Algorithm Analysis

Annalise Tarhan

March 29, 2021

1 What value is returned by the following function? Express your answer as a function of n. Give the worst-case running time using the Big Oh notation.

```
Mystery(n)
        r = 0
        for i = 1 to n-1 do
                for j = i+1 to n do
                        for k = 1 to j do
                                r = r+1
        return r
```

This function is equivalent to the summation

$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{j} 1$

The innermost sum, $\sum_{k=1}^{j} 1$, is equivalent to j, so simplify to

$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} j$

Because $\sum_{j=i+1}^{n} j = \sum_{j=1}^{n} j - \sum_{j=1}^{i} j$, we can use the formula $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
Applying it twice gives

$\sum_{i=1}^{n-1} (\frac{n(n+1)}{2} - \frac{i(i+1)}{2})$

$1/2 \sum_{i=1}^{n-1} (n^2 + n - i^2 + i)$

$1/2 (\sum_{i=1}^{n-1} n^2 + \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i^2 + \sum_{i=1}^{n-1} i)$

Now, apply the previous formula again as well as $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

$1/2(n^2(n-1) + n(n-1) - \frac{(n-1)(n)(2n-1)}{6} + \frac{(n-1)n}{2})$

$1/2(n^3 - n^2 + n^2 - n - 1/6(2n^3 - 3n^2 + n) + 1/2(n^2 - n))$

$1/2(n^3 - n - 1/3n^3 + 1/2n^2 - 1/6n + 1/2n^2 - 1/2n)$

$1/3n^3 + 1/2n^2 - 5/6n$

Therefore, the function returns $1/3n^3 + 1/2n^2 - 5/6n$. In the function, each instruction inside the loops increments the return value by one, so the number of instructions executed is roughly equal to the number returned by the function. Since the highest power of n in that return value is n^3, $Mystery(n) = O(n^3)$.

2 What value is returned by the following function? Express your answer as a function of n. Give the worst-case running time using Big Oh notation.

```
Pesky(n)
          r=0
          for i=0 to n do
                    for j=1 to i do
                              for k=j to i+j do
                                        r = r+1
          return(r)
```

This function is equivalent to the summation

$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j}^{i+j}1$

Similarly to the previous problem, the innermost loop is equivalent to incrementing r by i+1, so simplify to

$\sum_{i=1}^{n}\sum_{j=1}^{i}(i+1)$

$\sum_{i=1}^{n}(\sum_{j=1}^{i}i+\sum_{j=1}^{i}1)$

$\sum_{i=1}^{n}(i^2+\frac{i(i+1)}{2})$

$1/2\sum_{i=1}^{n}(3i^2+i)$

$1/2(3\sum_{i=1}^{n}i^2+\sum_{i=1}^{n}i)$

$1/2(\frac{n(n+1)(2n+1)}{2}+\frac{n(n+1)}{2})$

$1/2n^3+n^2+1/4n+1/4$

The function returns $1/2n^3+n^2+1/4n+1/4$, so by the same reasoning as before, $Pesky(n)=O(n^3)$.

3 What value is returned by the following function? Express your answer as a function of n. Give the worst-case running time using Big Oh notation.

```
Pestiferous(n)
          r=0
          for i=1 to n do
                    for j=1 to i do
                              for k=j to i+j do
                                        for l=1 to i+j-k do
                                                  r = r+1
          return(r)
```

This function is represented by

$\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=j}^{i+j} \sum_{l=1}^{i+j-k} 1$

The innermost loop is equivalent to incrementing the result by $i + j - k$, so the summation simplifies to

$\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=j}^{i+j} (i + j - k)$

$\sum_{i=1}^{n} \sum_{j=1}^{i} ((\sum_{k=j}^{i+j} i + j) - \sum_{k=j}^{i+j} k)$

In the first part of the split innermost term, $i + j$ is constant relative to k, so the value of its sum is $(i + j)(i + 1)$, since the look will execute $i + 1$ times. The second part of that split term can be further divided.

$\sum_{i=1}^{n} \sum_{j=1}^{i} ((i + j)(i + 1) - (\sum_{k=1}^{i+j} k - \sum_{k=1}^{j-1} k))$

$\sum_{i=1}^{n} \sum_{j=1}^{i} (i^2 + ij + i + j - \frac{(i+j)(i+j+1)}{2}) + \frac{(j-1)j}{2}$

$\sum_{i=1}^{n} \sum_{j=1}^{i} (3/2i^2 + 2ij + 3/2i + 2j)$

The terms that don't include j are constant inside the inner sum, so simply multiply them by i.

$\sum_{i=1}^{n} (3/2i^3 + 3/2i^2 + 2i \sum_{j=1}^{i} j + 2 \sum_{j=1}^{i} j)$

$\sum_{i=1}^{n} (3/2i^3 + 3/2i^2 + 2i\frac{i(i+1)}{2} + 2\frac{i(i+1)}{2})$

$\sum_{i=1}^{n} (5/2i^3 + 7/2i^2 + i)$

$5/2 \sum_{i=1}^{n} i^3 + 7/2 \sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i$

$5/2(\frac{n(n+1)}{2})^2 + 7/2(\frac{n(n+1)(2n+1)}{6}) + \frac{n(n+1)}{2}$

$5/2(n^4 + 2n^3 + n^2) + 7/12(2n^3 + 3n^2 + n) + 1/2(n^2 + n)$

$5/2n^4 + 37/6n^3 + 19/4n^2 + 13/12n$

The function returns $5/2n^4 + 37/6n^3 + 19/4n^2 + 13/12n$, whose highest term is n^4, so $Pestiferous(n) = O(n^4)$.

4 What value is returned by the following function? Express your answer as a function of n. Give the worst-case running time using Big Oh notation.

```
Conundrum(n)
        r=0
        for i=1 to n do
                for j=i+1 to n do
                        for k=i+j-1 to n do
                                r=r+1
        return(r)
```

$\sum_{i=1}^{n} \sum_{j=1+i}^{n} \sum_{k=i+j-1}^{n} 1$

$\sum_{i=1}^{n} \sum_{j=1+i}^{n} (n - i + j)$

$\sum_{i=1}^{n} (\sum_{j=1+i}^{n} (n - i) + \sum_{j=1+i}^{n} (j))$

$\sum_{i=1}^{n} ((n - i)(n - (i + 1) + 1) + \sum_{j=1}^{n} j - \sum_{j=1}^{i} j)$

$\sum_{i=1}^{n}(n^2 - 2ni + i^2 + \frac{n(n+1)}{2} - \frac{i(i+1)}{2})$
$\sum_{i=1}^{n}(3/2n^2 - 2ni + 1/2i^2 + 1/2n - 1/2i)$
$3/2n^3 + 1/2n^2 - 2n\sum_{i=1}^{n} i + 1/2\sum_{i=1}^{n} i^2 - 1/2\sum_{i=1}^{n} i$
$3/2n^3 + 1/2n^2 - n^2(n+1) + 1/2\frac{n(n+1)(2n+1)}{6} - 1/2\frac{n(n+1)}{2}$
$3/2n^3 + 1/2n^2 - n^3 - n^2 + 1/6n^3 + 1/4n^2 + 1/12n - 1/4n^2 - 1/4n$
$2/3n^3 - 1/2n^2 - 1/6n$

The function returns $2/3n^3 - 1/2n^2 - 1/6n$ and $Conundrum(n) = O(n^3)$.

