

## hw 2a

***I pledge my honor that I have abided by the Stevens Honor System.*** -Joshua Schmidt

page 67:

```
def mystery(n): # n = [non negative number input]
    S = 0
    for i in range(1, n + 1)
        S = S + i * i
    return S
```

1. consider the above algorithm:
  - a. what does this algorithm compute?
    1.  $n = 3 : 0 + 1 + 4 + 9 = 14$
    2. this algorithm computes the sum of all squared integers from 0 to n.
  - b. What is its basic operation?  $S = S + i \cdot i$ : This basic operation sets the current sum equal to the current sum + the square of the iterator.
  - c. How many times is the basic operation executed? n times.
  - d. What is the efficiency class of this algorithm?  $\theta(n)$
  - e. Suggest an improvement to the algorithm, or prove it cannot be improved.  
 $1 + 2 + \dots + n = \frac{n(n+1)}{2}$ .  $1^2 + 2^2 + \dots + n^2 = (\frac{n(n+1)}{2})(\frac{2n+1}{3}) = \frac{n(n+1)(2n+1)}{6}$ . Use algorithm defined below. Efficiency class of mystery\_2:  $\theta(1)$ .  $\theta(1)$  is better than  $\theta(n)$ .

```
def mystery_2(n):
    return int(n*(n+1)*(2*n+1)/6)
```

page 76:

1. solve the following recurrence relations:
  - a.  $x(n) = x(n-1) + 5$  for  $n > 1, x(1) = 0$ 
    1.  $x(n-1) = x(n-2) + 5$

1.  $x(n) = (x(n-2) + 5) + 5$
2.  $x(n) = x(n-2) + 10$
2.  $x(n-2) = x(n-3) + 5$ 
  1.  $x(n) = (x(n-3) + 5) + 10$
  2.  $x(n) = x(n-3) + 15$
3.  $x(n) = x(n-k) + 5 \cdot k$
4.  $n-k=1$ 
  1.  $k=n-1$
5.  $x(n) = x(n-(n-1)) + 5 \cdot (n-1)$ 
  1.  $x(n) = x(1) + 5 \cdot (n-1)$
  2.  $x(n) = 0 + 5 \cdot (n-1)$
  3.  $x(n) = 5 \cdot n - 5$
- b.  $x(n) = 3 \cdot x(n-1)$  for  $n > 1, x(1) = 4$ 
  1.  $x(n-1) = 3 \cdot x(n-2)$ 
    1.  $x(n) = 3 \cdot 3 \cdot x(n-2)$
    2.  $x(n) = 9 \cdot x(n-2)$
  2.  $x(n-2) = 3 \cdot x(n-3)$ 
    1.  $x(n) = 9 \cdot (3 \cdot x(n-3))$
    2.  $x(n) = 27 \cdot x(n-3)$
  3.  $x(n) = 3^k \cdot x(n-k)$
  4.  $n-k=1$ 
    1.  $k=n-1$
  5.  $x(n) = 3^{n-1} \cdot x(n-(n-1))$ 
    1.  $x(n) = 3^{n-1} \cdot x(1)$
    2.  $x(n) = 4 \cdot 3^{n-1}$
- c.  $x(n) = x(n-1) + n$  for  $n > 0, x(0) = 0$ 
  1.  $x(n-1) = x(n-2) + (n-1)$ 
    1.  $x(n) = (x(n-2) + n-1) + n$
    2.  $x(n) = x(n-2) + 2 \cdot n - 1$
  2.  $x(n-2) = x(n-3) + (n-2)$ 
    1.  $x(n) = (x(n-3) + (n-2)) + 2 \cdot n - 1$
    2.  $x(n) = x(n-3) + 2 \cdot n + n - 1 - 2$
    3.  $x(n) = x(n-3) + 3 \cdot n - 3$
  3.  $1 \ 3 \ 6 \ 10 \ 15 \ \dots = \frac{k(k+1)}{2}$ 
    1.  $x(n) = x(n-k) + k \cdot n - (k \cdot (k+1)/2)$
  4.  $n-k=0$ 
    1.  $n=k$
  5.  $x(n) = x(n-n) + n \cdot n - (n \cdot (n+1)/2)$ 
    1.  $x(n) = x(0) + n^2 - n(n+1)/2$
    2.  $x(n) = n^2 - n(n+1)/2$
- d.  $x(n) = x(\frac{n}{2}) + n$  for  $n > 1, x(1) = 1$  (solve for  $n = 2^k$ )
  0.  $x(2^k) = x(2^{k-1}) + 2^k$ 
    1.  $x(2^{k-1}) = x(2^{k-2}) + 2^{k-1}$ 
      1.  $x(2^k) = (x(2^{k-2}) + 2^{k-1}) + 2^k$
      2.  $x(2^k) = x(2^{k-2}) + \frac{2^k}{2} + 2^k$

