Lab 4: Algorithm Analysis

Due: Tuesday, February 18 at 4PM; **NO LATE DAYS**

Overview

In this lab, you will answer several math and algorithm questions and complete the mystery function exercise. You will submit a hard copy of this lab. You may type your solution, if you wish. **This is an individual lab**. You can retrieve requisite code from `update35` for the mystery function portion of the lab.

Deliverables

Your submission should be **concise and easy to read**. You should answer all 4 short answer questions, the run times of all 6 functions, written support of the mapping to mystery functions, and at-least two print out of graphs to support that argument.

Submission

You will submit this lab in hard copy to my mail slot outside my office, Science 270. Sorry, no late labs will be accepted for this lab since the quiz study session Tuesday night will go over answers.

Short Answer Questions

1. Justify the following Big-O for the given functions:

   (a) $5n^4 - 2n^3 + 10$ is $O(n^4)$

   (b) $\frac{1}{2}n - log(n)$ is $O(n)$

2. Prove the following claim by induction:

   (a) $\forall n \geq 1, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

3. Using loop-invariants/induction, prove that the `isSorted` function from class correctly verifies that an array A is sorted in descending order. HINT: Formally, you are trying to prove S: A is sorted in descending order.

```
function IsSorted(A, size)
    for i ← 0 ... size − 2 do
        if A[i] < A[i + 1] then
            return False
        end if
    end for
    return True
end function
```

4. (C-4.23 from Goodrich et. al) An evil king has n bottles of wine, and a spy has just poisoned one of them. Unfortunately, the king doesn't know which bottle it is. The poison is very deadly; just one drop diluted even a billion to one will still kill. Even so, it takes a full month for the poison to take effect. Design a scheme for determining exactly which one of the wine bottles was poisoned in just one month's time while utilizing $O(\log n)$ taste testers. Make a short argument for what you think the run time of your technique is, even if you cannot come up with an optimal solution. Please, keep solution to a paragraph or less.

Mystery Functions

In this part of the lab, you will first analyze six simple loop structures and determine their run time performance in terms of Big-O. Then, using the provided program `function_timer`, you will graph the empirical run times of these functions.

Functions

To begin, identify the Big-O run time for each of the following 6 functions. You do not need to provide a justification, simply write the function name followed by the Big-O on your solution. Use the strictest Big-O (i.e., the closest upper bound) and ignore all but the most significant term (e.g. **Ex 7:** $O(\log n)$).

```
Ex1(n)
  for(i=0;  i<n;  i++){
    a=i;
  }

Ex2(n)
  for(i=0; i<n; i+=2){
    a=i;
  }

Ex3(n)
  for(i=0; i<n*n; i++){
    a=i;
  }

Ex4(n)
  for(i=0; i<n; i++){
    for(j=0; j<=i; j++){
        a=i;
    }
  }

Ex5(n)
  for(i=0; i<n*n; i++){
    for(j=0; j<=i; j++){
      a=i;
    }
  }

Ex6(n)
  k=1;
  for(i=0; i<n; i++){
    for(j=0; j<=k; j++){
      a=j;
    }
    k=k*2;
  }
```

Decoding functions using `function_timer`

Each of the above functions has implemented and packaged in the executable `function_timer`, which has placed in your lab directory for this week. This program will provide an empirical run-time for each method and plots them automatically using the unix tool `gnuplot`.

To begin, here is the usage (this is an abbreviated version; obtain more details on the command line):

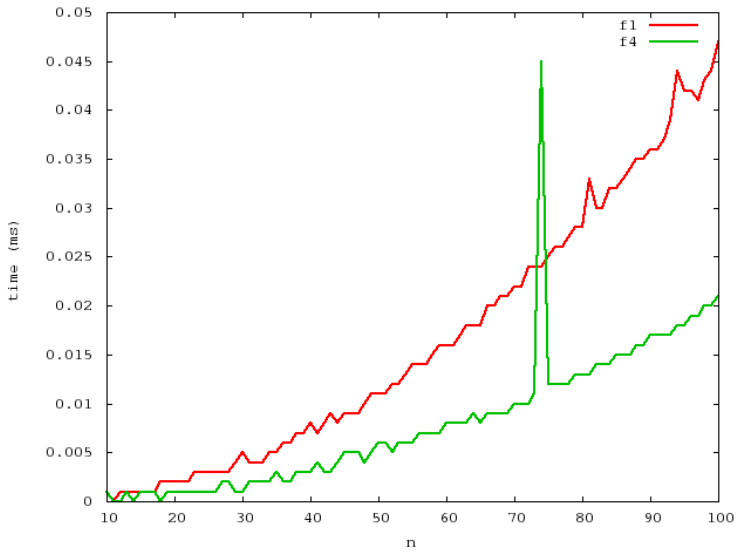
```
$ ./function_timer -h

OPTIONS:
        -h              print this help screen
        -n min_n        set the min value for n (dflt: 1)
        -m max_n        set the max value for n (dflt: 10)
        -[1-6]          Turn on mystery function number, e.g.
                        to run function 2 and 3: function_timer -2 -3
        -s out.png      Save the graph to a file out.png instead of graphing live
```

The key choices you have to make are:

- Which function(s) to plot. To plot `func1`, add `-1` as a command line argument. To plot `func2` vs `func3`, add `-2 -3`, etc.
- The minimum and maximum values for n. You should recall from lecture that n may have to be very large for fast algorithms and small for slow ones. One size definitely does not fit all.
- Whether to save to file or immediately load plot. You'll want to save once you get a result you like by using the -s option.

For example, you can compare functions 1 and 4 from $n = 10 \ldots 100$ and view the result using gnuplot:

```
$ ./function_timer -1 -4 -n 10 -m 100 | gnuplot
```



Note that you must 'pipe' the output of the program to the plotting program, `gnuplot`. This is done using the vertical bar `|`, which simply sends the output from `function_timer` straight to `gnuplot`. To compare functions 1 and 4 from $n = 10 \ldots 100$ **and** save the output to a file named out.png, do the following:

```
$ ./function_timer -1 -4 -n 10 -m 100 -s out.png | gnuplot
```

Now here is the twist: I've jumbled up the ordering of functions in `function_timer`. Your job is to figure out the mapping from the functions in `function_timer` (e.g., `-1, -2, ...`) to their

corresponding number above (e.g., `Ex1, Ex2, ...`). In addition to **determining the mapping between mystery functions and the functions above, you must support your argument in writing for each and provide a graph to support two of your mappings in your hard-copy submission.**