CptS 223 - Homework #1

Big-O and general 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
√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? Chapter 2.1 notes that: T(N) ≤ cf(N). For this you'll need to find the c (constant scaling factor) for a given Big-O growth rate.

1. O(N)

35 = 20
x = 100
(35 *100)/ 20 = 175
Therefore O(N) = 175 seconds for input size 100.

2. O(N + log N)

35 = 20 + log20
x = 100 + log100

x= 35*(100 + log 100)/(20 +log20)
x = 159 .211
Therefore O(N + log N) = 159.211 seconds

3. O(N^3)

35 = 20^3
x= 100^3
x= (100^3 * 35)/ 20^3
x= 4375
Therefore O(N^3) = 4375 seconds

4. O(2^N)[1]

35 = 2^20
x= 2^100
x= (35 * 2^100)/ 2^20
x= 42.3 *10^24 seconds
Therefore O(2^N) = 42.3 *10^24 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()

The runtime complexity of f() is O(n) since for loop calls times.

The runtime complexity of g() is O(n) since function call n times.

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

The space complexity for f() = O(1) and g() = O(n).

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:

The runtime complexity for g(n) = log(n) * log(n)

=O(log^2(n))

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.Length() ){

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

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

```
// Excerpt from OpenJDK's implementation of string searching in
//  src/java.base/share/classes/java/util/regex/Pattern.java
//  The key component is how long it takes to run to find the next
match
boolean match(Matcher matcher, int i, CharSequence seq) {
            if (i > matcher.to - minLength) {
                matcher.hitEnd = true;
                return false;
            }
            int guard = matcher.to - minLength;
            for (; i <= guard; i++) {
                if (next.match(matcher, i, seq)) {
                    matcher.first = i;
                    matcher.groups[0] = matcher.first;
                    matcher.groups[1] = matcher.last;
                    return true;
                }
            }
            matcher.hitEnd = true;
            return false;
```

The runtime complexity of Adam's famous splitter code is O(N^2), because while loop runs N times and each time find()method runs on an average of n times. Therefore it is O(N^2).

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:

and thus k would be 10. Other examples:

Integer Value	K value
10	9
123456789	3
3141592	5

```
int findK(int input){
int digit = 0;
int arr[] = new int [10];
for(int k=1; digit <10;k++){
      int temp = input*k;
      int num =0;
      while(temp > 0){
            num = temp%10;
            arr[num] + =1;
            temp/=10;
            }
for(int i=0;i<10;i++){
      if(arr[i]>0){
      digit++;
      }
}


}
```

Time complexity is O(log n).

Note: I took help from a tutor.

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

```
      int (n){
      if(n%2==0){
```

```
n is even;
else {
n is odd;
}
}
```

The time complexity is O(1)

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

To check whether a given number is in the list or not, we need to iterate through the list until we find the number.

```
find(int n){
current = front.data
while ( current!= null){
if( n == current.data;
return true;
}
current= current.next;
}
return false;
}
```

The time complexity is O(n).

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

If a list is sorted, we just return the first element in the list. Therefore it has time complexity is O(1).

For unsorted list

```
current = front.data;

int min = front.data;

while(current!=null){

      if(min> current.data){

      min = current.data;

      }

      current = current.next;

      }

return min;

}
```

The time complexity for finding minimum value from the unsorted list is O(n) because we need to iterate through all the list.

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

The time complexity for checking whether two unsorted lists are same or not is O(n^2) because we are checking each element of list1 is present in the list2 or not. For that for each element in list1 we have to traverse the whole list 2.

```
      int len1 = list1.size();

      int len2 = list2 .size();

if(len1 == len2){

 for(int i=0;i<len1;i++){

for(int j=0;j<len2;j++){

  if( list1[i]=list2[j]){
```

```
}
```

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

```
       if (list1.size()!=list2.size()){
            return false;
        }


    current = list 1.front;
   current2 = lis2t.front;
        while(current!=null){
            if(current.data != current2.data){
             return false;
            }
           current = current.next;
     current2= current 2.next;
        }
        return true;
        }
```

The time complexity for checking two sorted lists are equal or not is O(n) because elements are sorted. Therefore we just check the corresponding elements of list1 and list2.

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

The time complexity for checking a number is in a BST is O(N) in the worst case and sometime it is O(log(N)).

```
boolean contains( IntTree root, int num){
if(root==null){
return false;
}
```

```
if(mum == root.data){

return true;

}

if(num<= root.data){

return contains (root.left, num);

}

else{

return contains(root.right, num);

}

}
```

8. [4] What is Git and what is it for?

Git is a free open source command line tool. It is a distributed version control system. It manages the changes in the source code during the software development. And it designed for coordinating the work among the programmers but it can also track down any changes made in the file. Git helps in handling the projects with speed and efficiency.

9. [2] If I need to get a copy of a Git repository off of the GitLab server, what command do I use?

git clone

10. [2] Once I've created/edited/removed a file in my Git repository, what command do I use to stage it for committing?

git add .

11. [2] Once I've staged all of my changes, which command do I use to create the next version of the repository?

git commit -m" message"

12. [2] Now that I've created at least one new update to my repository, which command do I use to send those changes to the GitLab server?

git push

13. [2] If the server has had updates from another computer, which command do I use to get these changes on my local computer without starting from a whole new copy of the Git repository?

git merge

14. [4] How does this variable get set and what is it get set with?


```
public static int                main(String args[]) {
     return(0);
}
```

The variable args[] is the command line argument which gets the values from the command line and set the values to the program to the command line argument.

For example, when we run a java program through the command line, the following argument store int the arg[] variable.

It should be noted that arg[0] holds the name of the program itself and arg[1] is a pointer to the first command line argument supplied, and *arg[n] is the last argument. If no arguments are supplied, argc will be one

Note: get help from online.