CSI 3334 Data Structures
Homework 1

Yufan Xu

January 23, 2018

Do your own work for this assignment; do not work with others. Consult the book and your professor for help if you need it. Please write neatly, or preferably type your answers. Use good grammar, correct spelling, and complete sentences.

1. (15 points) Using proof by induction, prove that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

   Solution:

   Base Case:
   Let $n = 1$.

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

   So $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ is true when$n = 1$.

   Inductive Hypothesis:
   Let $n = k$.
   Assume $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$ is true.

   Inductive Step:
   Let $n = k + 1$.

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

   So $\sum_{i=1}^{k+1} i = \frac{(k+1)(k+2)}{2}$ is true when $n = k + 1$.
   Which implies $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ is true.

2. (15 points) Using proof by induction, prove that $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

Solution:

Base Case:
Let $n = 1$.

$\sum_{i=1}^{1} i^2 = \frac{1(1+1)(2+1)}{6}$
$1^2 = \frac{1(2)(3)}{6}$
$1 = \frac{6}{6}$
$1 = 1$

So $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ is true when $n = 1$.

Inductive Hypothesis:
Let $n = k$.
Assume $\sum_{i=1}^{k} i^2 = \frac{k(k+1)(2k+1)}{6}$ is true.

Inductive Step:
Let $n = k + 1$.
$\sum_{i=1}^{k+1} i^2 = \frac{(k+1)(k+2)[2(k+1)+1]}{6}$
$\sum_{i=1}^{k} i^2 + (k+1)^2 = \frac{(k+1)(k+2)(2k+3)}{6}$
$\frac{k(k+1)(2k+1)}{6} + (k+1)^2 = \frac{(k+1)(k+2)(2k+3)}{6}$
$\frac{k(k+1)(2k+1)+6(k+1)^2}{6} = \frac{(k+1)(k+2)(2k+3)}{6}$
$\frac{(k+1)[k(2k+1)+6(k+1)]}{6} = \frac{(k+1)(k+2)(2k+3)}{6}$
$\frac{(k+1)(2k^2+k+6k+6)}{6} = \frac{(k+1)(k+2)(2k+3)}{6}$
$\frac{(k+1)(k+2)(2k+3)}{6} = \frac{(k+1)(k+2)(2k+3)}{6}$

So $\sum_{i=1}^{k+1} i^2 = \frac{(k+1)(k+2)[2(k+1)+1]}{6}$ is true when $n = k + 1$.
Which implies $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ is true.

3. (10 points) Exercise 2.1 in your textbook.

Solution:
$2/N < 37 < \sqrt{N} < N < NloglogN < NlogN = Nlog(N^2) < Nlog^2N < N^{1.5} < N^2 < N^2logN < N^3 < 2^{N/2} < 2^N$

Because $Nlog(N^2) = 2NlogN$
So $2NlogN = (NlogN)$

The function $Nlog(N^2)$ and $NlogN$ grow at the same rate.

4. (10 points) Exercise 2.6 in your textbook.

   Solution to (a):
   The fines are \$2 on day 1, \$4 on day 2, \$16 on day 3, \$256 on day 4, \$65536 on day 5.
   I have noticed that:
   $2 = 2^1$
   $4 = 2^2$
   $16 = 2^4$
   $256 = 2^8$
   $65536 = 2^{16}$

   The exponents are 1, 2, 4, 8, 16.
   $1 = 2^0$
   $2 = 2^1$
   $4 = 2^2$
   $8 = 2^3$
   $16 = 2^4$

   So the fine on day N would be: $2^{2^{N-1}}$

   Solution to (b):
   Suppose the fine on day D is N.
   So $D = 2^{2^{N-1}}$
   $logD = 2^{N-1}$
   $loglogD = N - 1$
   Which implies the Big-O of $2^{2^{N-1}} = O(loglogN)$
   So after N days, the fine D will be $O(loglogN)$ dollars.

5. (30 points) Exercise 2.7 in your textbook. For parts 1-4, label each finite-time statement with its own constant as we have in class (i.e. c1, c2, etc.). For example, code fragment (1) should be labeled as:

```
sum = 0;                        // c1
for (int i = 0;                 // c2
                i < n;          // c3
                       i++) {   // c4
    sum++;                      // c5
}
```

For parts 1-4, show how you assigned constants for each code fragment, give a precise T(n) for each function before giving your big-Oh answer. For parts 5 and 6, just give a big-Oh answer. Give your code that tests and times each function.

