

CptS 223 - Homework #1

Big-O and general Linux/Git topics

Please complete the homework problems on the following page. Note that this is an individual assignment and all work must be your own. Be sure to show your work when appropriate.

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. I have provided MS Word (doc) and LibreOffice (ODF) versions for your platform of choice.

Once you have your PDF file, put it into your Git repository in the HW1 directory, commit and push it. Once you've pushed your PDF file up, put something onto Blackboard so the TA knows to grade your work.

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

Unordered Complexities	Ordered Complexities
N	2/N
\sqrt{N}	37
$N^{1.5}$	\sqrt{N}
N^2	N
$N \log N$	$N \log(\log(N))$
$N \log(\log(N))$	$N \log N$
$N \log^2 N$	$N \log^2 N$
2/N	$N^{1.5}$
2^N	N^2
$2^{(N/2)}$	$N^2 \log(N)$
37	N^4
$N^2 \log(N)$	$2^{N/2}$
N^4	2^N

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. $O(N)$: 175 seconds
2. $O(N + \log N)$: 1176 seconds
3. $O(N^3)$: 4375 seconds
4. $O(2^N)^1$: $4.2312403681 \times 10^{25}$ seconds

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

3. [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()
g() runtime complexity = $O(1)$
f() runtime complexity = $O(N)$
- B. [2] State the memory (space) complexity for both f() and g()
g() space complexity = $O(N)$
f() space complexity = $O(1)$
- C. [4] Write another function called "int h(int n)" that does the same thing, but is significantly faster.

```
int h(int n)
{
    If( n <= 0) return 0;
    Else return n;
}
```

4. [5] State g(n)'s runtime complexity: $O(N/2)$

```
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);
    }
}
```

5. [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;
}
```

RUNTIME: $O(N)$

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

$$n, 2n, 3n, \dots, kn$$

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

```

Int main(int num)
{
    Bool hasNum[10] = false;
    Int k = 0;
    Int tmp = 0;
    String numChar = to_string(num)
    For(i= 0, i < numchar.length(), i++)
    {
        hasNum[numchar[i]] = true;
    }
    If(allTrue(hasNum)) return k;

    While(true)
    {
        ++K;
        Tmp = k * num;
        Numchar = to_string(tmp)
        For(I = 0, I < numchar.length(), i++)
        {
            hasNum[numchar[i]] = true;
        }
        If(allTrue(numChar)) return k;
    }
}

Bool allTrue(Bool &array[])
{
    For(int i, i < 10, i++)
    {
        If (array[i] == false) return false;
    }
    Return true;
}

```

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

$O(1)$ - is num wholly divisible by 2, yes or no

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

$O(N)$ - traverse thorough list until num exists or does not exist

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

$O(\log N)$ - put all numbers into a BST $O(N)$, then search for smallest $O(\log N)$

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

$O(n)$ - insert all values into a hash table $O(n)$, then search table with keys from other list $O(1)$

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

$O(n)$ - insert all values into a hash table $O(n)$, then search table with keys from other list $O(1)$

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

$O(\log N)$ search the BST for the item

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

For example:

ls list files/directories

cp Copy item

rm Remove Item

ssh Remotely login to a server via ssh


g++ Compiles a program with g++

scp Copies a file from one location to another with scp

9. [4] What is Git and what is it for?

Git is a version control software used for sharing and keeping track of different versions of files as they are updated.

10. [4] How do these variables get set and what do they get set with?



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

These variables are optionally set by the user when running from the terminal, if none are passed in, they default to 0 and null respectively. Argc is for the number of arguments passed in and argv is for the actual commands passed in.