# CSCI 104L: Lecture 6

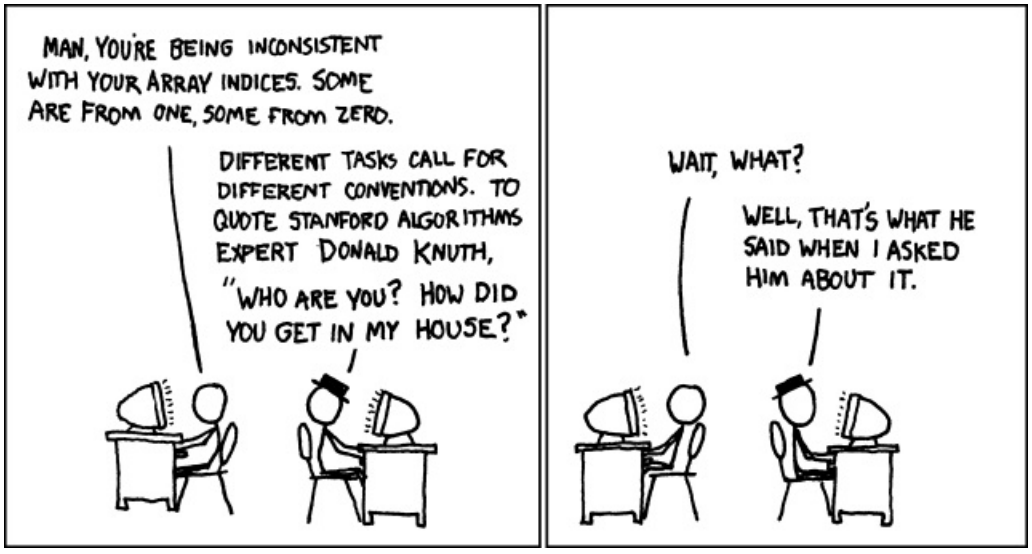


Figure 1: XKCD # 163: His books were kinda intimidating; rappelling down through his skylight seemed like the best option.

**Exercise 1.** Calculate the runtime:

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

**Exercise 2.** What is the running time of the following code?

```
//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 3.** Calculate the runtime:

```
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 4.** Calculate the runtime:

```
for (int i = 1; i < n; i++)
  for (int j = 0; j < n; j += i)
    a[i][j] = i*j;
```

**Exercise 5.** 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 6.** 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 7.** 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 8.** 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;
}
```