When timing these code fragments, use values of n that vary enough to show clear trends in the running times. Also, you should run each function many times in a row so that the running time is not zero (which isn't meaningful). You can time your code by calling the C function clock() before the code fragment runs, saving its result, then running the code fragment many times, then calling clock() again and finding the difference between the two calls. The value that clock returns is in units of CLOCKS_PER_SEC. Both clock and CLOCKS_PER_SEC are defined in the <ctime>library.

Solution to (1):

```
sum = 0;                        // c1(1)
for (int i = 0;                 // c2(1)
               i < n;           // c3(n+1)
                      ++i) {    // c4(n)
    ++sum;                      // c5(n)
}
```

$T(n) = c1(1) + c2(1) + c3(n+1) + c4(n) + c5(n)$
So $T(n) = O(n)$

Solution to (2):

```
sum = 0;                              // c1(1)
for (int i = 0;                       // c2(1)
               i < n;                 // c3(n+1)
                      ++i) {          // c4(n)
      for (int j = 0;                 // c5(1)
                     j  < n;          // c6(n+1)n
                            ++j) {    // c7(n^2)
          ++sum;                      // c8(n^2)
      }
}
```

$T(n) = c1(1) + c2(1) + c3(n+1) + c4(n) + c5(1) + c6(n+1)n + c7(n^2) + c8(n^2)$
So $T(n) = O(n^2)$

Solution to (3):

```
sum = 0;                              // c1(1)
for (int i = 0;                       // c2(1)
               i < n;                 // c3(n+1)
                     ++i) {           // c4(n)
      for (int j = 0;                 // c5(1)
                     j  < n*n;        // c6(n^2+1)n
                           ++j) {     // c7(n^2)n
          ++sum;                      // c8(n^2)n
      }
}
```

$T(n) = c1(1) + c2(1) + c3(n+1) + c4(n) + c5(1) + c6(n^2+1)n + c7(n^2)n + c8(n^2)n$
So $T(n) = O(n^3)$

Solution to (4):

```
sum = 0;                              // c1(1)
for (int i = 0;                       // c2(1)
               i < n;                 // c3(n+1)
                     ++i) {           // c4(n)
      for (int j = 0;                 // c5(1)
                     j  < i;          // c6(n-1)n
                           ++j) {     // c7((n-1)n-1)
          ++sum;                      // c8((n-1)n-1)
      }
}
```

$T(n) = c1(1) + c2(1) + c3(n+1) + c4(n) + c5(1) + c6(n-1)n + c7((n-1)n-1) + c8((n-1)n-1)$
So $T(n) = O(n^2)$

Solution to (5):
$T(n) = O(n^5)$

Solution to (6):
$T(n) = O(n^4)$

Test for (1):

```
void loop1(){
    clock_t start = clock();
    double sum = 0.0;
    int n = 1000;
```

```
    for (int i = 0;i < n; ++i) {
        ++sum;
    }
    clock_t end = clock();
    cout << fixed << setprecision(10);
    cout << static_cast<double>(end - start) /CLOCKS_PER_SEC << endl;
}
```

when n = 5000, time is 0.000018
when n = 10000, time is 0.00003
when n = 20000, time is 0.000057

Test for (2):

```
void loop2(){
    clock_t start = clock();
    double sum = 0.0;
    int n = 2;
    for (int i = 0;i < n; ++i) {
        for(int j = 0; j <n; ++j) {
            ++sum;
        }
    }
    clock_t end = clock();
    cout << fixed << setprecision(10);
    cout << static_cast<double>(end - start) /CLOCKS_PER_SEC << endl;
}
```

when n = 2, time is 0.000002
when n = 4, time is 0.000003
when n = 8, time is 0.000015