5 Consider the following algorithm. Let $f(n)$ be the time complexity of this algorithm. Provide correct bounds for $O(f(n))$ and $\Omega(f(n))$, ideally converging on $\Theta(f(n))$.

```
for k = 1 to n:
        x = k
        while (x < n):
                print '*'
                x=2x
```

For $k = 1$, the inner loop will execute $\lceil log_2(n) \rceil$ times, since $x = 1$ doubles until it equals or exceeds n. Because the inner loop will execute fewer times for all $k < n$, $log_2(n) + 1$ is an upper bound. The outer loop executes n times, so $O(f(n)) = n\ log(n)$. For all $k < n$, the inner loop executes $\lceil log_2(\lceil n/k \rceil) \rceil$ times. Dropping the ceilings gives a lower bound, so $\Omega(f(n)) = n\ log(n)$. Conveniently, these converge, so $\Theta(f(n)) = n\ log(n)$.

6 Suppose the following algorithm is used to evaluate the polynomial $p(x) = a_b x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$.

```
p = a0;
xpower = 1;
for  i = 1 to n do
        xpower = x*xpower
        p = p + ai*xpower
```

6.1 How many multiplications are done in the worst case? How many additions?

On every iteration, one addition and two multiplications are performed, so in total, n additions and $2n$ multiplications.

6.2 How many multiplications are done on average?

The average case, best case, and worst case are all handled the same way, so $2n$ multiplications on average.

6.3 Can you improve this algorithm?

Iterate through the terms in reverse order. Start with a_n and multiply by x. On the next iteration, add a_{n-1} to the product and multiply the whole thing by x. By the last iteration, a_n will have been multiplied by x n times, and the final step will be to add a_0. This reduces the number of operations to n multiplications and n additions.

Another improvement would be to skip the second line in the loop whenever $a_n = 0$, sparing a multiplication and an addition.

7 Prove that the following algorithm for computing the maximum value in an array $A[1..n]$ is correct.

```
max(A)
        m = A[1]
        for i = 2 to n do
                if A[i] > m then m = A[i]
        return (m)
```

Prove by induction. In the base case, A only has one element, $A[1]$. Then $n = 1$, the loop never executes, and the function correctly returns $A[1]$. Now assume the function works for n elements. Then when given an array with size $n+1$, m will be set to the maximum of the first $n-1$ elements by the time the for loop executes for the final time. The if statement will compare the nth element to the previous maximum and set m to the new maximum if appropriate. Either way, the maximum element of the array will be returned.

8

8.1 Is $2^{n+1} = O(2^n)$?

Yes. $2^{n+1} = 2 * 2^n$, and $2 * 2^n < c * 2^n$ when $c > 2$.

8.2 Is $2^{2n} = O(2^n)$?

No. $2^{2n} = 2^n * 2^n$, and there is no $c > 2^n$ that would make $2^n * 2^n < c * 2^n$ true for all n, since eventually n will be greater than $log_2(c)$.

9 For each of the following pairs of functions, $f(n)$ is in $O(g(n))$, $\Omega(g(n))$, or $\Theta(g(n))$. Determine which relationships are correct and briefly explain why.

9.1 $f(n) = log(n^2) \quad g(n) = log(n) + 5$

$f(n) = \Theta(g(n))$
$log(n^2) = 2 * log(n)$, and any constant larger than 2 will make either function dominate the other. More formally, $\lim_{n\to\infty} \frac{2log(n)}{log(n)+5} = \lim_{n\to\infty} \frac{2}{n*ln(10)} / \frac{1}{n*ln(10)} = 2$.

9.2 $f(n) = \sqrt{n} \quad g(n) = log(n^2)$

$f(n) = \Omega(g(n))$
$\lim_{n\to\infty} \frac{\sqrt{n}}{2log(n)} = \lim_{n\to\infty} \frac{1}{2\sqrt{n}} / \frac{2}{n*ln(10)} = \lim_{n\to\infty} \frac{ln(10)\sqrt{n}}{4} = \infty$.

9.3 $f(n) = log^2(n) \quad g(n) = log(n)$

$f(n) = \Omega(g(n)) \quad \lim_{n\to\infty} \frac{log^2(n)}{log(n)} = \lim_{n\to\infty} log(n) = \infty$

9.4 $f(n) = n \quad g(n) = log^2(n)$

$f(n) = \Omega(g(n))$
$\lim_{n\to\infty} \frac{n}{log^2(n)} = \lim_{n\to\infty} \frac{ln(10)n}{2log(n)} = \lim_{n\to\infty} \frac{ln^2(10)n}{2} = \infty$

9.5 $f(n) = n\ log(n) + n \quad g(n) = log(n)$

$f(n) = \Omega(g(n))$
$\lim_{n\to\infty} \frac{n\ log(n)+n}{log(n)} = \lim_{n\to\infty}(log(n) + \frac{n}{ln(10)n} + 1)(ln(10)n)$
$= \lim_{n\to\infty} ln(10)n\ log(n) + n + ln(10)n = \infty$

9.6 $f(n) = 10 \quad g(n) = log(10)$

$f(n) = \Theta(g(n)) \quad \lim_{n\to\infty} \frac{10}{log(10)} = 10$

9.7 $f(n) = 2^n$
$g(n) = 10n^2$

$f(n) = \Omega g(n)$
$\lim_{n\to\infty} \frac{2^n}{10n^2} = \lim_{n\to\infty} \frac{2^n ln(2)}{20n} = \lim_{n\to\infty} \frac{2^n ln^2(2)}{20} = \infty$

9.8 $f(n) = 2^n \quad g(n) = 3^n$

$f(n) = O(g(n)) \quad \lim_{n\to\infty} \frac{2^n}{3^n} = \lim_{n\to\infty}(2/3)^n = 0$

10 For each of the following pairs of functions $f(n)$ and $g(n)$, determine whether $f(n) = O(g(n))$, $g(n) = O(f(n))$, or both.

