CSE 361 Lecture Notes

Daniel Dyla

October 11, 2016

1 Sums

1.1 Sum of constant

$$\sum_{i=k}^{n} c = c + c + c + c + ... + c \tag{1}$$
$$= n - k + 1 \tag{2}$$

1.2 Sum of integers

$$\sum_{i=0}^{n} i = 0 + 1 + 2 + 3 + ... + n \tag{3}$$
$$= \frac{n(n+1)}{2} \tag{4}$$

1.3 Sum of squares

$$\sum_{i=0}^{n} i^2 = 0 + 1 + 4 + 9 + ... + n^2 \tag{5}$$
$$= \frac{n(n+1)(2n+1)}{3} \tag{6}$$

2 Complexity

- Big Theta has upper and lower bound
- Big Omega has lower bound
- Big Oh (Omnicron) has upper bound

3 Recursive Complexity

1. Take the base case
2. Take other cases
3. Expand and solve

3.1 Simple Case

How many times is the function $f()$ run?

```
def r(n):
  f()
  if n == 1:
    return 5
  else:
    return r(n-1)
```

$$T(n) = \begin{cases} 1 & n = 1 \\ 1 + T(n-1) & else \end{cases}$$

$$\begin{aligned} T(n) &= 1 + T(n-1) && (7) \\ &= 1 + 1 + T(n-2) && (8) \\ &= 1 + 1 + 1 + T(n-3) && (9) \\ &= 1 + 1 + 1 + 1 + T(n-4) && (10) \\ &= 1 + 1 + 1 + 1 + ... + 1 && (11) \\ &= n && (12) \end{aligned}$$

3.2 More Complex Case

How many times is the function $f()$ run?

```
def r(n):
  f()
  if n == 1:
    return 5
  else:
    return r(n/2)
```

$$T(n) = \begin{cases} 1 & n = 1 \\ 1 + T(n/2) & else \end{cases}$$

$$\begin{aligned} T(n) &= 1 + T(n/2) & (13) \\ &= 1 + (1 + T(n/4)) & (14) \\ &= 1 + (1 + (1 + T(n/8))) & (15) \\ &= 1 + \log_2 n & (16) \end{aligned}$$

4 prove $2n^3 - n^2 + n - 2 \in \Theta(n^3)$

Claim: there is a C_1, C_2, n_0 such that $C_1 n^3 <= 2n^3 - n^2 + n - 2 <= C_2 n^3 \forall n >= n_0$

$C_1 = 1$, $C_2 = 3$, $n_0 = 2$

5 Analyze the function

```
def mystery(n):
    t = 0
    for i in range(1, n+1):
        s = 1
        for j in range(1, n+1):
            s = s * j
        t = t + s
    return t

mystery(5)
```

5.1 What does it do?

The sum of all factorials from 0 to n.

- The inner loop computes factorials
- The outer loop sums those factorials

5.2 Runtime

$$
\begin{aligned}
T(n) &= 1 + \sum_{i=1}^{n}(2 + \sum_{j=1}^{n} 1) && (17) \\
&= 1 + \sum_{i=1}^{n}(2 + n) && (18) \\
&= 1 + n(2 + n) && (19) \\
&= n^2 + 2n + 1 && (20) \\
&\in \Theta(n^2) && (21)
\end{aligned}
$$

6 Lecture 5

Mostly we talked about HW-1 today. See hw1.org.

6.1 Problem

Given an increasing function $f(x)$, defined for non-negative x, and a number T, find a number $z \in [0,]$, such that $f(z) = T$. f is expensive.

```
def increase(n):
    return 2 * n

def binsearch(f, T, a, b, e=0.1):
    while 1:
        m = (float(a) + b) / 2
        fm = f(m)
        err = (fm - T) ** 2
        if (err < e ** 2):
            return fm
```

```
        elif fm < T:
            a = m
        else:
            b = m

def find_z(f, T, e=0.1):
    a, b = 0, 1
    while (f(b) < T):
        a = b
        b = increase(b)

    z = binsearch(f, T, a, b)
    return z
```

Initial search runs in $f'(T)$. Binary search runs in $log_2((b-a)/\epsilon)$.

