1/23

Efficiency -- Complexity Analysis

Efficiency

- How efficient is a program? -- How much time does it take a program to complete -- How much memory does a program use? -- How do these change as the amount of data changes?

Technique

- View algorithms as Java programs
- Count executable statements in program or method
- Find number of statements as function of the amount of data
- Focus on *dominant* term in the function

Counting Statements

```
int x;  // one statement
x = 12; // one statement
int y = z * x + 3 % 5 * x / i; // 1
x++;    // one statement
boolean p = x < y && y % 2 == 0 || z >= y * x;  // 1
int[] list  = new int[100]; // 1
list[0] = x * x + y * y;    // 1
```

Ex

```
public int total(int[] values) {
  int result = 0;
  for (int i = 0; i < values.length; i++) {
      result += values[i];
  }
  return result;
}
```

Big O

Formal Definition

- T(N) is O(F(N)) if there are positive constants c and N~~0~~ such that T(N) <= cF(N) when N >= N~~0~~ -- N is the size of the data set the algorithm works on -- T(N) is a function that characterizes the *actual* running time of the algorithm -- F(N) us a function that characterizes an upper bounds on T(N). It is a limit on the running time of the algorithm -- c and N~~0~~ are constants
- T(N) is the actual growth rate of the algorithm
- F(N) is the function that bounds the growth rate
- T(N) may not necessarily equal F(N)