10.1 $f(n) = (n^2 - n)/2 \quad g(n) = 6n$

$f(n) = \Omega(g(n))$
$\lim_{n\to\infty} \frac{1/2(n^2-n)}{6n} = \lim_{n\to\infty} \frac{2n-1}{12} = \infty$

10.2 $f(n) = n + 2\sqrt{n} \quad g(n) = n^2$

$f(n) = O(g(n))$
$\lim_{n\to\infty} \frac{n+2\sqrt{n}}{n^2} = \lim_{n\to\infty} \frac{1+1/\sqrt{n}}{2n} = \lim_{n\to\infty} \frac{(-1/2)n^{-3/2}}{2} = \lim_{n\to\infty} \frac{-1}{4n^{3/2}} = 0$

10.3 $f(n) = n\ log(n) \quad g(n) = n\sqrt{n}/2$

$f(n) = O(g(n))$
$\lim_{n\to\infty} \frac{n\ log(n)}{n\sqrt{n}/2} = \lim_{n\to\infty} \frac{log(n)+n/(nln(10))}{(3/4)\sqrt{n}} = \lim_{n\to\infty} \frac{1/(nln(10))}{(3/8)n^{-1/2}} = \lim_{n\to\infty} \frac{8}{3ln(10)} \frac{\sqrt{n}}{n} =$
$\lim_{n\to\infty} \frac{8}{3ln(10)} \frac{1}{\sqrt{n}} = 0$

10.4 $f(n) = n + log(n) \quad g(n) = \sqrt{n}$

$f(n) = \Omega(g(n))$
$\lim_{n\to\infty} \frac{n+log(n)}{\sqrt{n}} = \lim_{n\to\infty} \frac{1+\frac{1}{nln(10)}}{\frac{1}{2\sqrt{n}}} = 2\sqrt{n} + \frac{2\sqrt{n}}{nln(10)} = \infty$

10.5 $f(n) = 2(log(n)^2) \quad g(n) = log(n) + 1$

$f(n) = \Omega g(n)$
$\lim_{n\to\infty} \frac{2(log(n))^2}{log(n)+1} = \lim_{n\to\infty} \frac{4log(n)/(nln(10))}{1/(nln(10))} = \lim_{n\to\infty} 4log(n) = \infty$

10.6 $f(n) = 4n\ log(n) + n \quad g(n) = (n^2 - n)/2$

$f(n) = O(g(n))$
$\lim_{n\to\infty} \frac{4n\ log(n)+n}{(n^2-n)/2} = \lim_{n\to\infty} \frac{4log(n)+4/ln(10)+1}{n-1/2} = \lim_{n\to\infty} 4/(nln(10)) = 0$

11 For each of the following functions, which of the following asymptotic bounds hold for $f(n) :\ O(g(n)),\ \Omega(g(n))$, or $\Theta(g(n))$?

11.1 $f(n) = 3n^2 \quad g(n) = n^2$

$f(n) = \Theta(g(n))$
$\lim_{n\to\infty} \frac{3n^2}{n^2} = 3$

11.2 $f(n) = 2n^4 - 3n^2 + 7 \quad g(n) = n^5$

$f(n) = \Omega(g(n))$
$\lim_{n\to\infty} \frac{2n^4-3n^2+7}{n^5} = \lim_{n\to\infty} \frac{8n^3-6n}{5n^4} = \lim_{n\to\infty} \frac{24n^2-6}{20n^3} = \lim_{n\to\infty} \frac{48n}{60n^2} = 0$

11.3 $f(n) = log(n) \quad g(n) = log(n) + 1/n$

$f(n) = \Theta(g(n))$
$\lim_{n\to\infty} \frac{log(n)}{log(n)+1/n} = \lim_{n\to\infty} \frac{1/(nln(10))}{1/(nln(10))-1/n^2} = \lim_{n\to\infty} \frac{1}{1-ln(10)/n} = 1$

11.4 $f(n) = 2^{klog(n)} \quad g(n) = n^k$

$f(n) = \Omega(g(n))$

$\lim_{n\to\infty} \frac{2^{klog(n)}}{n^k} = \lim_{n\to\infty}(\frac{2^{log(n)}}{n})^k = \lim_{n\to\infty}(\frac{2^{\frac{log_2(n)}{log_2(10)}}}{n})^k = \lim_{n\to\infty}(\frac{(2^{log_2(n)})^{\frac{1}{log_2(10)}}}{n})^k =$
$\lim_{n\to\infty}(\frac{n^{\frac{1}{log_2(10)}}}{n})^k = \lim_{n\to\infty}(n^{\frac{1}{log_2(10)}+1})^k = \infty$

11.5 $f(n) = 2^n \quad g(n) = 2^{2n}$

$f(n) = O(g(n))$
$\lim_{n\to\infty} \frac{2^n}{2^{2n}} = \lim_{n\to\infty} \frac{1}{2^n} = 0$

12 Prove that $n^3 - 3n^2 - n + 1 = \Theta(n^3)$.

$\lim_{n\to\infty} \frac{n^3-3n^2-n+1}{n^3} = \lim_{n\to\infty} \frac{3n^2-6n-1}{3n^2} = \lim_{n\to\infty} \frac{6n-6}{6n} = \lim_{n\to\infty} \frac{6}{6} = 1$

Or, for any $c > 0$, $cn^3 > n^3 - 3n^2 - n + 1$ for any $c > 4$, $c(n^3 - 3n^2 - n + 1) > n^3$ for sufficiently large n. Therefore, $n^3 - 3n^2 - n + 1 = \Theta(n^3)$.

13 Prove that $n^2 = O(2^n)$.

$\lim_{n\to\infty} \frac{n^2}{2^n} = \lim_{n\to\infty} \frac{2n}{2^n * ln(2)} = \lim_{n\to\infty} \frac{2}{2^n * ln(2)^2} = 0$

14 Prove or disprove: $\Theta(n^2) = \Theta(n^2 + 1)$.

$n^2 = O(n^2 + 1)$ and $n^2 + 1 = \Omega(n^2)$ because $n^2 + 1$ is always greater than n^2. $n^2 + 1 = O(n^2)$ and $n^2 = \Omega(n^2 + 1)$ because for $c = 2$, cn^2 is always greater than $n^2 + 1$. Therefore, $\Theta(n^2) = \Theta(n^2 + 1)$.

15 Suppose you have algorithms with the five running times listed below. How much slower do each of these algorithms get when you (a) double the input size, or (b) increase the input size by one?

15.1 n^2

(a) Four times longer. $(2n)^2/n^2 = 4n^2/n^2 = 4$
(b) Takes an extra $2n+1$ units of time. $(n+1)^2 - n^2 = n^2+2n+1-n^2 = 2n+1$

15.2 n^3

(a) Eight times longer.
$(2n)^3/n^3 = 8n^3/n^3 = 8$

(b) Takes an extra $3n^2 + 3n + 1$ units of time.
$(n+1)^3 - n^3 = n^3 + 3n^2 + 3n + 1 - n^3 = 3n^2 + 3n + 1$

15.3 $100n^2$

(a) Four times longer.
$100(2n)^2/(100n^2) = 100 * 4n^2/(100n^2) = 4$

(b) Takes an extra $200n + 100$ units of time.
$100(n+1)^2 - 100n^2 = 100(n^2 + 2n + 1) - 100n^2 = 200n + 100$

15.4 $n\ log(n)$

(a) Just over twice as long.
$(2n\ log(2n))/(n\ log(n)) = 2n(log(2)+log(n))/(n\ log(n)) = 2n\ log(n)/(n\ log(n)) + log(n)/(n\ log(n)) = 2 + 1/n$

(b) Takes just a tiny bit more than $log(n)$ units of time longer.
$(n+1)log(n+1) - n\ log(n) = n\ log(n+1) + log(n+1) - n\ log(n)$

15.5 2^n

(a) Takes 2^n times longer. $2^{2n}/2^n = 2^n 2^n/2^n = 2^n$
(b) Twice as long. $2^{n+1}/2^n = 2 * 2^n/2^n = 2$

16 Suppose you have algorithms with the six running times listed below. Suppose you have a computer that can perform 10^{10} operations per second. For each algorithm, what is the largest input size n that you can complete within an hour?

