Worksheet 9: Summing Execution Times

In preparation: Read Chapter 4 to learn more about big-Oh notation.

Function	Common name	Running time
N!	Factorial	
2^n	Exponential	> century
N^d, d > 3	Polynomial	
N^3	Cubic	31.7 years
N^2	Quadratic	2.8 hours
N sqrt n		31.6 seconds
N log n		1.2 seconds
N	Linear	0.1 second
sqrt (n)	Root-n	$3.2 * 10^{-4}$ seconds
Log n	Logarithmic	$1.2 * 10^{-5}$ seconds
1	Constant	

The table at left, also found in Chapter 4, lists functions in order from most costly to least. The middle column is the common name for the function.

Suppose by careful measurement you have discovered that a program has the running time as shown at right. Describe the running time of each function using big-Oh notation.

$3n^3 + 2n + 7$	$O(n^3)$
(5 * n) * (3 + log n)	O(nlog n)
1 + 2 + 3 + … + n	$O(n^2)$
$n + \log n^2$	O(n)
((n+1) log n) / 2	O(nlog n)
$n^3 + n! + 3$	O(n!)
$2^n + n^2$	$O(2^n)$
n (sqrt(n) + log n)	O(nsqrt(n)

Worksheet 9: Summing Execution Times Name:

Using the idea of dominating functions, give the big-Oh execution time for each of the following sequences of code. When elipses (…) are given you can assume that they describe only constant time operations.

for (int i = n; i > 0; i = i / 2) {
  …
}
for (int j = 0; j * j < n; j++) …
	O(sqrt(n))

for (int i = 0; i < n; i++) {
  for (int j = n; j > 0; j = j / 2) {
  …
  }
  for (int k = 0; k < n; k++) {
  …
  }
}
	$O(n^2)$

for (int i = 0; i < n; i++)
  …
for (int j = 0; j * j < n; j++)
  …
	O(n)

for (int i = 0; i < n; i++)
  …
for (int j = n; j > 0; j--)
  …
	O(n)

for (int i = 1; i * i < n; i += 2)
  …
for (int i = 1; i < n; i += 5)
  …
	O(n)