CptS 223 Homework #1

Elijah Andrushenko

1.

2/N, 37, √N, N log(log(N)), N, N logN, Nlog^2N, 2^(N/2), N^1.5, N^2, 2^N, N^4

2.

1) 175 Seconds

2) 177.24

3) 5359375

4) 4.78*10^52

4.

A) f()-> O(N) g()-> O(2^N)

B) f()-> θ(1), g()-> θ(1)

C) int h(int n) {return n}

5. g()-> O(n/2)

7. Adam's famous string splitter code -> O(N^2)

8. Input: A real number x and a nonnegative integer n.

Output: x^n

power(x, n)

if m = 0 then y = 1

else

y = power(x,[m/2])

y = y^2

if m is odd then y = xy

end if

return y

7. A) O(1) since its just finding if n % 2 = 0 then its even otherwise its odd so this is done in constant time

B) O(N) You have to check every element in the list to see if it exists in the list based on n elements.

C) O(1) if the list is sorted since you either check the end or the front of the list O(N) if it isn't since you need to compare every element in the list to find the smallest one

D) O(N^2) As every element in one list gets compared with every element in another list until you have done this process with all the elements in your initial list.

E) O(N) as you index both lists at the same time checking if two lists at the same index share the same values so you do this n times

F) O(logN) because every time you step through you are halving your searches thus this is a log pattern

8. ls – list files/directories

cp – copy files to another location

rm – remove files/directories

mkdir – make a new directory

ssh – login to a machine

g++ - run g++ on my machine

scp – lets you copy files to/from other hosts

9. argv and argc are how command line arguments are passed to main() in C and C++. argc will be the number of strings pointed to by argv This will (in practice) be 1 plus the number of arguments, as virtually all implementations will prepend the name of the program to the array.

The variables are named argc(argument count) and argv(argument vector) by convention, but they can be given any valid identifier: int main(int num_args, char** arg_strings) is equally valid.

They can also be omitted entirely, yielding int main(),if you do not intend to process command line arguments

1. [5] Order the following set of functions by their growth rate:

1) N

2) √N

3) N^1.5

4) N^2

5) N log N

6) N log(log(N))

7) N log^2 N

8) 2/N

9) 2^N

10) 2^(N/2)

11) 37

12) N^2 log(N)

13) N^4

2. [5] A program takes 35 seconds for input size 20 (i.e., n=20). Ignoring the effect of constants, approximately how much time can the same program be expected to take if the input size is increased to 100 given the following run-time complexities?

1. O(N)

2. O(N + log N)

3. O(N^3)

4. O(2^N)[1]

[1] You might need an online calculator with arbitrarily large numbers for this one. Scientific notation and 8 significant figures is just fine.

4. [8] Given the following two functions:

```
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;
}
```

A. [2] State the runtime complexity of both f() and g()

B. [2] State the memory (space) complexity for both f() and g()

C. [4] Write another function called "int h(int n)" that does the same thing, but is significantly faster.

5. [5] State g(n)'s runtime complexity:

```
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);
   }
}
```

7. [5] 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;
}
```

GCC/G++ source downloaded from: http://mirrors.concertpass.com/gcc/releases/gcc-6.3.0/
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;
    }
```

6. [10] (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

$$n, 2n, 3n, \ldots, kn$$

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

(space for #6)

7. [18] 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. [3] Determining whether a provided number is odd or even

B. [3] Determining whether or not a number exists in a list

C. [3] Finding the smallest number in a list

D. [3] Determining whether or not two **unsorted** lists of the same length contain all of the same values (assume no duplicate values)

E. [3] Determining whether or not two **sorted** lists contain all of the same values (assume no duplicate values)

F. [3] Determining whether a number is in a BST

8. [6] Fill in what these Linux commands do.

For example:

ls list files/directories

cp _____

rm _____

mkdir _____

ssh _____

g++ _____

scp _____

9. [4] How do these variables get set and what do they get set with?

```
int main(int argc, char* argv[]) {
      return(0);
}
```