3.3 Complexity of Algorithms

Commonly Used Terminology for the Complexity of Algorithms

Complexity	Terminology
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	Linearithmic complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

3.3 pg 229 # 1

Give a big-O estimate for the number of operations (where an operation is an addition or a multiplication) used in this segment of an algorithm.

t := 0
for i := 1 **to** 3
  for j := 1 **to** 4
    $t := t + ij$

$t + ij$ will result in 2 operations per loop iteration (one multiplication and one addition).
The j-for loop will execute $t + ij$ 4 times.
The i-for loop will execute 3 times.
Since the j-for loop is executed for every iteration for the i-for loop, then we have $2 \cdot 3 \cdot 4 = 24$ total operations.
Therefore, the algorithm is $O(1)$ (i.e. constant complexity).

3.3 pg 229 # 3

Give a big-O estimate for the number of operations, where an operation is a comparison or a multiplication, used in this segment of an algorithm (ignoring comparisons used to test the conditions in the for loops, where $a_1, a_2, ..., a_n$ are positive real numbers).
m := 0
for i := 1 **to** n
  for j := i + 1 **to** n
    m := **max**($a_i a_j, m$)

For the first iteration of the i-for loop (the outer loop), the j-for loop (the inner loop) will run 2 to n times ($n - 1$ times).
For the second iteration of the i-for loop, the j-for loop will run 3 to n times ($n - 2$ times).
...
For the third to the last iteration of the i-for loop, the j-for loop will run $n - 1$ to n times (2 times).
For the second to the last iteration of the i-for loop, the j-for loop will run from n to n times (1

time).
For the last iteration of the i-for loop, the j-for loop will run 0 times because $i+1 > n$.
Now we know that the number of times the loops are run is

$$1 + 2 + 3 + \ldots + (n-2) + (n-1) = n(n-1)/2$$

So we can express the number of total iterations as $n(n-1)/2$.
Since we have two operations per loop (one comparison and one multiplication), we have $2 \cdot n(n-1)/2 = n^2 - n$ operations.

So $f(n) = n^2 - n$
$f(n) \leq n^2$ for $n > 1$.
Thus, the algorithm is $O(n^2)$ with our witnesses $C = 1$ and $k = 1$.

3.3 pg 230 #21

What is the effect in the time required to solve a problem when you increase the size of the input from n to $n+1$, assuming that the number of milliseconds the algorithm used to solve the problem with input size n is each of these function? [Express you answer in the simplest form possible, either as a ratio or a difference. Your answer may be a function of n or a constant.]

a) $\log n$

$\log(n+1) - \log(n) = \log((n+1)/n)$
Note that as n grows large, the expression $((n+1)/n)$ approaches 1 and that $\log 1 = 0$.
This means that the required time for $n+1$ is negligible.

b) $100n$

$100(n+1) - 100n = 100n + 100 - 100n = 100$
This means that 100 additional ms is required.

c) n^2

$(n+1)^2 - n^2 = n^2 + 2n + 1 - n^2 = 2n + 1$
Additional $2n+1$ ms is required.

d) n^3

$(n+1)^3 - n^3 = n^3 + 3n^2 + 3n + 1 - n^3 = 3n^2 + 3n + 1$
Additional $3n^2 + 3n + 1$ ms is required.

e) 2^n

$2^{n+1}/2^n = 2$
2 times as long.

g) $n!$

$(n+1)!/n! = n + 1$
$n+1$ times as long.