Algorithm Analysis

CSE/IT 122 ~ Algorithms and Data Stuctures

Counting Three For Loops

```
for i = 1 to n
    for j = 1 to i
        for k = 1 to j
            x = i * j * k
```

Counting Three For Loops

➔ Can apply the string approach

i	1	2			3					
j	1	1	2		1	2		3		
k	1	1	1	2	1	1	2	1	2	3

➔ Have ordered triplets

- (1,1,1),(1,1,2),(1,2,2),(2,2,2),(1,1,3),(1,2,3),(2,2,3),(1,3,3),(2,3,3),(3,3,3)

Counting Three Four Loops

➔ Apply what we know about r-combinations to the strings

$$\binom{3+3-1}{3} = \frac{(5)!}{3!(5-3)!} = \frac{5 \cdot 43!}{3!2!} = \frac{5 \cdot 4!}{2!} = 10$$

Counting Three For Loops

- ➔ Use the table approach

i	j	k	body
1	2	2	1
2	3	$2 + 3$	$1 + 2 = 3$
3	4	$2 + 3 + 4$	$1 + 2 + 3 = 6$

Counting Three For Loops

➔ Generalize what we learned from the table and r-combination formula

➔ In general, for three for loops the body executes:

$$\binom{r+n-1}{r} = \binom{3+n-1}{3} = \binom{n+2}{3} = \frac{(n+2)!}{3!(n+2-3)!}$$

$$= \frac{(n+2)!}{3!(n-1)!} = \frac{(n+2)(n+1)n(n+1)!}{6(n-1)!} = \frac{(n+2)(n+1)n}{6}$$

Counting Three For Loops

- We get the same answer if we expand our table approach

i	j	k	body
1	2	2	1
2	3	$2 + 3$	$1 + 2$
3	4	$2 + 3 + 4$	$1 + 2 + 3$
$\dots$	$\dots$	$\dots$	$\dots$
n	$n + 1$	$2 + 3 + \dots + (n + 1)$	$1 + 2 + \dots + n$

Counting Three For Loops

- The i-loop executes *n* times
- The j-loop executes:
  - $2 + 3 + \cdots + (n+1) = \frac{(n+1)(n+2)}{2} - 1 = \frac{n(n+3)}{2}$
- The k-loop executes:
  - $2 + 5 + 9 + \cdots + (2 + 3 + \cdots + (n+1))$
  - Or writing as a sum:

$$\sum_{i=2}^{n+1} \frac{i(i+1)}{2} - 1$$

Counting Three For Loops

- ➔ Change index to *k* and use *k = i - 1, k + 1 = i*
- ➔ When i = 2, k = 1, and when i = n + 1, k = n
- ➔ Thus you have:

$$\sum_{k=1}^{n} \frac{(k+1)(k+2)}{2} - 1$$

- ➔ Simplify and you get:

$$= \sum_{k=1}^{n} \frac{k^2 + 3k + 2 - 2}{2} = \sum_{k=1}^{n} \frac{k(k+3)}{2} = \frac{n(n+1)(n+5)}{6}$$

Counting Three For Loops

- ➔ And the body of the k-loop executes:
  - 1+3+...+(1+2+3+...+n) times.
  - Writing as a sum:

$$\sum_{i=1}^{n} \frac{i(i+1)}{2} = \frac{n(n+1)(n+2)}{6}$$

- ➔ Which is the same as our derived r-combination formula!
- ➔ Note: Used Wolfram Alpha to get the formula

If-Then-Else

- Worst case running time: the test, plus either the then part or the else part
  - Use whichever is LARGER

Back To Summing Those Integers

```
long sum_forloop(long n)
{
   long i = 1; //cost $c_1$; one time
   long sum = 0.0; //cost $c_2$; one time

   // for each iteration f loop body executes once
   // loop test adds one more time
   for( ; i <= n; i++) //cost $c_3$; $n + 1$ times
     sum += i; //cost $c_4$; $n$ times

   return sum; //cost $c_5$; one time

   //$T(n) = c_1 + c_2 + c_3(n + 1) + c_4 n + c_5 = kn + d$
   //where k and d constants
}
```

Back To Summing Those Integers

- ➔ Summing the counts, the function's run time is
  - $T(n) = n + 3$ which is approximately n for large values of n
- ➔ The running time is proportional to n
- ➔ In other words for input size of 1000, it would take 1000 instructions
- ➔ Can we do better?

Back to Summing Those Integers

- Duh, of course we can!

```
long sum(long sum)
{
    return (n * (n + 1))/2 ; // cost $c_1$ ; one time
}
```

- Running time of this is $T(n) = C_1$ for all *n*
  - Constant running time

Back to Summing Those Integers