Jeonghyeon Woo

School of Electrical and Computer Engineering

CptS223: Advanced Data Structures C/C++

Spring 2018

Homework #1

Due: Monday 01/29/2018 @ 11:59pm

Instructions: This is an individual assignment and all work must be your own. Be sure to show your work when appropriate. This assignment must be turned in by the due date via Blackboard in *PDF* format. You may use any editor you like (or even print it out, *legibly* write in answers, and scan it in), but convert it to *PDF* for submission.

1. (10 points) Order the following set of functions by their growth rate: N, $\sqrt{N}$, $N^{1.5}$, N^2, N×log(N), N×log(log(N)), N×$\log^2$(N), 2/N , 2^N, 2^(N/2) , 37, N^2×log(N), N^4.

As N goes infinity, the number goes bigger in this order.

2/N < 37 < sqrt(N) < N < N*log(log(N)) < N*log(N) < N*log(N)^2 < N^1.5 < N^2 < N^2*log(N) < N^4 < 2^(N/2) < 2^N

And, it means the order of their growth rate.

2. (10 points) Consider the following algorithm to sort 'n' integers in array A[1...n]. First, find the smallest element and swap it with A[1]. Next, find the second

smallest element and swap it with A[2]. Repeat the process until all elements are sorted. Write a small pseudocode for this algorithm. Then state its worst-case and best-case running times using the θ-notation.

Pseudocode:

```
(i=1; i<n+1; i++){
        (j=i+1; j<n+1; j++){
                If(A[j] < A[i]){
                int temp = A[i];
                A[i] = A[j];
                A[j] = temp;
                }
        }
}
```

Best suit function is n(n+1)/2.

As n goes infinity, it come n^2/2.

1/2*n^2 = c*(n^2) when c = 1/2.

n(n+1)/2=theta(n^2)

Therefore, worst case is theta(n^2), and best case is theta(n^2) = O(1).

3. (10 points) Answer the following by using the definitions for asymptotic notations:

a) Prove that $10\sqrt{n} = O(n)$.

10*sqrt(n) <= c*n

c = 10

10*sqrt(n) <= 10*n

⇨ 10*sqrt(n) <= 10*sqrt(n)*sqrt(n) (As n goes infinity)

Thus, 10*sqrt(n) = O(n)

b) Is $2n = \theta(2^{n+1})$? Either prove or explain your answer.

2n = c*2^(n+1)?

As n goes infinity, 2^(n+1) = 2^n

We know, 2^n > 2n when n > 2

There is no constant value that can always make n^2 equal to 2n.

Thus, 2n != theta(2^(n+1))

4. (10 points) A program takes 10 seconds for input size 1000 (i.e., n=1000). Ignoring the effect of constants, approximately how much time can the same program be expected to take if the input size is increased to 2000, under each of the following run-time complexities?

a) O(N) = 20 sec

b) O(N logN) = 20*log(2000)/3 sec

c) $O(N^2)$ = 40 sec

5. (10 points) Given the following two functions, f() and g():

a) State the runtime complexity of both f() and g() in Big-O notation.

f(n) = O(n)

g(n) = O(n)

b) State the memory (space) complexity for both f() and g()in Big-O notation.

g(n) is recursive, so it saves variable n times: it saves n per calling.

g(n) = O(n)

f(n) is in one function. It save the variable in one place; it saves two variables: n and sum.

f(n) = O(2)

c) Write another function called "int h(int n)" that does the same task, but is significantly faster.

```
int h(int n){
        if(n<0){
        return 0;
        }
        return n;
}
```

```
int g(int n)
{
   if(n <= 0)    {
   return 0;
     }
   return 1 + g(n - 1);
}
```

```
int f(int n)
{
   int sum = 0;
   for(int i = 0; i < n; i++)
   {
      sum += 1;
   }
   return sum;
}
```

6. (10 points) State g(n)'s runtime complexity in Big-O notation.

```
int f(int n){    if(n
<= 1){
      return 1;    }
   return 1 + f(n/2);
}
int g(int n){
   for(int i = 1; i < n; i *= 2){
      f(i);    }
}
```

f(n) = O(log(n)
g(n) = O(log(n)*log(n))
= O(log(n)^2)

7. (10 points) Provide the algorithmic efficiency for the following tasks. Justify your answer, often with a small piece of pseudocode or a drawing to help with your analysis.

a) Determining whether a provided number is odd or even

```
bool isodd(int a){
        if(a%2==0) return false;
        return true;
}
```

isodd(a) = O(1)

b) Determining whether or not a number exists in a list

```
bool exist(int *A, int n, int a){
```

```
//A is array, n is number of items, a is number for searching
        For(i=0; i<n; i++) //loop n times
        {
          If(A[i]==a) return true;
        }
        return false;
}
```

exist(A, n, a) = O(n)

c) Finding the smallest number in a list

The same algorithm as problem number 2. However, we only need to find A[1], so loop is for n times.

