

Problem 3: Runtime Analysis (24%)

Part (a)

```
void f1(int n)
{
    int i=2;
    while(i < n){
        /* do something that takes O(1) time */
        i = i*i;
    }
}
```

How many times does loop repeat? it squares i until it reaches n

i starts at 2, and squares till it reaches n but it's more than that...

Dummy Counter: use var k to count how many times we do something

$2^{2^i}$

number of times we run

k	i
0	2
1	4
2	16
kth	$2^{2^k}$

stops when  $i = or > n$

$2^{2^k} < n$

solve for k

$\log_2 \log_2(n)$

Part (b)

```
void f2(int n)
{
    for(int i=1; i <= n; i++){
        if( (i % (int)sqrt(n)) == 0){
            for(int k=0; k < pow(i,3); k++) {
                /* do something that takes O(1) time */
            }
        }
    }
}
```

$\theta(i^3)$

gets involved only something

$$\sum_{k=0}^{\sqrt{n}} \left( \sum_{d=0}^{i^3} \theta(1) \right)$$
  
 $i^3$

use k counter for inner loop

if it stat that doesn't always run... # times we don't enter if statement are inconsequential

k	i
1	$\sqrt{n}$
2	$2\sqrt{n}$
kth	$k\sqrt{n}$

$\sum_{i=0}^k \theta(i^3)$

treat this as  $\theta(k^3 \sqrt{n}^3)$

$$\sum_{j=0}^k \theta(j^3)$$

From here what?

Recall:  $\sum_{i=0}^n i^p = \theta(n^{p+1})$   
 $= k^4$

Part (c)

```
for(int i=1; i <= n; i++){
    for(int k=1; k <= n; k++){
        if( A[k] == i){
            for(int m=1; m <= n; m=m+m){
                // do something that takes O(1) time
                // Assume the contents of the A[] array are not changed
            }
        }
    }
}
```

this can only happen once

$\theta(n)$

$\log_2 n$

$A[k]$

imagine what  $A[k] = 1, 1, 1, 1, \dots$   
it would enter once

$$\sum_{i=0}^n \sum_{k=0}^n \left( \text{reason} \sum_{m=0}^n \right)$$

runtime of entering if statement:  $n \log n$  (cost # times)  
not entering:  $n^2 \cdot \theta(1)$

$\theta(n^2 + n \log n)$

Part (d)

Notice that this code is very similar to what will happen if you keep inserting into an ArrayList (e.g. vector). Notice that this is NOT an example of amortized analysis because you are only analyzing 1 call to the function f(). If you have discussed amortized analysis, realize that does NOT apply here since amortized analysis applies to multiple calls to a function. But you may use similar ideas/approaches as amortized analysis to analyze this runtime. If you have NOT discussed amortized analysis, simply ignore it's mention.

```
int f (int n)
{
    int *a = new int [10];
    int size = 10;
    for (int i = 0; i < n; i++)
    {
        if (i == size)
        {
            int newsize = 3*size/2;
            int *b = new int [newsize];
            for (int j = 0; j < size; j++) b[j] = a[j];
            delete [] a;
            a = b;
            size = newsize;
        }
        a[i] = i*i;
    }
}
```

inner loop =  $\theta(i)$   
# times we enter if statement

k	$\text{size}(i)$
1	10
2	15
3	22
15	$10(\frac{3}{2})^k$

enter loop stops when  $i < n$

$10(\frac{3}{2})^k < n$  from previous assumption

$k = \log_{\frac{3}{2}}(\frac{n}{10})$  enter if statement this many times

$\log_{\frac{3}{2}}(\frac{n}{10})$

$\sum_{i=0} \theta(i)$

$10 + 10(\frac{3}{2}) + 10(\frac{3}{2})^2 + \dots + (10(\frac{3}{2})^k)$

$10 \sum_{j=0}^k (\frac{3}{2})^j$

$= \theta(\frac{3}{2})$

$\theta(\frac{3}{2}) \log_{\frac{3}{2}}(\frac{n}{10})$