Number of operations per hour: $10^{10}\frac{ops}{sec} * 3600\frac{secs}{hour} = 3.6 * 10^{13}$

16.1 n^2

$n^2 = 3.6 * 10^{13}$
$n = 6 * 10^6$

16.2 n^3

$n^3 = 3.6 * 10^{13}$
$n = 33019$

16.3 $100n^2$

$100n^2 = 3.6 * 10^{13}$
$n = 6 * 10^5$

16.4 $n\ log(n)$

$n\ log(n) = 3.6 * 10^{13}$
$n = 1.29095 * 10^{12}$
(WolframAlpha says so)

16.5 2^n

$2^n = 3.6 * 10^{13}$
$n = log_2(3.6 * 10^{13})$
$n = 45$

16.6 2^{2^n}

$2^{2^n} = 3.6 * 10^{13}$
$2^n = log_2(3.6 * 10^{13})$
$n = log_2(log_2(3.6 * 10^{13}))$
$n = 5$

17 For each of the following pairs of functions, give an appropriate positive constant c such that $f(n) \leq cg(n)$ for all $n > 1$.

17.1 $f(n) = n^2 + n + 1 \quad g(n) = 2n^3$

$n^2 + n + 1 \leq 2cn^3$
$c \geq \frac{n^2+n+1}{2n^3}$
$c \geq \frac{1}{2n} + \frac{1}{2n^2} + \frac{1}{2n^3}$
As n grows, the right side of the inequality shrinks, so its maximum occurs at n's minimum. The inequality must hold for $n > 1$ where n is an integer, so n's minimum value is two.
$c \geq \frac{1}{4} + \frac{1}{8} + \frac{1}{16}$
The inequality holds for all positive integers c, so let $c = 1$.

17.2 $f(n) = n\sqrt{n} + n^2 \quad g(n) = n^2$

$n\sqrt{n} + n^2 \leq cn^2$
$c \geq \frac{n\sqrt{n}+n^2}{n^2}$
$c \geq \frac{\sqrt{n}}{n} + 1$
For all positive integers n, $\frac{\sqrt{n}}{n} \leq 1$, and for all integers greater than one, $\frac{\sqrt{n}}{n} < 1$.
$c = 1 + 1 > \frac{\sqrt{n}}{n} + 1$
$c = 2$

17.3 $f(n) = n^2 - n + 1 \quad g(n) = n^2/2$

$n^2 - n + 1 \leq cn^2/2$
$c \geq 2 - \frac{2}{n} + \frac{2}{n^2}$
As n grows, the right side of the inequality shrinks, so its maximum occurs when $n = 2$.
$c \geq 2 - 1 + \frac{1}{2}$
$c = 2$

18 Prove that if $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$.

By definition, $f_1(n) \leq c_1(g_1(n))$ and $f_2(n) \leq c_2(g_2(n))$ for some positive constants c_1 and c_2 and all sufficiently large n, so $f_1(n) + f_2(n) \leq c_1(g_1(n)) + c_2(g_2(n))$. Let $c = max\{c_1, c_2\}$. Then, $f_1(n) + f_2(n) \leq c_1(g_1(n)) + c_2(g_2(n)) \leq c(g_1(n) + g_2(n))$. Therefore, $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$.

19 Prove that if $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$, then $f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$.

By definition, $f_1(n) \geq c_1(g_1(n))$ and $f_2(n) \geq c_2(g_2(n))$, so $f_1(n) + f_2(n) \geq c_1(g_1(n)) + c_2(g_2(n))$ where c_1 and c_2 are positive constants. Let $c = min\{c_1, c_2\}$. Then, $f_1(n) + f_2(n) \geq c_1(g_1(n)) + c_2(g_2(n)) \geq c(g_1(n) + g_2(n))$, so $f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$.

20 Prove that if $f_1(n) = O(g_1(n))$ and $f_2 = O(g_2(n))$, then $f_1(n) * f_2(n) = O(g_1(n) * g_2(n))$.

By definition, $f_1(n) \leq c_1 g_1(n)$ and $f_2(n) \leq c_2 g_2(n)$ for some positive constants c_1 and c_2. Then, because all values involved are positive, we can multiply to get $f_1(n) * f_2(n) \leq c_1 g_1(n) * c_2 g_2(n)$. Let $c = c_1 * c_2$. Then $f_1(n) * f_2(n) \leq c * g_1(n) * g_2(n)$, and $f_1(n) * f_2(n) = O(g_1(n) * g_2(n))$.

21 Prove that for all $k \geq 0$ and all sets of real constants $\{a_k, a_{k+1}, ..., a_1, a_0\}$, $a_k n^k + a_{k-1} n^{k-1} + ... + a_1 n + a_0 = O(n^k)$.

It must be shown that there exists some positive constant c such that $a_k n^k + a_{k-1} n^{k-1} + ... + a_1 n + a_0 \leq c(n^k)$ for all sufficiently large n. Dividing both sides by n^k, $c \geq a_k + \frac{a_{k-1}}{n} + ... + \frac{a_1}{n^{k-1}} + \frac{a_0}{n^k}$. Each term except the first shrinks as n grows, so to prove the strongest case, let $n = 1$. Then, because c must be positive, $c = max\{1, a_k + a_{k-1} + ... + a_1 + a_0\}$ and $a_k n^k + a_{k-1} n^{k-1} + ... + a_1 n + a_0 = O(n^k)$.

22 Show that for any real constants a and b, $b > 0$, $(n + a)^b = \Theta(n^b)$.

To prove the theta, it must be shown that $(n+a)^b = O(n^b)$ and $(n+a)^b = \Omega(n^b)$.

First, find c_1 such that $(n+a)^b \leq c_1(n^b)$, or $c_1 \geq \frac{(n+a)^b}{n^b}$. According to the binomial theorem, $(n+a)^b$ expands to $\binom{b}{0}n^b a^0 + \binom{b}{1}n^{b-1}a^1 + ... + \binom{b}{b-1}na^{b-1} + \binom{b}{b}n^0 a^b$. Each binomial coefficient is a positive integer, represent them by x_i. Dividing each term by n^b, this gives $c_1 \geq \frac{x_0 n^b a^0}{n^b} + \frac{x_1 n^{b-1} a^1}{n^b} + ... + \frac{x_{b-1} n^1 a^{b-1}}{n^b} + \frac{x_b n^0 a^b}{n^b}$, or $c_1 \geq x_0 + \frac{x_1 a^1}{n} + ... + \frac{x_{b-1} a^{b-1}}{n^{b-1}} + \frac{x_b a^b}{n^b}$. The value of the sum decreases as n grows. To make the inequality hold even when $n = 1$, let $c_0 = x_0 + x_1 a + ... + x_{b-1} a^{b-1} + x_b a^b$. Or, if the sum turns out to be negative, let $c_1 = 1$, which will still hold, since it is larger than the calculated c_1, but is a positive constant as the definition requires.