3.  $x(2^k) = x(2^{k-2}) + \frac{3}{2} \cdot 2^k$
2.  $x(2^{k-2}) = x(2^{k-3}) + 2^{k-2}$ 
  1.  $x(2^k) = (x(2^{k-3}) + 2^{k-2}) + \frac{3}{2} \cdot 2^k$
  2.  $x(2^k) = x(2^{k-3}) + \frac{2^k}{4} + \frac{3}{2} \cdot 2^k$
  3.  $x(2^k) = x(2^{k-3}) + \frac{7}{4} \cdot 2^k$
3.  $1 \ 3 \ 7 \ 15 \ 31 \ \dots = 2^n - 1$ 
  1.  $x(2^k) = x(2^{k-j}) + \frac{2^j-1}{2^j} \cdot 2^k$
  2.  $x(n) = x(\frac{n}{2^j}) + \frac{2^j-1}{2^j} \cdot n$
4.  $1 = \frac{n}{2^j}$ 
  1.  $2^j = n$
  2.  $j = \lg(n)$
5.  $x(n) = x(\frac{n}{2^{\lg(n)}}) + \frac{2^{\lg(n)}-1}{2^{\lg(n)}} \cdot n$ 
  1.  $x(n) = x(\frac{n}{n}) + \frac{2^{\lg(n)}-1}{n} \cdot n$
  2.  $x(n) = x(1) + \frac{n-1}{n} \cdot n \cdot 2$
  3.  $x(n) = 1 + 2n - 2$
  4.  $x(n) = 2n - 1$
- e.  $x(n) = x(\frac{n}{3}) + 1$  for  $n > 1, x(1) = 1$  (solve for  $n = 3^k$ )
  1.  $x(3^k) = x(3^{k-1}) + 1$
  2.  $x(3^{k-1}) = x(3^{k-2}) + 1$ 
    1.  $x(3^k) = (x(3^{k-2}) + 1) + 1$
    2.  $x(3^k) = x(3^{k-2}) + 2$
  3.  $x(3^{k-2}) = x(3^{k-3}) + 1$ 
    1.  $x(3^k) = (x(3^{k-3}) + 1) + 2$
    2.  $x(3^k) = x(3^{k-3}) + 3$
  4.  $x(3^k) = x(3^{k-j}) + j$ 
    1.  $x(n) = x(\frac{n}{3^j}) + j$
  5.  $1 = \frac{n}{3^j}$ 
    1.  $3^j = n$
    2.  $j = \log_3(n)$
  6.  $x(n) = x(\frac{n}{3^{\log_3(n)}}) + \log_3(n)$ 
    1.  $x(n) = x(\frac{n}{n}) + \log_3(n)$
    2.  $x(n) = x(1) + \log_3(n)$
    3.  $x(n) = 1 + \log_3(n)$

pages 76-77

```
def S(n):
    if n is 1 return 1
    return S(n-1) + n * n * n
```

1. consider the following recursive algorithm for computing sum on  $n$  cubes:  
 $S(n) = 1^3 + 2^3 + \dots + n^3$ 
  - a. set up and solve a recurrence relation for the number of times the algorithm's basic operation is executed
    0.  $x(n) = x(n-1) + 1, x(1) = 1$
    1.  $x(n-1) = x(n-2) + 1$

1.  $x(n) = (x(n-2) + 1) + 1$
2.  $x(n) = x(n-2) + 2$
2.  $x(n-2) = x(n-3) + 1$ 
  1.  $x(n) = (x(n-3) + 2) + 1$
  2.  $x(n) = x(n-3) + 3$
3.  $x(n) = x(n-k) + k$
4.  $n-k = 1$ 
  1.  $k = n-1$
5.  $x(n) = x(n-(n-1)) + (n-1)$ 
  1.  $x(n) = x(1) + (n-1)$
  2.  $x(n) = 1 + (n-1)$
  3.  $x(n) = n$

- b. The non-recursive, straightforward algorithm for computing the sum is also  $\mathcal{O}(n)$ . This is because there is a for loop, and you have a sum variable that you keep adding the cube of the iterator to (see `iterative` function below). There is also a  $\theta(1)$  algorithm, which I showed below in the `constant_time` function. This non-recursive algorithm is faster than the recursive algorithm, or takes the same amount of time for  $n = 1$ . It uses the equation  $S(n) = \frac{n^2 \cdot (n+1)^2}{4}$ .

```
def iterative(n):
    if n is 1 return 1
    S = 0
    for i in range(1, n + 1):
        S += i ** 3
    return S

def constant_time(n):
    return (n**2) * ((n+1)**2) / 4
```