Analysis, Part 2

1. Bad News: Figuring out tight upper bounds is hard
   a. As hard as solving a differential equation…
   b. …which can't be done in general (Really! Most differential equations have no closed solution.)

2. Good News:
   a. We can always **be conservative** and give an upper bound that is too big
   b. Most cases you'll encounter are common and look like **the table on the back of this page**

3. Solve by example:
   a. int *f* (int *x*) { if (*x* <= 0) { return 1;} else { return *x* * *f* (*x* - 1); }

   b. int *g* (int *x*) { if (*x* <= 0) { return 1;} else { return *x* * *g* (*x* / 2); }

   c.
```
int[ ] merge (int[ ] x, int[ ] y) {
   int n = x.length + y.length;
   int[ ] z = new int[n];
   int i, j, k;
   for (k = 0; (i < x.length) && (j < y.length); ++k)
      if (x[i] < y[j]) {   z[k] = x[i]; ++i; }
      else             {   z[k] = y[j]; ++j; }

   if (i < x.length) System.arraycopy(x, i, z, k, x.length – i);
   else              System.arraycopy(y, j, z, k, y.length – j);

   return z;
}
```

4. Understanding bounds:
   a. Things you already know:
      i. $a * b$ = "a **added** to itself b times"
      ii. a^b = "a **multiplied** by itself b times"
      iii. n / c = "how many times you have to **subtract** c from n to reach 0"
      iv. $\log_c n$ = "how many times you have to **divide** n by c to reach 1"
   b. Analyze:
      i. int *f* (int *x*) { if (*x* <= 0) { return 1;} else { return *x* * *f* (*x* - 1); }
      ii. int *g* (int *x*) { if (*x* <= 0) { return 1;} else { return *x* * *g* (*x* / 2); }

Example Function	Time Bound

int f (int n) {
  return n * 2 + 14 + n * n * n;
}
	$O(1)$

int pow(int b, int n) {
  if (n == 0) {
  return 1;
  } else {
  int x = pow(b, **n / 2**);
  return x*x;
  }
}
	$O(\log_2 n)$

int factorial(int n) {
  if (n == 1) return 1;
  else return n * factorial(**n – 1**);
}
 	
float sum(float[ ] x) {
  int n = x.length;
  float s;
  for (int i = 0; i < n; ++i) sum += x[i];
  return s;
}
	$O(n)$

int silly(int n) {
  int sum = 0;
  for (int i = 0; i < n; ++i) sum += i;
  return sum + silly(**n / 2**) + silly(**n / 2**);
}
	$O(n \log n)$

float matrixVector(float[ ][ ] A, float[ ] v) {
  int n = v.length;
  float sum = 0;
  for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
  sum += A[i][j] * v[i];
  return sum;
}
	$O(n^2)$

float[ ][ ] matrixMatrix(float[ ][ ] A, float[ ][ ] B) {
  int n = M.length;
  float C = new float[n][n];
  for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
  for (int k = 0; k < n; ++k)
  C[i][j] += A[i][k] * B[k][j];
  return C;
}
	$O(n^3)$

int fibonacci(int n) {
  if (n <= 1) return n;
  else return fibonacci(**n – 1**) + fibonacci(**n - 2**);
}
	$O(2^n)$