Now, find c_2 such that $(n+a)^b \geq c_2(n^b)$. If $a \geq 0$, this is trivial, since $n+a \geq n$, so $(n+a)^b \geq n^b$ and $c_2 = 1$. If $a < 0$, let $n = -2a$, which will of course be a positive number. According to the definition, the inequality must hold only for all $n > n_0$ for some constant n_0, in this case $-2a$. So, we must find c_2 such that $(-2a + a)^b \geq c_2(-2a)^b$. Simplifying, $(\frac{-a}{-2a})^b \geq c_2$, or $\frac{1}{2^b} \geq c_2$. Let $c_2 = \frac{1}{2^b}$, and we have proven that there exist constants c_1 and c_2 such that $(n+a)^b \leq c_1(n^b)$ and $(n+a)^b \geq c_2(n^b)$, so $(n+a)^b = \Theta(n^b)$.

23 List the functions below from the lowest to the highest order. If any two or more are of the same order, indicate which.

$lg(lg(n)) \ll ln(n) = lg(n) \ll lg(n)^2 \ll \sqrt{n} \ll n \ll nlg(n) \ll n^{1+\epsilon} \ll n^2 = n^2 + lg(n) \ll n^3 \ll n - n^c + 7n^5 \ll e^n = 2^{n-1} = 2^n \ll n!$

24 List the functions below from the lowest to the highest order. If any two or more are of the same order, indicate which.

All credit to WolframAlpha.
$n^\pi \ll \binom{n}{n-4} \ll \binom{n}{5} \ll n^4\binom{n}{n-4} \ll n^{5log^2(n)} \ll 2^{log^4(n)} \ll \sqrt{2^{\sqrt{n}}} \ll \pi^n$

25 List the functions below from the lowest to the highest order. If any two or more are of the same order, indicate which.

$2^{log(n^2)} \ll \binom{n}{n-4} \ll log(n)^{log(n)} \ll n^{log(n)^2} \ll 2^{log^4(n)} \ll n! \ll n^n = \sum_{i=1}^{n} i^i$

26 List the functions below from the lowest to the highest order. If any two or more are of the same order, indicate which.

$(1/3)^n \ll 6 \ll log(log(n)) \ll log(n) = ln(n) \ll log^2(n) \ll n^{1/3} + log(n) \ll \sqrt{n} \ll \frac{n}{log(n)} \ll n \ll n\ log(n) \ll n^2 = n^2 + log(n) \ll n^3 \ll n - n^3 + 7n^5 \ll (3/2)^n \ll 2^n \ll n!$

27 Find two functions $f(n)$ and $g(n)$ that satisfy the following relationship. If no such f and g exist, write 'None.'

27.1 $f(n) = o(g(n))$ and $f(n) \neq \Theta(g(n))$

$f(n) = n \quad g(n) = n^2$

27.2 $f(n) = \Theta(g(n))$ and $f(n) \neq o(g(n))$

$f(n) = g(n) = n$

27.3 $f(n) = \Theta(g(n))$ and $f(n) \neq O(g(n))$

None

27.4 $f(n) = \Omega(g(n))$ and $f(n) \neq O(g(n))$

$f(n) = n^2 \quad g(n) = n$

28 True or False?

28.1 $2n^2 + 1 = O(n^2)$

True

28.2 $\sqrt{n} = O(log(n))$

False

28.3 $log(n) = O(\sqrt{n})$

True

28.4 $n^2(1+\sqrt{n}) = O(n^2 log(n))$

False

28.5 $3n^2 + \sqrt{n} = O(n^2)$

True

28.6 $\sqrt{n} log(n) = O(n)$

True

28.7 $log(n) = O(n^{-1/2})$

False

29 For each of the following pairs of functions $f(n)$ and $g(n)$, state whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \Theta(g(n))$, or none of the above.

29.1 $f(n) = n^2 + 3n + 4,\ g(n) = 6n + 7$

$f(n) = \Omega(g(n))$

29.2 $f(n) = n\sqrt{n},\ g(n) = n^2 - n$

$f(n) = O(g(n))$

29.3 $f(n) = 2^n - n^2,\ g(n) = n^4 + n^2$

$f(n) = \Omega(g(n))$

30 For each of these questions, answer yes or no and briefly explain your answer.

30.1 If an algorithm takes $O(n^2)$ worst-case time, is it possible that it takes $O(n)$ on some inputs?

Yes. For example, consider an algorithm that traverses a list. In the worst case, it traverses the entire list again for each item in the list. It is possible that there is a best case where if some condition is met, it simply moves on to the next element. In this case, the algorithm would take $O(n)$ time.

30.2 If an algorithm takes $O(n^2)$ worst-case time, is it possible that it takes $O(n)$ on all inputs?

Technically yes. Big oh simply gives an upper bound. This algorithm also takes $O(n^3)$, $O(n^4)$, etc.

30.3 If an algorithm takes $\Theta(n^2)$ worst-case time, is it possible that it takes $O(n)$ on some inputs?

Yes, as illustrated in the example in the first part of this problem.

30.4 If an algorithm takes $\Theta(n^2)$ worst-case time, is it possible that it takes $O(n)$ on all inputs?

No. If it did, there would be no c such that the algorithm takes longer than cn^2 time, as required by a $\Theta(n^2)$ algorithm, which gives a lower bound as well as an upper bound.

30.5 Is the function $f(n) = \Theta(n^2)$, where $f(n) = 100n^2$ for even n and $f(n) = 20n^2 - n\ log_2(n)$ for odd n?

Yes. In both cases, the function is dominated by the n^2 term, which is clearly in $\Theta(n^2)$.

31 For each of the following, answer yes, no, or can't tell. Explain your reasoning.

31.1 Is $3^n = O(2^n)$?

No. For there to be a c such that $3^n \leq c2^n$, c would have to be larger than $(3/2)^n$ for all n, which is impossible for a constant.

31.2 Is $log(3^n) = O(log(2^n))$?

Yes. $log(3^n) = n\ log(3)$ and $log(2^n) = n\ log(2)$. For any $c > log(3)/log(2)$, $log(3^n) < c\ flog(2^n)$.

31.3 Is $3^n = \Omega(2^n)$?

Yes. $3^n \geq 2^n$ for all $n \geq 1$.

31.4 Is $log(3^n) = \Omega(log(2^n))$?

Yes. For all $c > log(2)/log(3)$, $c\ log(3^n) > log(2^n)$.