smallest(A, n) = O(n)

d) Determining whether or not two unsorted lists of the same length contain all of the same values (assume no duplicate values)

Use the algorithm in problem 2 to sort the arrays: f(Array, n)=O(n^2)

```
bool issame(int *A, int *B, int n)
{
        f(A, n); //n(n+1)/2
        f(B, n); //n(n+1)/2

        for(i=0; i<n; i++){
        if(A[i]!=B[i]) return false; //n
        }

        return true;
}
```

2(n(n+1)/2)+n = n^2+2n
As n goes infinity, it goes n^2
n^2 = O(n^2)
Thus, issame(A, B, n) = O(n^2)

e) Determining whether or not two sorted lists contain all of the same values (assume no duplicate values)

Skip sorting part.

n = O(n)

Thus, the efficiency is O(n)

8. (10 points) What is transitivity property for notation θ? Prove that θ notation holds the transitivity.

f(n) = theta(g(n)) and g(n) = theta(h(n)), then f(n) = theta(h(n))

f(n) = theta(g(n))

⇨ f(n) = c1*g(n)

g(n) = theta(h(n))

⇨ g(n) = c2*h(n)

⇨ g(n) = c1*g(n)/c1

⇨ c1*g(n)/c1 = c2*h(n)

⇨ c1*g(n) = c1*c2*h(n)

⇨ c1*c2 = c since constant multiplied by constant is constant

⇨ f(n) = c*h(n)

⇨ f(n) = theta(h(n))

9. (10 points) What is the runtime complexity of Adam's famous string splitter code? Hint: Make sure to look into the source code for string.find() in the C++ std library. I've included that code (downloaded from GNU).

```
static vector<string> split(string text, string
delimiter) {
      vector<string> pieces;
       int location = text.find(delimiter);
         int start = 0;

       //while we find something
interesting       while (location !=
string::npos){

             //build substring
               string piece = text.substr(start, location - start);
               pieces.push_back(piece);
                start = location + 1;

             //find again
                location = text.find(delimiter, start);
          }
     string piece = text.substr(start, location - start);
      pieces.push_back(piece);
        return pieces;
}
```

Jeonghyeon Woo

GCC/G++ source downloaded from:

http://mirrors.concertpass.com/gcc/releases/gcc6.3.0/

Source file: gcc-6.3.0/libstdc++-v3/include/ext/vstring.tcc

```
template<typename _CharT, typename _Traits,
typename _Alloc,              template <typename,
typename, typename> class _Base>
    typename __versa_string<_CharT, _Traits, _Alloc, _Base>::size_type
    __versa_string<_CharT, _Traits, _Alloc, _Base>::
    find(const _CharT* __s, size_type __pos, size_type
__n) const     {
      __glibcxx_requires_string_len(__s, __n);
      const size_type __size = this->size();
      const _CharT* __data = this-
>_M_data();
      if (__n == 0)
        return __pos <= __size ? __pos : npos;

      if (__n <= __size)
        {
          for (; __pos <= __size - __n; ++__pos)
            if (traits_type::eq(__data[__pos], __s[0])
                && traits_type::compare(__data + __pos + 1,
                                        __s + 1, __n - 1) == 0)
              return __pos;
        }
      return
npos;     }
```

find = O(n) // find do loop for n times since n is number of characters

Jeonghyeon Woo

The while loop in splitting algorithm works as goto (goto label 'find something interesting'). Therefore, it does loop for c times.

total loop = c*n times

c*n = c*n = O(n)

Thus, split(t, d) = O(n)

10. (10 points) (adapted from the 2012 ICPC programming competition) Write an algorithm to solve the following problem and specify its runtime complexity using the most relevant terms:

Given a nonnegative integer, what is the smallest value, k, such that

n, 2n, 3n, …, kn

contains all 10 decimal numbers (0 through 9) at least once? For example, given an input of "1", our sequence would be:

1, 2 1 , 3 1 , 4 1 , 5 1 , 6 1 , 7 1 , 8 1 , 9 1 , 10 1

and thus k would be 10. Other examples:

Integer Value	K value
10	9
123456789	3
3141592	5

Pseudo code:

```
int findk(int n){
        int j=0;
        bool digit[10]; //lights for each digit
```

```
        for(int i=0; i<10; i++){
                digit[i]=false; //turn off all lights
        }
        while(!(digit[0]*digit[1]* ….*digit[9]))
        //if it is not the case that all lights are on, then loop continues: loop ends
        when the product of all boolean values is true.
        {
                j++;
                int k=n*j; //try this number
                while(k>1 or k=1){
                        int a = k%10; //try the last digit of the number
                        digit[a] = true; //turning on the light of the digit
                        k=k/10; //try the second last digit of the number
                }//goes on unless the number is less than 1
        }
        return j; //it is k: j is the k value
}
```

Inner loop is repeating n times, and outer loop is repeating j times.

findk(n) = nj = c;

findk(n) = O(constant)

Thus, findk(n) = O(n*findk(n));