HW 1: runtimes

Monday, August 22, 2022 8:45 PM

Problem 3: Runtime Analysis (24%)

Part (a)

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

How many times does loop repeat? It depends ... until it reaches n

i starts at 2, and goes +1 until it reaches n but it's more than that...

O(1)

Dummy counter var k to count how many times we do something

k	i
0	2
1	4
2	16
...	2^{2^k}

number of times inner loop runs

stops when $i = 2^{2^k} > n$

Subst for k

$= O(\log_2 \log_2 (n))$

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 */
            }
        }
    }
}
```

$O(i^3)$

goto involved only for this

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

i^3

use k counter for inner loop

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

k	i
1	$\sqrt{n}$
2	$2\sqrt{n}$
k^{th}	$k\sqrt{n}$

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

treat this as

$O(k^3 \sqrt{n}^3)$ = # times we enter in loop

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

From last what?

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

$= O(n \log(\sqrt{n}))$

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
      }
    }
  }
}
```

$O(n)$ — this can only happen once

$O(n)$

$O(\log_2 n)$

A[k]

i might not be A[k] = 1, 1, 1, 1, ... if it would return

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

running if entering if statement: $n \log n$ — cost # times

not entering: $n^2 \cdot O(1)$

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

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];  O(3)
  int size = 10;  O(1)
  for (int i = 0; i < n; i ++)  O(n)
  {
      if (i == size)
        {
          int newsize = 3*size/2;  O(1)
          int *b = new int [newsize];  O(newsize)
          for (int j = 0; j < size; j ++) b[j] = a[j];  O(size)
          delete [] a;  O(1)
          a = b;  O(1)
          size = newsize;  O(1)
        }
      a[i] = i*i;  O(1)
  }
}
```

inner loop = $O(i)$

times we enter if statement

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

outer loop stops when $i < n$

$10(\frac{3}{2})^k < n$ from prof we assume = enter if statement this many times

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

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

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

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

$$10 \sum_{i=0}^{k} \left(\frac{3}{2}\right)^i$$

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

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

$O(\log n)$