32 For each of the following expressions $f(n)$, find a simple $g(n)$ such that $f(n) = \Theta(g(n))$.

32.1 $f(n) = \sum_{i=1}^{n} \frac{1}{i}$

$g(n) = 1$

32.2 $f(n) = \sum_{i=1}^{n} \left\lceil \frac{1}{i} \right\rceil$

$g(n) = n$

32.3 $f(n) = \sum_{i=1}^{n} log(i)$

$g(n) = n\ log(n)$

32.4 $f(n) = log(n!)$

$g(n) = n\ log(n)$

33 Place the following functions into increasing order: $f_1(n) = n^2 log_2(n)$, $f_2(n) = n(log_2(n))^2$, $f_3(n) = \sum_{i=0}^{n} 2^i$, $f_4(n) = log_2(\sum_{i=0}^{n} 2^i)$.

$f_4(n) << f_2(n) << f_1(n) << f_3(n)$
$(n << n\ log^2(n) << n^2 log(n) << 2^n)$

34 Which of the following are true?

34.1 $\sum_{i=1}^{n} 3^i = \Theta(3^{n-1})$

True. Using the identity $\sum_{i=0}^{n-1} a^i = \frac{1-a^n}{1-a}$, we can see that $\sum_{i=1}^{n} 3^i$ is on the order of 3^n, which differs from 3^{n-1} by a constant factor.

34.2 $\sum_{i=1}^{n} 3^i = \Theta(3^n)$

True.

34.3 $\sum_{i=1}^{n} 3^i = \Theta(3^{n+1})$

True.

35 For each of the following functions f, find a simple function g such that $f(n) = \Theta(g(n))$.

35.1 $f_1(n) = (1000)2^n + 4^n$

$g(n) = 4^n$

35.2 $f_2(n) = n + n\ log(n) + \sqrt{n}$

$g(n) = n\ log(n)$

35.3 $f_3(n) = log(n^{20}) + (log(n))^{10}$

$g(n) = (log(n))^{10}$

35.4 $f_4(n) = (0.99)^n + n^{100}$

$g(n) = n^{100}$

36 For each pair of expressions (A, B) below, indicate whether A is O, o, Ω, ω, or Θ of B.

36.1 $(n^{100}, 2^n)$

$A = o(B), O(B)$

36.2 $((log(n))^{12}, \sqrt{n})$

$A = \Omega(B), \omega(B)$

36.3 $(\sqrt{n}, n^{cos(\pi n/8)})$

None

36.4 $(10^n, 100^n)$

$A = o(B), O(B)$

36.5 $(n^{lg(n)}, (lg(n))^n)$

$A = o(B), O(B)$

36.6 $(lg(n!), nlg(n))$

$A = o(B), O(B)$

37 Find an expression for the sum of the ith row of the following triangle, and prove its correctness. Each entry is the sum of the three entries directly above it.

If the first row is considered row 0, the sum of the ith row is 3^i. Each element of a row contributes its value three times over in the following row. In other words, each element is the sum of (at most) three values from the previous row. So, the sum of row i is three times the sum of the elements in row $i-1$. Since row 0's sum is 1, the sum of row 1 is 3, the sum of row 2 is 9, and the sum of row i is 3^i.

38 Assume that Christmas has n days. Exactly how many presents did my "true love" send to me?

$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

39 An unsorted array of size n contains distinct integers between 1 and $n+1$, with one element missing. Give an $O(n)$ algorithm to find the missing integer, without using any extra space.

Iterate through the array, keeping a running total by adding the previous total to the current element. At the end, subtract the final element from $\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$, which will give the missing integer.

40 Consider the following code fragment. Let $T(n)$ denote the number of times 'foobar' is printed as a function of n.

```
for i=1 to n do
        for j=1 to 2*i do
                output ''foobar''
```

40.1 Express $T(n)$ as a summation.

$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{2*i} 1$

40.2 Simplify the summation.

$\sum_{i=1}^{n} \sum_{j=1}^{2*i} 1$
$\sum_{i=1}^{n} 2i$
$2 \sum_{i=1}^{n} i$
$2 \frac{n(n+1)}{2}$
$n^2 + n$

41 Consider the following code fragment. Assume n is even. Let $T(n)$ denote the number of times 'foobar' is printed as a function of n.

```
for i=1 to n/2 do
        for j=1 to n-i do
                for k=1 to j do
                        output 'foobar'
```

41.1 Express $T(n)$ as three nested summations.

$T(n) = \sum_{i=1}^{n/2} \sum_{j=1}^{n-i} \sum_{k=1}^{j} 1$

41.2 Simplify the summation.

$\sum_{i=1}^{n/2} \sum_{j=1}^{n-i} \sum_{k=1}^{j} 1$
$\sum_{i=1}^{n/2} \sum_{j=1}^{n-i} j$
$\sum_{i=1}^{n/2} \frac{(n-i)(n-i+1)}{2}$
$1/2(\sum_{i=1}^{n/2} n^2 - 2ni + i^2 + n - i$
$1/2(\frac{n}{2} * n^2 - 2n \sum_{i=1}^{n/2} i + \sum_{i=1}^{n/2} i^2 + \frac{n}{2} * n - \sum_{i=1}^{n/2} i$
$1/2(\frac{n^3}{2} - 2n\frac{n/2(n/2+1)}{2} + \frac{n/2(n/2+1)(n+1)}{6} + \frac{n^2}{2} - \frac{n/2(n/2+1)}{2})$
$\frac{n^3}{4} - \frac{n^3}{8} - \frac{n^2}{4} + \frac{n^3}{48} + \frac{n^2}{48} + \frac{n^2}{24} + \frac{n}{24} + \frac{n^2}{4} - \frac{n^2}{16} - \frac{n}{8}$
$\frac{7n^3}{48} - \frac{n}{12}$

42 When you first learned to multiply numbers, you were told that $x \times y$ means adding x a total of y times, so $5 \times 4 = 5+5+5+5 = 20$. What is the time complexity of multiplying two n-digit numbers in base b using the repeated addition method, as a function of n and b? Assume that single-digit by single-digit addition or multiplication takes $O(1)$ time.

As an example, consider multiplying three digits in base 10, where the possible inputs range from 100 to 999. Each full addition involves between n and $2n$ digits, $n = 3$ at first (100+100 or 999+999) and either $2n = 6$ or $2n - 1 = 5$ by the last addition (9900+100=10000 or 997002+999=998001). The number of full additions equals the input value, which must be between $b^{n-1} = 100$ and $b^n = 1000$, which differ only by the constant $b = 10$.

The total number of operations equals the number of full additions, which must be between b^{n-1} and b^n, times the number of single-digit additions at each step, n to $2n$. This value ranges from nb^{n-1} to $2nb^n$. Ignoring the constants, the time complexity is on the order of nb^n.

43 In grade school, you learned to multiply long numbers on a digit-by-digit basis, so that $127 \times 211 = 127 \times 1 + 127 \times 10 + 127 \times 200 = 26,797$. Analyze the time complexity of multiplying two n-digit numbers with this method as a function of n, assuming constant base size. Assume that single-digit by single-digit addition or multiplication takes $O(1)$ time.

