CSCI 104L: Lecture 6

Runtime Analysis

- \( f(n) \) is \( O(g(n)) \) means \( f(n) \leq c \cdot g(n) \), for all \( n \geq n_0 \), for some constants \( c, n_0 \). That is, our algorithm never takes more time than \( g(n) \) times a constant for sufficiently large inputs.

- \( f(n) \) is \( \Theta(g(n)) \) means \( f(n) \) is \( O(g(n)) \) and \( f(n) \) is \( \Omega(g(n)) \). This means that, up to constant factors, \( f \) and \( g \) are the same function.

- Big-Oh notation measures the growth rate of an algorithm, ignoring those pesky constant factors, and concerning itself with what happens for very large inputs. We typically consider the worst-case.

Calculating worst-case

- Do not assume a specific input. If some inputs do one thing, and other inputs do the other, assume the worst of the two happens.

- An elementary statement such as \( a[i]++ \), is a constant number of steps, so write it as \( \Theta(1) \).

- If you have two code blocks with runtime \( T_1(n) \) and \( T_2(n) \), and you run them in sequence, then add the runtimes. The runtime would be \( T_1(n) + T_2(n) \).

- When you have a for loop, add up each element. So if a block takes \( T_i(n) \) time, for \( i \) ranging from \( i = 0 \) up to \( i = n - 1 \), then the runtime would be \( \sum_{i=0}^{n-1} T_i(n) \). Note the variable \( i \) is saying that the block may take a different amount of time for each iteration.

Exercise 1. Calculate the runtime:

```c
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        a[i][j] = i * j;
```

The following sums may come up in analysis and may prove useful to you.

- \( \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2) \). This is called the arithmetic series.

- \( \sum_{i=0}^{n} \theta(i^p) = \Theta(n^{p+1}) \). This is a general form of the arithmetic series.

- \( \sum_{i=0}^{n} c^i = \frac{c^{n+1} - 1}{c-1} = \Theta(c^n) \). This is called the geometric series.

- \( \sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n) \). This is called the harmonic series.

Exercise 2. Calculate the runtime:

```c
for (int i = 0; i < n; i++)
    if (a[i][0] == 0)
        for (int j = 0; j < i; j++)
            a[i][j] = i * j;
```

Exercise 3. Calculate the runtime:

```c
for (int i = 1; i < n; i *= 2)
    for (int j = 0; j < i; j++)
        a[i][j] = i * j;
```
Figure 1: XKCD # 163: His books were kinda intimidating; rappelling down through his skylight seemed like the best option.

Exercise 4. Calculate the runtime:

```java
for (int i = 0; i < n; i++)
    if (i == 0)
        for (int j = 0; j < n; j++)
            a[i][j] = i * j;
```

Exercise 5. What is the running time of the following code?

```java
// t = target element. b = array. len = length of array.
int iterativeBinarySearch(int t, int *b, int len) {
    int lo = 0, hi = len - 1, mid;
    while (lo <= hi) {
        mid = (hi + lo) / 2;
        if (b[mid] == t) return mid;
        else if (t < b[mid]) hi = mid - 1;
        else lo = mid + 1;
    }
    return -1;
}
```

Exercise 6. Calculate the runtime:

```java
for (int i = 0; i < n; i++)
    if ((i % 2) == 0)
        for (int j = 0; j < n; j++)
            a[i][j] = i * j;
    else
        a[i][0] = i;
```
Exercise 7. Calculate the runtime:
for (int i = 1; i < n; i++)
    for (int j = 0; j < n; j += i)
        a[i][j] = i*j;

Exercise 8. Calculate the runtime:
for (int i = 0; i < n; i++)
    for (int j = i; j < n; j++)
        for (int k = i; k < j; k++)
            a[i][j][k] = i*j*k;

Analyzing recursive functions

Exercise 9. How can you analyze something like this?
void recurse(int *A, int size) {
    if (size <= 1) return;
    // do stuff (taking O(1) time)
    recurse(first half of A, size/2);
    recurse(second half of A, size/2);
}

Exercise 10. Find a recurrence relation, and analyze the runtime:

int binarySearch(int t, int *b, int lo, int hi) {
    if (hi < lo) return -1; // nothing to search, it's not in the array.
    else {
        int mid = (hi+lo)/2; // the middle of the array, rounded down.
        if (t == b[mid]) return mid; // found it!
        else if (t < b[mid]) return binarySearch(t, b, lo, mid-1); // search left.
        else return binarySearch(t, b, mid+1, hi); // search right.
    }
}

Exercise 11. Find a recurrence relation:
int *a;
void foo(int left, int right, int digit) {
    for (int i = left; i <= right; i++) a[i] += digit;
    if (right > left) {
        foo(left, (left+right)/2, 0);
        foo((left+right)/2+1, right, 1);
    }
}

int main() {
    int n = 8; // guaranteed to be a power of 2.
    a = new int[n];
    for (int i = 0; i < n; i++) a[i] = 0;
    foo(0, n-1, 0);
    for (int i = 0; i < n; i++) cout << a[i] << endl;
    return 0;
}