $\parallel O(n)$

Worst Case

3) Insertion algoritmasında, eleman sayısı az olursa avantajlı bir algoritma olur. Verimsiz bir algoritmadır. Uygulaması kolay bir algoritmadır.

Worst Case : $O(n^2)$
Base Case $O(n)$

Ö4) 3) Insertion Search Algoritma Mantığı

Example 9 5 4 8 1

1) 5 9 4 8 1

2) 5 4 9 8 1

3) 4 5 9 8 1

4) 4 5 8 9 1

Yer değişen kendinden öncekilerle kendini tek tek kıyaslar.

5) 4 5 8 1 9

6) 4 5 1 8 9

7) 4 1 5 8 9

8) 1 4 5 8 9

9) 1 4 5 8 9

Q5) 1) $f(n) = n^{0.1}$, $g(n) = (\log n)^{10}$

Limit ile daha kolay bir çözüm elde edildiği için bu yolu kullandım.

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) = \Theta(g(n))$$ temelinden

yola çıkarak;

Pozitif ifadelerde geçerlidir

$$\frac{f(n)}{g(n)} > 1 \Rightarrow f(n) > g(n)$$

$$\frac{f(n)}{g(n)} < 1 \Rightarrow f(n) < g(n)$$

1) $$\lim_{n\to\infty} \frac{n^{0.1}}{(\log n)^{10}} = \frac{n^{0.1}}{(\log n)^{10}} < 1$$

$n^{0.1} \rightarrow$ üstel

$(\log n)^{10} \rightarrow$ logaritmik

Bu örnekte $(\log n)^{10}$ $n^{0.1}$'e göre daha hızlı büyür

$$\underbrace{f(n) \le c_1 g(n)}_{O(g(n))}$$ eşitliğini elde ettim.

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

$$\lim_{n\to\infty} \frac{n!}{2^n}$$

Stirling's Formula $\Rightarrow n! \cong \sqrt{2\pi n} \times \left(\frac{n}{e}\right)^n$

Wikipedia'dan yararlanıldı.

$g(n) = 2^n$

$$\frac{f(n)}{g(n)} = \frac{n!}{2^n} = \frac{\sqrt{2\pi n}\cdot n^n \cdot e^{-n}}{2^n} > 1 \Rightarrow \underbrace{f(n) > c_1 g(n)}_{\Omega(g(n))}$$

$n > n_0$, $c_1 > 0$

$f(n) \in \Omega(g(n))$

3) $f(n) = (\log n)^{\log n}$, $g(n) = 2^{(\log_2 n)^2}$

$$\frac{f(n)}{g(n)} = \sqrt[\log n]{\frac{(\log n)^{\log n}}{2^{(\log_2 n)^2}}}$$

$\sqrt[\log n]{}$ alındı

sadeleştirildi.

$n > n_0$

$c_1 > 0$

$c_2 > 0$

$$\frac{f(n)}{g(n)} = \frac{\log n}{2^{\log n}} =$$

$2^{\log n}$, $\log n$'den hızlı büyür.

$$\underbrace{f(n) \le c_2 g(n)}$$

$f(n) \in O(g(n))$