Test for (3):

```
void loop3(){
    clock_t start = clock();
    double sum = 0.0;
    int n = 2;
    for (int i = 0;i < n; ++i) {
        for(int j = 0; j <n*n; ++j) {
            ++sum;
        }
    }
```

```
    clock_t end = clock();
    cout << fixed << setprecision(10);
    cout << static_cast<double>(end - start) /CLOCKS_PER_SEC << endl;
}
```

when n = 10, time is 0.000005
when n = 20, time is 0.000023
when n = 40, time is 0.00018

Test for (4):

```
void loop4(){
    clock_t start = clock();
    double sum = 0.0;
    int n = 2;
    for (int i = 0;i < n; ++i) {
        for(int j = 0; j <i; ++j) {
            ++sum;
        }
    }
    clock_t end = clock();
    cout << fixed << setprecision(10);
    cout << static_cast<double>(end - start) /CLOCKS_PER_SEC << endl;
}
```

when n = 100, time is 0.000016
when n = 200, time is 0.000058
when n = 400, time is 0.000225

Test for (5):

```
void loop5(){
    clock_t start = clock();
    double sum = 0.0;
    int n = 400;
    for (int i = 0;i < n; ++i) {
        for(int j = 0; j <i*i; ++j) {
            for(int k = 0; k <j;++k) {
                ++sum;
            }
        }
    }
    clock_t end = clock();
```

```
    cout << fixed << setprecision(10);
    cout << static_cast<double>(end - start) /CLOCKS_PER_SEC << endl;
}
```

when n = 10, time is 0.000026
when n = 20, time is 0.000774
when n = 40, time is 0.02694

Test for (6):

```
void loop6(){
    clock_t start = clock();
    double sum = 0.0;
    int n = 400;
   for (int i = 0;i < n; ++i) {
        for(int j = 0; j <i*i; ++j) {
            if(j%i==0) {
                for (int k = 0; k < j; ++k) {
                    ++sum;
                }
            }
        }
    }
    clock_t end = clock();
    cout << fixed << setprecision(10);
    cout << static_cast<double>(end - start) /CLOCKS_PER_SEC << endl;
}
```

when n = 10, time is 0.000006
when n = 20, time is 0.000058
when n = 40, time is 0.00899

6. (20 points) Chapter 2.1 in your textbook describes how to determine the rank of two functions in little-oh notation using limits. Use this method to rank the following functions in terms of their relative growth rates:

   (a) $f(n) = log(n), g(n) = log2(n)$

   (b) $f(n) = 3n, g(n) = 2n$

   (c) $f(n) = log(n), g(n) = n$

   (d) $f(n) = 5n, g(n) = n$

Show your work. Use l'Hpital's rule where appropriate. Your final answers should indicate whether f(n) = o(g(n)), g(n) = o(f(n)), or f(n) = (g(n)).

Solution to (a):
$lim_{n\to\infty}\frac{f(n)}{g(n)} = \frac{log(n)}{log_2(n)}$
Because when $n \to \infty$,
Both $f(n) \to \infty$ and $g(n) \to \infty$
Now use l'Hpital's rule,
So $\frac{f'(n)}{g'(n)} = \frac{\frac{1}{nln10}}{\frac{1}{nln2}}$
$\frac{f'(n)}{g'(n)} = \frac{nln2}{nln10}$
So $\frac{f''(n)}{g''(n)} = \frac{\frac{d}{dn}nln2}{\frac{d}{dn}nln10}$
$\frac{f''(n)}{g''(n)} = \frac{ln2}{ln10}$
As we can see the final result $\frac{f''(n)}{g''(n)} = \frac{ln2}{ln10}$ is a non-zero constant.
So $f(n) = (g(n))$

Solution to (b):
$lim_{n\to\infty}\frac{f(n)}{g(n)} = \frac{3n}{2n}$
Because when $n \to \infty$,
Both $f(n) \to \infty$ and $g(n) \to \infty$
Now use l'Hpital's rule,
So$\frac{f'(n)}{g'(n)} = \frac{\frac{d}{dn}3n}{\frac{d}{dn}2n}$
$\frac{f'(n)}{g'(n)} = \frac{3}{2}$
As a result,$\frac{f'(n)}{g'(n)} = \frac{3}{2}$ is the final result which is a non-zero constant.
So $f(n) = (g(n))$

