CSE222 / BİL505
Homework #2 – Solution

Q1. ***(35 pts)*** For each of the function pairs below, show whether $f(n) = O(g(n))$, or $f(n) = \Omega(g(n))$, or $f(n) = \Omega(g(n))$ by using the limit approach. Make sure you solve ***all*** indeterminities and show the details of your work.

a) $f(n) = (n^2 - 3n)^2$ *and* $g(n) = 5n^3 + n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{(n^2-3n)^2}{5n^3+n} = \lim_{n\to\infty} \frac{n^4-6n^3+9n^2}{5n^3+n} = \lim_{n\to\infty} \frac{\frac{(n^4-6n^3+9n^2)}{n^3}}{\frac{(5n^3+n)}{n^3}}$$

$$= \lim_{n\to\infty} \frac{\frac{n^4}{n^3} - \frac{6n^3}{n^3} + \frac{9n^2}{n^3}}{\frac{5n^3}{n^3} + \frac{n}{n^3}} = \lim_{n\to\infty} \frac{n - 6 + \frac{9}{n}}{5 + \frac{1}{n^2}} = \frac{\infty - 6 + 0}{5 + 0} = \infty$$

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

b) $f(n) = n^3$ *and* $g(n) = \log_2 \quad n^4$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3}{\log_2 n^4} = \frac{1}{4}\lim_{n\to\infty} \frac{n^3}{\log_2 n} \to L'Hopital \to \frac{1}{4}\lim_{n\to\infty} \frac{3n^2}{\frac{1}{n\ln 2}} = \frac{1}{4}\lim_{n\to\infty} 3n^3 \ln 2 = \infty$$

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

c) $f(n) = 5n \cdot \log_2(4n)$ *and* $g(n) = n \cdot \log_2(5^n)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{5n\log_2(4n)}{n\log_2(5^n)} = \lim_{n\to\infty} \frac{5\log_2(4n)}{n\log_2(5)} \to L'Hopital \to \lim_{n\to\infty} \frac{5\left(\frac{1}{n\ln 2}\right)}{\log_2 5}$$

$$\lim_{n\to\infty} \frac{5}{n\ln 2\log_2 5} = \frac{5}{\ln 2\log_2 5}\lim_{n\to\infty} \frac{1}{n} = 0$$

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

d) $f(n) = n^n$ *and* $g(n) = 10^n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^n}{10^n} = \lim_{n\to\infty} \left(\frac{n}{10}\right)^n = \lim_{n\to\infty} \left(\left(\frac{n}{10}\right)^n\right)^{\ln e} = \lim_{n\to\infty} e^{\ln\left(\left(\frac{n}{10}\right)^n\right)}$$

$$= \lim_{n\to\infty} e^{n\ln\left(\frac{n}{10}\right)} = \infty$$

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

e) $f(n) = 8n \cdot \sqrt[5]{2n}$ *and* $g(n) = n \cdot \sqrt[3]{n}$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{8n\sqrt[5]{2n}}{n\sqrt[3]{n}} = 8\sqrt[5]{2}\lim_{n\to\infty} \frac{n^{\frac{6}{5}}}{n^{\frac{4}{3}}} = 8\sqrt[5]{2}\lim_{n\to\infty} \frac{1}{n^{\frac{2}{15}}} = 0$$

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

Limit approach:

- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \rightarrow f(n) = O(g(n))$
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \rightarrow f(n) = \Omega(g(n))$
- $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \rightarrow f(n) = \Theta(g(n))$

Q2. ***(35 pts)*** Analyze the ***worst-case*** time complexity of the following methods.

PS: Assume that all of the arrays has a length of n, $where\ n \in Z^+$.

a)

```
static void methodA (String str_array[]) {
    for (int i = 0; i < str_array.length; i++)
      str_array [i] = "";
}
```

Since this method iterates through the array once and makes a constant-time operation at each iteration, the worst-case time complexity of methodA is O(n).

b)

```
static void methodB (String str_array[]) {
    for (int i = 0; i < str_array.length; i++)
      methodA (str_array);
    for (int j = 0; j < str_array.length; j++)
      System.out.println(str_array[j]);
}
```

The first loop iterates through the array once, so it has n iterations. But, at each iteration, methodA (with O(n) complexity) is called. So, this loop has a complexity of O(n*n) = O(n^2).

The second loop on the other hand, has n iterations and each iteration has a constant time operation. Thus, its complexity is O(n*1)=O(n).

The overal complexity of methodB becomes O(n^2 + n) = O(n^2).

c)

```
static void methodC (String str_array[]) {
    for (int i = 0; i < str_array.length; i++)
        for (int j = 0; j < str_array.length; j++)
            methodB (str_array);
}
```

This method contains a nested loop, each with n iterations. So, just to complete all these iterations (without implementing anything in the loop) means n^2 iterations in total. However, the loop calls

methodB, which has the worst-case time complexity of O(n^2). Since we call this method n^2 times, the complexity of methodC becomes O(n^2 * n^2) = O(n^4).

d)

```
static void methodD (String str_array[]) {
    for (int i = 0; i < str_array.length; i++){
        System.out.println(str_array[i]);
        str_array[i--] = "";
    }
}
```

Since this method has an endless loop (due to "i--"), we cannot talk about complexity here. An algorithm should consist of a **finite** number of steps.

e)

```
static void methodE (String str_array[]) {
    for (int i = 0; i < str_array.length; i++)
      if (str_array [i] == "")
          break;
}
```

In this method, we have a loop with n iterations. At each iteration, an constant time operation is made. Thus, the worst-case time complexity of methodE is O(n*1) = O(n).

Q3. ***(30 pts)*** Design an algorithm to find the maximum difference between two elements of a given array $A = [a_0, a_1, \dots, a_{n-1}]$ $where\ a_i \in \mathbb{Z}\ for\ i \in \mathbb{N}\ and\ i < n$. Provide the ***pseudo-code*** of the algorithm along with an explanation and analyze its worst-case time complexity. Repeat this process for the following cases:

a) Assuming the array is sorted in ascending order.
b) Assuming the array is not sorted.

PS: To be graded, your algorithms' worst-case time complexities should be linear time at most.

a)

Pseudo-Code:

```
FindMaxDiff (array)
        n ← length of array
        Return array[n-1] - array[0]
```

Explanation:

The maximum difference in an array is the difference between the largest and smallest integers of the array. Since the array is already sorted, the first element is the smallest and the last element is the largest. Thus, we can just return the difference of these two elements.

PS: It was assumed that the array has at least 1 element.

Time Complexity:

If we ignore the time complexity of calculating the length of the array, the complexity of the given algorithm is O(1). Because it only has 2 operations and both can be done in constant time.

b)

Pseudo-Code:

```
FindMaxDiff (array)
        max_element ← array[0]
        min_element ← array[0]
        For each element in array
            If element > max_element Then
                max_element ← element
            End If
            If element < min_element Then
                min_element ← element
            End If
        End For

Return max_element – min_element
```

Explanation:

The maximum difference in an array is the difference between the largest and smallest integers of the array. Since the array is not sorted, we don't know the location of maximum and minimum integers. Thus, we have to iterate the array and check each element to find the maximum and minimum elements. Once we find them, we return their difference.

PS: It was assumed that the array has at least 1 element.

Time Complexity:

The declarations of the minimum and maximum elements can be done in constant time. After that, we have a loop which iterates the array once. In this loop, a few constant-time operations are done. Thus, the complexity of the loop becomes linear, and it dominates the whole algorithm's complexity. As a result, we define the worst-case time complexity of the given algorithm as O(n).