The grade-school algorithm involves two basic steps. First, multiply the first n digit number by a single digit from the second number. Repeat for each of the n digits. Second, add all n results together. The first part involves up to $2n$ operations per digit, one for the single-digit by single-digit multiplication and another for a potentially carried value. For n digits, this gives $2n^2$ operations. The second part involves n additions of up to $2n$ digits. Again, this gives $2n^2$ operations. Together, this multiplication involves up to $4n^2$ operations, making it $O(n^2)$.

44 Prove the following identities on logarithms.

44.1 $log_a(xy) = log_a(x) + log_a(y)$

By the definition of logarithms, there exist some numbers z, z', and z'' that satisfy $log_a(xy) = z$, $log_a(x) = z'$, and $log_a(y) = z''$, which is equivalent to $a^z = xy$, $a^{z'} = x$, and $a^{z''} = y$.

Because of the properties of exponents, $xy = a^{z'}a^{z''} = a^{z'+z''}$. We know that $a^z = xy$, so $a^z = a^{z'+z''}$. Taking log_a of both sides, $z = z' + z''$, so by substitution, $log_a(xy) = log_a(x) + log_a(y)$.

44.2 $log_a(x^y) = ylog_a(x)$

Let $log_a(x) = z$. Then $a^z = x$. Substituting into $log_a(x^y)$, $log_a((a^z)^y)$. By the properties of exponents, this equals $log_a(a^{yz})$, or yz. Substituting again, $yz = ylog_a(x)$, so $log_a(x^y) = ylog_a(x)$.

44.3 $log_a(x) = \frac{log_b(x)}{log_b(a)}$

Let $log_a(x) = z$, so $a^z = x$. Taking log_b of both sides, $log_b(a^z) = log_b(x)$. By the property proved above, $log_b(a^z) = zlog_b(a)$, so $zlog_b(a) = log_b(x)$. Dividing both sides by $log_b(a)$, $z = \frac{log_b(x)}{log_b(a)}$. Substituting for z, $log_a(x) = \frac{log_b(x)}{log_b(a)}$.

44.4 $x^{log_b(y)} = y^{log_b(x)}$

Let $z = x^{log_b(y)}$. To prove the inequality, it must be shown that $z = y^{log_b(x)}$. First, take log_b of both sides, giving $log_b(z) = log_b(x^{log_b(y)})$. By the second identity proven above, $log_b(z) = log_b(y)log_b(x)$. Dividing both sides by $log_b(y)$, $\frac{log_b(z)}{log_b(y)} = log_b(x)$. By the third identity above, this simplifies to $log_y(z) = log_b(x)$. Taking y to the power of each side, we have $y^{log_y(z)} = y^{log_b(x)}$, which simplifies to $z = y^{log_b(x)}$. Therefore, $x^{log_b(y)} = y^{log_b(x)}$.

45 Show that $\lceil lg(n+1) \rceil = \lfloor lg(n) \rfloor + 1$

In any case where $x \leq lg(n) < lg(n+1) \leq x+1$ for some integer x, the relationship will clearly hold. By contradiction, assume there exists some number n such that $lg(n) < x < lg(n+1)$ for some integer x. Then $2^{lg(n)} < 2^x < 2^{lg(n+1)}$, or $n < 2^x < n+1$. But an integer to the power of another integer will always be another integer, which violates the inequality, since it requires 2^x to be strictly greater than one integer and strictly less than the next greatest one. This is a contradiction, so the relationship holds in all cases.

46 Prove that the binary representation of $n \geq 1$ has $\lfloor lg_2 n \rfloor + 1$ bits.

Proof by induction. In the base case, $n = 1$. The binary representation is the same, with a single digit. $\lfloor lg_2 1 \rfloor + 1 = 0 + 1 = 1$.

For the inductive step, assume that the binary representation of n has $\lfloor lg_2 n \rfloor + 1$ digits. It must be shown that the binary representation of $n+1$ has $\lfloor lg_2(n+1) \rfloor +$ 1 digits. There are two possibilities: either n is $2^x - 1$ for some integer x or it is not. If it is, $n+1$'s binary representation will have an additional bit, otherwise the number of bits will be the same.

In the first case, n has $\lfloor lg_2(2^x - 1) \rfloor + 1$ bits. For any $x > 1$, $\lfloor lg_2(2^x - 1) \rfloor = x - 1$, so the binary representation will have x bits. According to the formula $n+1$, or 2^x, will have $\lfloor lg_2 2^x \rfloor + 1 = \lfloor x \rfloor + 1 = x + 1$ bits, as expected. In the second case, n will have $\lfloor lg_2 n \rfloor + 1$ bits, and $n+1$ will have $\lfloor lg_2(n+1) \rfloor + 1$. Since n is not $2^x - 1$ for any x, $\lfloor lg_2 n \rfloor = \lfloor lg_2(n+1) \rfloor$, so the number of bits will be the same.

47 There is a comparison-based sorting algorithm that runs in $O(n\ log\sqrt{n})$ time. Given the existence of an $\Omega(n\ log(n))$ lower bound for sorting, how can this be possible?

$n\ log(\sqrt{n}) = n\ log(n^{1/2}) = \frac{1}{2} n\ log(n)$
For $c > 2$, $\frac{1}{2} cn\ log(n) > n\ log(n)$, so $n\ log(\sqrt{n}) = \Omega(n\ log(n))$.

48 You are given a set S of n numbers. You must pick a subset S' of k numbers from S such that the probability of each element of S occurring in S' is equal. You may make only one pass over the numbers. What if n is unknown?

First, allocate an array of size k to hold S'. As you iterate over S, you will make at least one pass through S'. During the first pass, you will add the numbers you encounter in S to S' with probability $1/1$. On the second pass through S', you will switch the existing number to the new number with probability $1/2$. On the third pass, switch with probability $1/3$ and on pass N with probability $1/N$. To account for the case where k does not divide n evenly (which would favor the numbers placed at the end of the array, sparing them the final round of possible replacement) S' should either be shuffled after each pass or the order in which it is iterated through should be randomized.

49 We have 1,000 data items to store on 1,000 nodes. Each node can store copies of exactly three different items. Propose a replication scheme to minimize data loss as nodes fail. What is the expected number of data entries that get lost when three random nodes fail?

To minimize data loss, the 1,000 data items need to be spread out. In the absence of information about neighboring nodes being more likely to fail simultaneously, store them (1,2,3) (2,3,4) ... (998,999,1) (999,1,2). Any single node failure doesn't require a response, but when two nodes that hold the same item fail, the final surviving copy should be replicated. At first, that will be easy: find a node that contains an item that also has two surviving copies and replace it with a copy of the now-precious item. Once no item has more than two copies, there is nothing else to do, since replicating any item erases the only backup of another item.

The probability of losing a data entry when three random nodes fail is equal to the probability of three consecutive nodes failing. After the first node fails, either the node before it or the node after it must fail as well, two possibilities out of 999. If those two fail, either the node before those two or the node after them must fail, two out of 998. Of course, the order in which they fail does not matter. $\frac{2}{999} \times \frac{2}{998} = 0.00000401202$. In all other cases, no data will be lost. So, the expected number of data entries lost when three random nodes fail is $0 \times (1 - 0.00000401202) + 1 \times 0.00000401202 = 0.00000401202$.

