Data Structures & Algorithms 2
Homework #1

Exo1 6marks Exo2 6marks Exo3 7marks + 1 mark for a well presented response

(1

Exercise 1 (6 marks 1 mark for each function)

a) $f(N) = 100 \log N^2 + 10 N^2 \log N$

$O(N^2 \log N)$)

$$\lim_{n\to\infty} \frac{100 \log N^2 + 10 N^2 \log N}{N^2 \log N} = 10$$

b) $f(N) = ((N + 1)(N + 2))/2$

$O(N^2)$

$$f(N) = \frac{(N+1)(N+2)}{2} = \frac{1}{2}N^2 + \frac{3}{2}N + 1$$

Polynomial => Biggest term N^2

$$\lim_{n\to\infty} \frac{\frac{1}{2}N^2 + \frac{3}{2}N + 1}{N^2} = \frac{1}{2}$$

c) $f(N) = N^2 (2\log N + \log N) + N^3$

$O(N^3)$

$f(N) = N^2 (2\log N + \log N) + N^3 = 3 N^2 \log N + N^3$

$$\lim_{n\to\infty} \frac{3N^2 \log N + N^3}{N^3} = 1$$

d) $f(N) = N \log^2 N + N \log \log N$

$O(N\log^2 N)$

$$\lim_{n\to\infty} \frac{N\, log^2 N + N \log\log N}{N log^2 N} = \lim_{n\to\infty} \frac{N\, log^2 N}{N log^2 N} + \lim_{n\to\infty} \frac{N \log\log N}{N log^2 N} = 1 + 0 = 1$$

$$\lim_{n\to\infty} \frac{\log\log N}{log^2 N} = 0 \text{ (TH. Hôpital)}$$

e) $f(N) = N^{2} (N + 2N) + (N^{3} . N^{3})$

$O(N^6)$

$f(N) = 3N^{3} + N^{6}$

Biggest term N^{6}

f) $f(N) = N^{1/4} + \log N$

$O(N^{1/4})$

$$\lim_{n\to\infty} \frac{N^{1/4} + \log N}{N^{1/4}} = 1 + \frac{\log N}{N^{1/4}} = 1 + \lim_{n\to\infty} \frac{\log N}{N^{1/4}} = 1 + 0 = 1$$

Exercise 2 (6 marks)

Describe the worst case running time of the following pseudocode functions in Big-Oh notation in terms of the variable n. Justify your answer.

(A)	<pre>void **fct1**(int n) { for (int i = n*n; i > 0; i--) { for (int k = 0; k < n; ++k) print("k = " , k); for (int j = 0; j < i; ++j) print("j = " , j); for (int m = 0; m < 5000; ++m) print("m = " , m); } }</pre>
(B)	<pre>int **fct2** (int n, int m) { if (n < 10) return n; else if (n < 100) return fct2 (n - 2, m); else return fct2 (n/2, m); }</pre>
(C)	<pre>void **fct3** (int n) { for (int i = 0; i < n; ++i) { for (int j = 0; j < n; ++j) print("j = " j); for (int k = 0; k < i; ++k) {</pre>

	`` print(”k = " ,, k); for (int m = 0; m < 100; ++m) print(”m = ”, m); } } } ```

A) fct1 : O(N^4) (2marks :0.5 final result +1.5 justification)

- The outer loop (i) has complexity O(N^2).
- The first inner loop has complexity O(N)
- the second inner loop has complexity O(N^2) , because it depends on i which has as upper bound N*N (N^2)
- the third loop has complexity O(1) since there is 5000 iterations which is constant and the operations that it performs take constant time .
- Thus the complexity of all the inner loops = O(N^2)

  O(N)+ O(N^2)+O(1) = O(N^2). •

  Thus, the function has complexity O(N^2) * O(N^2) = O(N^4)

B) fct2 : O(log(N)) (2marks :0.5 final result +1.5 justification)

for the outer and inner if , the complexity is O(1) .

In the worst case the fct2 performs x calls until n > 100 for each call n is devided by 2 .

$n/2^x = 100$

$n = 100\ 2^x$

log n = x log100

x = (1/log100) log (n)

Thus the fct2 is of logarithmic complexity

C) fct3 : $O(N^2)$ (2marks :0.5 final result +1.5 justification)

- The outer loop has complexity O(N).

  - The first inner loop also has complexity O(N)

  -The second inner loop has a complexity O(N) because it has as upper bound i

  - The third inner loop will has complexity O(1) because it runs 100 times, a constant time operation (print).

  • Thus, the function has complexity

  $O(N) * (O(N) + O(N) + O(1)) = O(N^2)$

Exercise 3 (7 marks)

Suppose you have a large linked list of n integers and you want to print them in reverse order (the numbers closer to the end of the list first).

The first version of your code follows this algorithm:

- Traverse the list from the beginning to determine what n is.
- For i = n, n − 1, n − 2, ..., 1, traverse the list from the beginning to the i th element and print it.

The second version of your code looks like this, calling ***printReverse*** on the first node in the list.

```
class ListNode {
     int x;
     ListNode next;
     Public :
     void printReverse() {
          if (next != null) next.printReverse();
          print(x);
     }
}
```

- Give an asymptotic analysis of the running time using big-O for both algorithms .Which version is faster?

Algorithm 1 **(3 marks** : 0.5 final result +2.5 justification **)**

($O(N^2)$) Quadratic

Step 1 : Traversing the linked list from the beginning on O(N)

Step 2 :

- to print n^{th} element we perform n iteration
- to print $(n-1)^{th}$ element we perform n-1 iteration

....

- to print $(1)^{th}$ element we perform 1 iteration

Thus

$n+(n-1)+(n-2)+\ldots\ldots\ldots\ldots\ldots\ldots.1 = n(n+1)/2 = \quad O(N^2)$

Algorithm 1 : $O(N) + O(N^2) = O(N^2)$

Algorithm 2 **(O(N)) Linear** **(3 marks** : 0.5 final result +2.5 justification **)**

Recursive call of reduced list by one element , then the print of current element (n-1) call

$T(N) + T(N-1) + 1$

$T(N) = (T(N-2) + 1) + 1$

...

$T(N) = T(N-k) + k \quad K = N$

$T(N) = N \quad = O(N)$

From this analysis, since N^2 grows faster than N the second version of the algorithm to print the list in reversed order is faster than the first algorithm. **(1 mark)**