Solution to (c):
$lim_{n\to\infty}\frac{f(n)}{g(n)} = \frac{log(n)}{n}$
Because when $n \to \infty$,
Both $f(n) \to \infty$ and $g(n) \to \infty$
Now use l'Hpital's rule,
So $\frac{f'(n)}{g'(n)} = \frac{\frac{1}{nln10}}{\frac{d}{dn}n}$
$\frac{f'(n)}{g'(n)} = \frac{1}{nln10}$
Because when $n \to \infty, nln10 \to \infty$,
the result $\frac{f'(n)}{g'(n)} = \frac{1}{nln10}$ can be considered as 0.
So $f(n) = o(g(n))$

Solution to (d):
$lim_{n\to\infty}\frac{f(n)}{g(n)} = \frac{5n}{n}$
Because when $n \to \infty$,
Both $f(n) \to \infty$ and $g(n) \to \infty$

Now use l'Hpital's rule,
So $\frac{f'(n)}{g'(n)} = \frac{\frac{d}{dn}5n}{\frac{d}{dn}n}$
$\frac{f'(n)}{g'(n)} = \frac{5}{1}$
$\frac{f'(n)}{g'(n)} = 5$
As a result, $\frac{f'(n)}{g'(n)} = 5$ is a non-zero constant.
So $f(n) = (g(n))$

7. (5 points) What is the tightest big-Oh running time one can expect of an algorithm that takes as input an array of length n, and prints out every unique permutation of that array? Remember that a permutation is a shuffling of the order of the elements of the array. Keep in mind the time it takes to print out each array. Explain your answer.

   Solution:
   Suppose the length n of the array is 3.
   Now supposing there is a set $\{1, 2, 3\}$.
   All the combinations of this set are: $\{1, 2, 3\}, \{1, 3, 2\}, \{2, 1, 3\}, \{2, 3, 1\}, \{3, 1, 2\}, \{3, 2, 1\}$.
   As we can see there are six different combinations of the set $\{1, 2, 3\}$.
   So when $n = 3$, the running time is 6.
   Then we can get that where $6 = 3!$,
   $6 = 3 * 2 * 1$
   As a result, we can get that the running time for permutation with a length of n is O(n!).

8. (10 points) Do parts (a) and (b) of exercise 2.10 from your textbook. Explain your answers. For this problem, imagine that N is unbounded.

   Solution to (a):
   Suppose $N = 3$.
   So we have two 3-digit numbers being added together.
   Suppose the numbers are 123 and 412.
   We need to first add the single digits 3 and 2 together which gives us the number 5, so this is the first addition.
   Then we need to add the 2 and 1 together which gives us the number 3, so this is the second addition.
   At last wee need to add 1 and 4 together which gives us the number 5, and this is the third addition.
   After these 3 times of addition we can get the result which is 535.
   The following code fragment indicates how the addition works:

```
const int N = 3;
```

```
int sum[N], num1[N] = {1,2,3}, num2[N} = {4,1,2};

for (int i = 0; i < N; i ++) {
  sum[i] = num1[i] + num2[i];
}
```

This function demonstrates the process of adding two 3-digit numbers together, the for loop would run three times in order to get each digit of the sum.
As a result, the loop needs to run N times in order to add two N-digit numbers together, so the running time of adding two N-digit numbers is $O(N)$.

Solution to (b):
Suppose $N = 3$.
So we have two 3-digit numbers being added together.
Suppose the numbers are 120 and 100.
In order to multiply them together, we need to first multiply the digit 0 with 0, which gives us the first time of multiplication.
Then we need to multiply 2 with 0, which gives us the second time of multiplication.
Third, we need to multiply 1 with 0, which gives us the third time of multiplication.
forth, we need to multiply 0 with 0, which gives us the forth time of multiplication.
Then, we need to multiply 2 with 0, which gives us the fifth time of multiplication.
Then, we need to multiply 1 with 0, which gives us the sixth time of multiplication.
Then, we need to multiply 0 with 1, which gives us the seventh time of multiplication.
Then, we need to multiply 2 with 1, which gives us the eighth time of multiplication.
Lastly, we need to multiply 1 with 1, which gives us the ninth time of multiplication.
As a result, we need 9 steps to multiply two 3-digit numbers,
and $9 = 3^2$,
So the running time of multiplying two N-digit numbers is $O(N^2)$.