Note: This answer doesn't feel complete. Am I missing something?

50 Consider the following algorithm to find the minimum element in an array of numbers $A[0, ..., n]$. One extra variable tmp is allocated to hold the current minimum value. Start from $A[0]$; tmp is compared against $A[1], A[2], ..., A[N]$ in order. When $A[i] < tmp$, $tmp = A[i]$. What is the expected number of times that the assignment operation $tmp = A[i]$ is performed?

The probability of the assignment operation occurring after comparing the nth number is equal to the probability that $A[N]$ is the greatest encountered so far, which equals $\frac{1}{n+1}$. For example, the probability of switching on the first comparison with $A[1]$ equals 1/2. The probability of switching after comparing the next number, $A[2]$, equals the probability that $A[2]$ is the largest of the three: 1/3. The total expected number of switches is $\sum_{i=2}^{N} \frac{1}{i}$.

Actually, it is slightly lower, because in the case of the two numbers being equal, no switch is made. Without knowing how likely that is to happen, it makes the most sense to leave the chance of one number being less than another at 1/2.

51 You are given ten bags of gold coins. Nine bags contain coins that each weigh 10 grams. One bag contains all false coins that weigh 1 gram less. You must identify this bag in just one weighing. You have a digital balance that reports the weight of what is placed on it.

Place one coin from the first bag, two from the second, three from the third, and so on on the scale. There will be 55 coins total. If the weight is 549, there is one false coin, which means the first bag is the culprit. If it is 548, it is the second bag. If it is 540, the tenth bag is the problem.

52 You have eight balls all of the same size. Seven of them weigh the same, and one of them weighs slightly more. How can you find the ball that is heavier by using a balance and only two weighings?

Put three on one side of the scale and another three on the other. If one side is heavier, the heavy ball is in that group. Choose two from that group and weigh them. If one is heavier, that's the one. Otherwise, the one from that group that didn't go on the scale is it. If neither group of three was heavier, one of the two that weren't weighed is heavier. Weigh them to find out.

53 Suppose we start with n companies that eventually merge into one big company. How many different ways are there for them to merge?

I made two assumptions here. The first is that order doesn't matter. Companies A and B merging into A' and companies C and D merging into C', then A' and C' merging is the same 'way' for our purposes as if C' was formed before A'. The second is that it any number of companies can merge at once. I wasn't able to find a closed form formula, but a recursive solution is as follows.

The base case is simple, as they usually are. There is exactly one way for two companies to merge, so $mrg(2) = 1$. There are two possibilities when three companies are merging: either all three merge at once, or two companies merge and then that larger company merges with the third. There is exactly one way for three companies to merge together at once, and there are $\binom{3}{2} = 3$ ways for two companies out of three to merge. Then, there are two companies left and we have our base case. So, $mrg(3) = \binom{3}{3} + \binom{3}{2}mrg(2) = 4$.

The pattern continues the same way:
$mrg(4) = \binom{4}{4} + \binom{4}{3}mrg(2) + \binom{4}{2}mrg(3) = 1 + 4 * 1 + 6 * 4 = 29$

$mrg(n) = \binom{n}{n} + \binom{n}{n-1}mrg(2) + \binom{n}{n-2}mrg(3) + ... + \binom{n}{2}mrg(n-1)$

In case my second assumption was incorrect, the problem is much simpler. If only two companies can merge at a time, $mrg(n) = \binom{n}{2}\binom{n-1}{2}...\binom{2}{2} = \Pi_{i=2}^{n}\binom{i}{2}$.

54 Six pirates must divide $300 among themselves. The division is to proceed as follows. The senior pirate proposes a way to divide the money. Then the pirates vote. If the senior pirate gets at least half the votes he wins, and that division remains. If he doesn't, he is killed and then the next senior-most pirate gets a chance to propose the division. Now tell what will happen and why. All the pirates are intelligent and the first priority is to stay alive and the next priority is to get as much money as possible.

The least senior pirate has no incentive to ever vote for a proposed division, since he cannot possibly be killed and his share is maximized if he is the only one left. He will never get that chance, though, because even if the first four pirates are killed, the second least senior pirate can vote for his own proposed division, whatever it is, and win, since his own vote is already half the votes. Both will vote no on all proposed divisions besides their own.

The third least senior pirate knows that the two pirates below him will vote against him if his turn comes, so it is in his interest to make sure someone above him makes a winning proposal. He will vote yes for anything. The fourth least senior, or the third most senior, pirate knows that if his turn comes, his own vote combined with that of the pirate right below him (who will vote for anything to save his own life) is enough for his proposal to win. Therefore, he wants his turn to come and will vote against the first two proposals.

And so it comes to the second most senior pirate. There are three no votes and one yes vote below him. If his turn comes, he is guaranteed to be killed. So, no matter what the most senior pirate proposes, he will accept. With two guaranteed yes votes below him, the most senior pirate can propose keeping the entire $300 to himself and be sure the division will be accepted. Why wouldn't he?

Note: The wikipedia page on this problem (https://en.wikipedia.org/wiki/Pirate_game) indicates that I didn't give the pirates enough credit. Specifically, the least-senior pirate isn't dumb, so he knows that he'll never get a chance to propose his own division. He'll take what he can get. This means the third least senior pirate would in fact be able to win by bribing the least senior pirate with a single dollar. The logic propagates upward, so the fourth least/third most senior pirate can also bribe the lowest ranked pirate and still win a majority. (The wikipedia page excludes this possibility, since it includes the rule that pirates will choose to kill in a tie, so the least senior pirate will vote to kill the fourth least and wait for a bribe from the third least instead. In this case, it's the second least senior pirate who gets the bribe.) Continuing upwards, the second most senior pirate will have to give two bribes to get a majority and would give them to the two who would lose if it went down one more rank: the fourth and sixth pirates. And now we're back to the most senior pirate, who also gives out two bribes, this time to the third and fifth pirates. Hey, $298 isn't bad.

55 Reconsider the pirate problem above, where we start with only one indivisible dollar. Who gets the dollar, and how many are killed?

I think this is where the wikipedia page's stipulation that by default pirates favor killing is important. Consider the least senior pirate. There's no way he's getting anything, so how are we to predict how he votes? With the condition that pirates prefer to see other pirates die, we can be sure he will vote no unless he has something to gain.

Going with the wikipedia condition, the dollar would go to the third most senior pirate. Only the first and second pirates have an incentive to let the first pirate keep the coin, since the second knows if it gets to him he will die. So, the first pirate only gets two votes and dies. The second pirate takes his turn, but no one has any incentive to let him keep it. This changes with the third pirate. The fourth pirate knows that the two below him will vote against him, the fifth because he can win in a tie of the lower two and the sixth because he's bloodthirsty. The fourth pirate would rather live, so he's incentivized to let the third one keep the dollar. With two votes, the third pirate wins the tie.