Part (a)

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

i is being squared each iteration

- hence we are cutting the time to reach n by a lot each time.
- this tells us runtime should be $< \Theta(n)$

i increases like 2, 4, 16, 256 . . .

We know from 170 that $\Theta(\log(n))$ looks like 2, 4, 8, 16, 32 . . .

$$\sum_{i=2}^{n-1}\left(\frac{1}{i}\right) = \Theta(\log(n))$$

$$\sum_{i=2}^{n-1}\left(\frac{1}{i^2}\right) = \boxed{\Theta(\log(\log(n)))}$$

basically, I just saw that $\log_2(n)$ is like halfing, and $\log_2(\log_2(n))$ is like sqrt, which is basically what we do to the runtime everytime we do $i = i*i$.

lets check:

say $n = 4$, $\log(4) = 2$, $\log(2) = 1$

true! first iteration $i = 2 \cdot 2 = 4 \nless n$

say $n = 256$, $\log(256) = 8$, $\log(8) = 3$

true! first iteration $i = 2 \cdot 2$

second $i = 4 \cdot 4$

third $i = 16 \cdot 16 = 256 \nless n$

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 */
            }
        }
    }
}
```

$\Theta(n)$ runtime

will evaluate to true $\sqrt{n}$ times

because i is counting to n, meaning

(continued)

it will pass through every factor of n including every time it is divisible by $\sqrt{n}$

- i.e. $n = 100$, $\sqrt{n} = 10$

i will $=$ $\underbrace{10, 20, 30, 40, 50, 60, 70, 80, 90, 100}_{\text{10 times it passes}}$

will take i^3 runtime. at its greatest $i = n$

so this for loop will take $\Theta(n^3)$ time

hence: $\sum_{i=1}^{n} \left(\Theta(1) + \sqrt{n} \left(\sum_{k=0}^{n^3-1} \Theta(1) \right) \right)$

$= \Theta(n) + \sqrt{n} \left(\Theta(n^3) \right)$

$= \Theta(n) + \Theta(n^{3+\frac{1}{2}})$

$= \Theta(n) + \Theta(n^{7/2})$

drop lower term

$\boxed{= \Theta(n^{7/2})}$

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
      }
    }
  }
}
```

$\Theta(n)$ runtime

$\Theta(n)$ runtime

start with big O since we do not ~~K~~now how many times it will evaluate to true

m = m+m tells us it is logarithmic $\Theta(\log(n))$ runtime write out:

$$\sum_{i=1}^{n}\left(\sum_{k=1}^{n}\left(\Theta(1) + O\sum_{m=1}^{n}\left(\frac{1}{m}\right)\right)\right)$$

At most, if statement will be true each iteration, aka, n times

$$= \sum_{i=1}^{n}\left(\Theta(n) + \sum_{i}\left(\Theta(\log(n))\right)\right)$$

$$= \sum_{i=1}^{n}\left(\Theta(n) + \sum_{i=1}^{n}\left(\Theta(\log(n))\right)\right)$$

$$\sum_{i=1}^{n}\left(\Theta(n) + \Theta(n\log(n))\right)$$

$= \Theta(n^2) + \Theta(n^2 \log(n))$

$\Theta(n^2 \log(n)) > \Theta(n^2)$

$\boxed{= \Theta(n^2 \log(n))}$

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;
     }
}
```

$\Theta(n)$

if statement will pass at most 2 times explanation below

will run size times $\Theta(size)$

write out:

$$\sum_{i=0}^{n-1}\left(\Theta(1) + 2\left(\sum_{j=0}^{size-1}(\Theta(1))\right)\right)$$

will pass when i=10, then size becomes 15 ($\frac{3 \cdot 10}{2} = 15$)

will pass again when i=15 and won't pass anymore because it's a decimal from then on and i is always an int.

$\left(\frac{15 \cdot 3}{2}\right) = 22.5$, $\frac{22.5 \cdot 3}{2} = 33.75$

and so on

$$= \sum_{i=0}^{n-1}(\Theta(1) + \Theta(2size))$$

$$= \Theta(n) + \Theta(2n \cdot size)$$

$\Theta(2n \cdot size) > \Theta(n)$ and remove constant $\boxed{= \Theta(n \cdot size)}$