$$
\begin{aligned}
R(f,T) &= \log_2(f'(T)) + \log_2(\frac{b-a}{\epsilon}) && (22)\\
&= \log_2(f'(T)) + \log_2(\frac{\frac{f'(T)}{2}}{\epsilon}) && (23)\\
&\in \Theta(\log_2(f'(T))) && (24)
\end{aligned}
$$

7 Lecture 6

Given an array of real numbers, find a contiguous subarray with the largest possible sum.

```
def A0(l):
    n = len(l)
    large = l[0]
    for i in range(n):
        for j in range(i,n):
            large = max(large, sum(l[i:j+1]))
    return large

l = [1, 3, 4, 2, -7, 5]
A0(l)
```

```python
def A1(l):
    n,largest = len(l),0
    for i in range(n):
        s = 0
        for j in range(i,n):
            s += l[j]
            largest = max(s,largest)
    return largest

A1(l)

def A2(l):
    c = [0]
    for i in range(len(l)):
        c.append(c[i] + l[i])
    largest = 0
    for i in range(len(l)):
        for j in range(i, len(l)):
            s = c[j+1] - c[i]
            largest = max(s, largest)
    return largest



A2(l)

def A3(a):
    n = len(a)
    m = n//2
    a1 = a[:n]
    a2 = a[n:]
    l = n-1
    r = n
    c = [0]
    for i in range(n):
        c.append(c[i]+a[i])

def A4(a):
    mf, mh = 0, 0
```

```
for i in a:
    mh = max(mh+i, 0)
    mf = max(mf,mh)
return mf
```

A4(1)

7.1 Homework

maximum product of 3 elements in the array

8 Lecture 7

9 Test 1 Prep

- Verify Strassen at least once before the test
- if T(m) >= T'(m), then T(m) $\in \Omega$(T'(m))
- if T(m) <= T'(m), then T(m) $\in$ O(T'(m))

10 Exam

10.1 Page 2

Count the arithmetic operations
sum from 0 to n-1
Don't forget T(n) where n is the length of the array

10.2 Page 3

It evaluates the polynomial at x
Code is efficient because it is in Θ(n) where the natural way to evaluate polynomials is in $\Theta(n^2)$
Horner's algorithm
$9 = C(10^4)^{7/2}$; $9 = C10^{14}$; $C = 9x10^{-14}$; $x = C10^{14}$ Solve for x

10.3 Page 4

Efficient algorithms for the maximum subarray problem by distance
"kadane's algorithm"

```
m = 0
subarray = [[0]]
for each row r1 in matrix:
    for each element e1 in r1:
        for each row r2 below r1:
            for each element e2 right of e1:
                s = 0
                for each row r3 from r2 to the end:
                    for each element e3 from e2 to the end:
                        s += a[r3,e3]
                if s > m:
                    m = s
                    subarray = submatrix(r1,r2,e1,e2)

def msum(a):
    m = len(a) # row
    n = len(a[0]) # col
    best = a[0][0]
    idxs = [0,0,0,0]
    for tlrow in range(m):
        for tlcol in range(n):
            for brrow in range(tlrow,m):
                for brcol in range(tlcol,n):
                    s=arrsum(a,tlrow,tlcol,brrow,brcol)
                    if s > best:
                        best,idxs = s, [tlrow,tlcol,brrow,brocl]
    return best

def arrsum(a,tlr,tlc,brr,brc):
    s = 0
    for i in range(tlr,brr+1):
        for j in range(trc,brc+1):
            s += a[i][j]
    return s
```

```
arrsum([[1,2][3,4]],0,0,1,1)
```

Count # of times "s += a[i][j] is called

10.4 Page 5

All true

set $\{g(n) \mid \exists$ c, $n_0 > 0$ g(n) $\leq$ cf(n), $\forall$ n $\geq n_0\}$

set $\{g(n) \mid \exists$ c, $n_0 > 0$ g(n) $\geq$ cf(n), $\forall$ n $\geq n_0\}$

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 1 + T(n-1) + T(n-2) + 1 & else \end{cases}$$