COMP108 Data Structures and Algorithms

Week 03 Tutorial Exercises

Due: 17 February 2023, 5:00pm

1. State the order of growth of the following functions in big-O notation. For example, the function $3n$ is $O(n)$. Remember you need to write the big-O notation.

   (a) $10 + 3n + 2n^2 + n^4 + 5n^3$ O(n^4)

   (b) $5 + 3n^3 + 2n^2 \log n + 5n + 2n^2$ O(n^3)

   (c) $\sqrt{n^5} + n^3$ O(n^3)

2. Consider each of the following algorithms. What is the time complexity in big-O notation? Give justifications.

   Note: If two loops are run one after another, the overall time complexity is the sum of the complexities of the two loops.

   (a)
   ```
   // Assume n and m are given positive integers
   x ← n
   while x > 1 do
       x ← x − 2
   for y ← 1 to m do
       output y
   ```

First loop: function: n/2, Big-O notation: O(n)
Second loop: function: m, Big-O notation: O(m)
Total: function: n/2 + m, Big-O notation: O(n + m)

The first loop iterates O(n) times
The second loop iterates O(m) times
Hence the total time complexity is O(n + m)

   (b)
   ```
   // Assume n is a given positive integer being power of 2
   count ← 0
   x ← n
   while x > 1 do
   begin
       x ← x/2
       count ← count + 1
   end
   output count
   ```

function: $\log_2(n)$, Big-O notation: O($\log_2(n)$)

The loop iterates O($\log(n)$) times

(c)
```
// Assume n is a given positive integer being power of 2
x ← n
y ← n
while x > 1 do
    x ← x − 2
while y > 1 do
    y ← y/2
```

First loop: function: n/2, Big-O notation: O(n)
Second loop: function: $\log_2(n)$, Big-O notation: O($\log(n)$)
Total: function: n/2 + $\log_2(n)$, Big-O notation: O(n)

The first loop iterates O(n) times
The second loop iterates O($\log(n)$) times
Hence the total time complexity is O(n)

(d)
```
// Assume n and m are given positive integers
x ← n
while x > 1 do
begin
    x ← x − 2
    for y ← 1 to m do
        output y
end
```

Inner loop: function: m, Big-O notation: O(m)
Outer loop: function: n/2, Big-O notation: O(n)
Total: function m(n/2), Big-O notation: O(mn)

The inner loop iterates O(m) times
The outer loop iterates O(n) times
For each outer loop iteration, we need to iterate the inner loop
Hence the total time complexity is O(mn)

(e)
```
// Assume n and m are given positive integers
x ← n
while x > 1 do
begin
    if x == n then
    begin
        for y ← 1 to m do
        begin
            output y
        end
    end
    x ← x − 2
end
```

Inner loop: function: m, Big-O notation: O(m)
Outer loop: function: n/2, Big-O notation: O(n)
The inner loop only runs when x = n. (Only the case the first time the outer loop runs)
Total: function m + n/2, Big-O notation: O(m + n)

The inner loop iterates O(m) times. The outer loop iterates O(n) times
The inner loop only tuns when x = n (the first time the outer loop runs)
Hence the total time complexity is O(m + n)

(f)
```
// Assume n is a given positive integer being power of 2
x ← 1
while x ≤ n do
begin
    x ← x + 3
    y ← 1
    while y ≤ n do
    begin
        y ← y * 2
    end
end
```

Inner loop: function: n/3, Big-O notation: O(n)
Outer loop: function: $\log_{\frac{1}{2}}(n)$, Big-O notation: O(log n)
Total: function :n/3 $\log_{\frac{1}{2}}(n)$, Big-O notation: O(n log n)

inner loop iterates O(log n) times
outer loop iterates O(n) times
For each outer loop iteration, we need to iterate the inner loop
Hence the total time complexity is O(n log n)

(g)
```
// Assume n and m are given positive integers
for x ← 1 to n do
begin
    for y ← 1 to m do
    begin
        for z ← 1 to n do
        begin
            output x + y + z
        end
    end
end
```

Inner inner loop: function: n, Big-O notation: O(n)
Inner loop: function: m, Big-O notation: O(m)
Outer loop: function: n, Big-O notation: O(n)
Total: function: mn^2, Big-O notation: O(mn^2)

The inner inner loop iterates O(n) times
The inner loop iterates O(m) times
The outer loop iterates O(n) times
For each outer loop iteration, we need to iterate the inner loop
Hence the total time complexity is O(mn^2)