CSE 222 — HW 2

Burak Ersoy
1901042703

1-)

a) $\lim\limits_{n\to\infty} \dfrac{(n^2-3n)^2}{5n^3} = \lim\limits_{n\to\infty} \dfrac{n^4-6n^3+9n^2}{5n^3} = \lim\limits_{n\to\infty} \dfrac{n-6+\frac{9}{n}}{5}$

(Divide n^3)

As an approaches infinity, this simplifies to; $\lim\limits_{n-\infty} \dfrac{n}{5} = \infty$

Since the limit is infinity, $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = \infty$, $f(n) = \Omega(g(n))$

b) $\lim\limits_{n\to\infty} \dfrac{n^3}{\log_2 n^4} = \dfrac{1}{4} \cdot \lim\limits_{n\to\infty} \dfrac{n^3}{\log_2 n}$ = Apply L' Hospital

$= \dfrac{1}{4} \cdot \lim\limits_{n\to\infty} \dfrac{3n^2}{\frac{1}{x.\ln(2)}} = \dfrac{1}{4} \cdot \lim\limits_{n\to\infty} 3\ln 2 . n^3 = \dfrac{1}{4} . 3\ln 2 . \lim\limits_{n\to\infty} n^3$

$\dfrac{1}{4} . 3\ln 2 . \lim\limits_{n\to\infty} n^3 = \infty$, $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = \infty$, $f(n) = \Omega(g(n))$

c) $\lim\limits_{n\to\infty} \dfrac{5.\log_2(n^4)}{n.\log_2(n^5)} = \lim\limits_{n\to\infty} \dfrac{20.\log_2 n}{n.5.\log_2 n} = \lim\limits_{n\to\infty} \dfrac{4}{n}$

which approaches 0 as n becomes large, so $f(n) = O(g(n))$

d) $\lim\limits_{n\to\infty} \dfrac{n}{10n} = \lim\limits_{n\to\infty} \dfrac{1}{10} = \dfrac{1}{10}$

$0 < c = \dfrac{1}{10} < \infty$, $\dfrac{1}{10}$ is constant, therefore, $f(n) = \Theta(g(n))$

e) $\lim\limits_{n\to\infty} \dfrac{8.n^{5/2}}{n.3^{n/2}} = \lim\limits_{n\to\infty} 8.\dfrac{n^{5/2}}{n.3^{n/2}} = 8.\lim\limits_{n\to\infty}. \dfrac{n^{5/2}}{3^{n/2}} . \lim\limits_{n\to\infty} \left(\dfrac{1}{n}\right)$

$\lim\limits_{n\to\infty} \dfrac{n^{5/2}}{3^{n/2}} = 0$, because, this a comparison between a polynomial and exponential function. As n approaches infinity, the exponential term $(3^{n/2})$ grows much faster than polynomial term $(n^{5/2})$.

$\lim\limits_{n\to\infty} \dfrac{1}{n} = 0$

Therefore, $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$, $f(n) = O(g(n))$

2-)

a-) The loop variable i is initialized once, so it's a constant time operation, O(1). The loop condition i< str_array_length is checked (n+1) times (where (n) is the length of the array.) checking the condition is a constant time operation, so this contributes O(n). Inside the loop, we're assigning an empty string to each array element. String assignment is generally considered a constant time operation, so this also contributes O(n).
The loop variable i is incremented (n) times which is O(n).
Adding these up, the dominant term is O(n), and since we ignore lower-order terms and constant factors in Big O notation the worst-case time complexity of method A is O(n).

b-) The first for loop calls methodA, which is O(n), n times. So, this loop contributes O(n^2) complexity. The second for loop prints each element of the array, contributing O(n) complexity. The combined time complexity is the sum of the complexities of these two loops, dominated by the first loop.
Time complexity O(n^2), because the first loop is the dominant factor.

c-) The outer loop runs (n) times, wher (n) is the length of the array.
The inner loop runs (n) times for each iteration of the outer loop resulting in (n^2) total iterations of the inner loop.
Within the inner loop, methodB is called, which has a time complexity of O(n^2)
To find overall time complexity: multiply the complexities of the nested loops with the complexity of the method call within the innermost loop:
(n) (outer loop) * (n)(inner loop) * O(n^2) (method B) = O(n^4)
Therefore, the worst-case time complexity of method C is O(n^4)

d-) This method prints each element and then resets it to an empty string.
However, the i-- operation inside the loop will cause the loop to iterate indefinetely, leading to an infinite loop as the index i is decremented each time after incrementing.
Time comlexity not applicable (infinite loop)

e) The loop runs at most (n) times, where (n) is the length of the array. The comparison str_array [i] = "" is a constant time operation, O(1). If an empty string is found early in the array, the loops breaks, resulting in fewer then (n) iterations.

In the worst case, the empty string is not present, or it is the last element, so the loop will run (n) times. Therefore, the worst-case complexity of methodE is O(n).

3-) a) Algorithm maxDifferenceSortedArray (A):

```
n = length (A)
if n<2:
    return "Array must have at least two elements"
max_diff = A[n-1] - A[0]
return max_diff
```

Since the array is sorted, the smallest elements is A[0], and the largest is A[n-1]. The maximum difference is simply the difference between these two.

Worst-case complexity : The time complexity is O(1) because it performs a constant number of operations regardless of the size of array.

b) Algorithm MaxDifferenceUnsortedArray (A):

```
n = length (A)
if n<2 :
    return "Array must have at least two elements"
min_val = A[0]
max_val = A[0]
for i from 1 to n-1:
    if A[i] < min_val :
        min_val = A[i]
    if A[i] > max_val:
        max_val = A[i]
max_diff = max_val - min_val
return max_diff
```

We initialize min-val and max-val with the first element. Then, we iterate through the array, updating min-val and max-val as we find smaller or larger elements, respectively. The maximum difference is the difference between max-val and min-val.

Worst-case Complexity: The time complexity is O(n), because we scan through all (n) elements of the array once.

Both algorithms meet the requirements of having a worst-case time complexity of linear time or better.