Lab 4: Algorithm Analysis

Due: Sunday, September 27 at 11:59PM

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 the `lab4-<your-username>` repository for the mystery function portion of the lab.

Deliverables

Your submission should be **concise and easy to read**. You should answer all 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 Center 251. Please remember to put your name on the submission.

Short Answer Questions

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

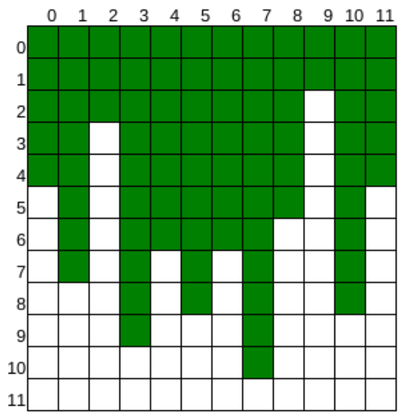
   (a) $2n^4 - \frac{1}{2}n^2 + 5n$ is $O(n^4)$

   (b) $n + \frac{1}{n}$ is $O(n)$

2. The function `checkMax`, given below, takes as input an array A of size n, and a number x, and should return true if A has an element bigger than x. Using loop invariants and induction, argue that `checkMax` works correctly. HINT: you are trying to show S: `checkMax` returns (a) true if there is i such that $A[i] > x$, and (b) false if no such i exists.

```
function CHECKMAX($A, n, x$)
    for $i \leftarrow 0 \ldots n-1$ do
        if $A[i] > x$ then
            return True
        end if
    end for
    return False
end function
```

3. Based on Goodrich, 4.26. The image below shows an n by n grid that is stored in memory as a two dimensional array. The element at row i column j can be indexed in constant time using `grid[i][j]`. Each cell in the grid contains either a 1 or a 0 (indicated in the figure with green and white blocks). In any column, all the ones appear before any of the zeros. Given such a grid, design an $O(n)$ algorithm for finding the column with the most ones (tallest green tower). Note there are $O(n^2)$ cells, so you cannot check every cell in the grid.



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. Use the strictest Big-O (i.e., the closest upper bound) and ignore all but the most significant term. (e.g. **Ex 7:** $O(n)$ **instead of** $O(n + 4 \log n)$). You will provide some written justification for your choices; see the end of the writeup.

```
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 been 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):

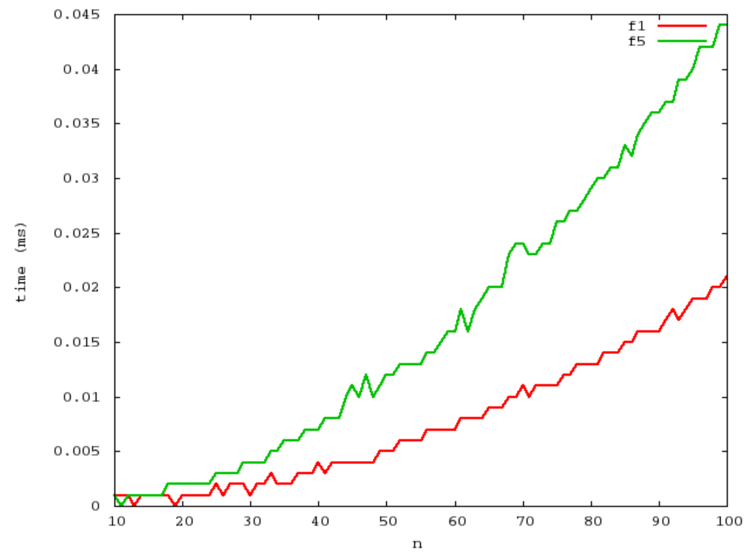
```
$ ./function_timer -h

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 filename     Use <filename> as the output file for gnuplot
```

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 lab 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 5 from $n = 10 \dots 100$ and view the result using gnuplot:

```
$ ./function_timer -1 -5 -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 sends the output from `function_timer` straight to `gnuplot`. To compare functions 1 and 5 from $n = 10 \dots 100$ **and** save the output to a file named out1vs5.png, do the following:

```
$ ./function_timer -1 -5 -n 10 -m 100 -s out1vs5.png |gnuplot
```

Much like in the testing assignment, 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 by writing a sentence or two justifying each matching. In addition, for (at least) two of the mappings, provide a graph that supports your arguments.