Example: polynomial evaluation.

input: polynomial f & evaluation point x.

output: $f(x)$.

$$f(x) = \sum_{i=0}^{d} a_i x^i$$

How to store in C++?

Save list of coefficients $a_0, a_1, \ldots a_d$.

```
int polyEval(int* a, int d, int x)
{
    // Note: a must point to an array
    // of d+1 elements!
    // if f = sum_{i=0}^{d} a_i x^i, a[i] = a_i
    int sum = 0;
    for(int i = 0; i <= d; i++)
        sum += a[i] * pow(x, i);

    return sum;
}
```

$f(x) = 3 + 7x + 2x^2$

How to call?

```
int a[3] = {3, 7, 2};
int result = polyEval(a, 2, 97);
// computes f(97)
```

How expensive is it to call polyEval?

Let's count the # of multiplications as a function of d.

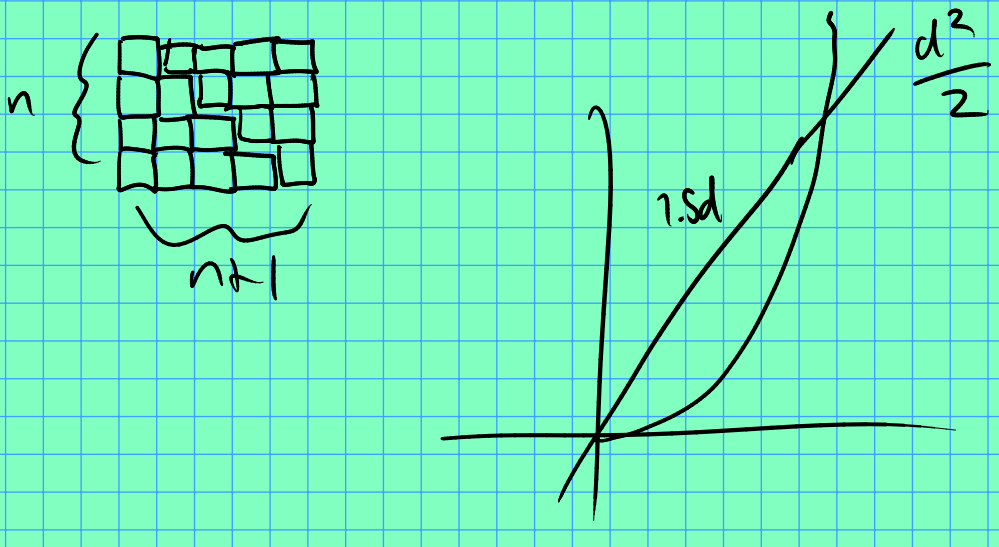
```
for(int i = 0; i <= d; i++)
    sum += a[i] * pow(x,i);
```

d+1 — costs i mults.

Total # mults: $1 + 2 + 3 + 4 + \cdots + (d+1)$

$$= \frac{(d+1)(d+2)}{2} \approx \frac{d^2}{2}$$

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$



Idea for saving on multiplications:

pow(x,i+1) is easy since pow(x,i)! just mult. by x.

```
int polyEvalBetter(int* a, int d, int x)
{   int sum = 0;
    int powxi = 1; // stores x^i
    for (int i = 0; i <= d; i++) {
```

```
        sum += a[i] * powxi;
        powxi *= x;
    }
    return sum;
}
```

Now how many multiplications?

2(d+1). Much better!

Challenge problem: can you do this with only d+1 or fewer multiplications?
(And a linear # of additions, of course.)