Department of Electrical and Software Engineering
Schulich School of Engineering

ENSF 694 - Principles of Software Development II
Summer 2023

Lab 1 – June 26, 2023
Topic: Algorithmic Complexity Analysis

Q1. Determine the Big-O notation for the following –

a) 5 + n (3 + 3n) + 4n
Solution: O(n^2)

b) 3n + 6 (n + 2n) n + $\frac{n}{4}$
Solution: O(n^2)

c) n^2 + 3n^3 log n + 5n + 10 + 6n^3 + n (log n)^2
Solution: O(n^3 log n)

Q2. Determine the complexity of the following code snippets:

a)
```
for (i = 0; i < n; i++)
{
     a = 0;
     for (j = 3; j < n; j++)
    {
       a = a + 1;
    }
}
b = 2;
for (k = 0; k < n; k++)
{
   b = b + 100;
}
```
Solution: O(n^2)

b)
```
a = 3;
for (i = 1; i <= n; i++)
{
     for (j = 1; j <= i; j++)
     {
          a  = a + 2;
     }
}
```
Solution: O(n^2)

Q3. Determine the complexity of the Fibonacci series computation using iterative approach and recursive approach and provide a decision on the better option:

a) **Iterative approach:**

```
int fibonacci(int n)
{
    int f_old= 1, f_new = 1, f_older;
    for (i = 2; i < n; i++)
   {
          f_older = f_old;
          f_old = f_new;
          f_new = f_older + f_old;
   }
   return f_new;
}
```

Solution: O(n)

b) **Recursive approach:**

```
int fibonacci(int n)
{
    if (n <= 1)
        return n;
    else
        return fibonacci(n-1) + fibonacci(n-2);
 }
```

Solution: $O(2^n)$

Solution Hint:

$T(n-2) \approx T(n-1)$ // assumption

$T(n) = T(n-1) + T(n-1) + 1 = 2*T(n-1) + 1$

So, $T(n-1) = 2*T(n-2) + 1$

$$
\begin{aligned}
T(n) &= 2*[2*T(n-2) + 1] + 1 = 4*T(n-2) + 3 \\
&= 2*[2*[2*T(n-3) + 1] + 1] + 1 = 8*T(n-3) + 7 \\
&= 2*[2*[2*[2*T(n-4) + 1]+ 1] + 1] + 1 = 16*T(n-4) + 15 \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&= 2^r *T(n-r) + (2^r-1)
\end{aligned}
$$

Here, $T(0) = 1$

So, for $T(0)$, $n - r = 0$, which gives us $r = n$.

Therefore,

$$T(n) = 2^n *T(0) + (2^n -1) = 2^n + 2^n - 1 = O(2^n)$$