EC 504 – Fall 2022 – Homework 1

Due Friday Sept. 24, 2022 by 11:59PM as a pdf for the written part and coding problem. Submit both in your directory `/projectnb/alg504/yourname/HW1`

Start reading Chapters 1, 2, 3 and 4 in CLRS . They give a very readable introduction to Algorithms.. Also glance at Appendix A for math tricks which we will use from time to time.

1. (25 pts)

   (a) In CRLS do Exercise 3.1-4 on page 53, Problem 3.2-7 on page 60, Problem 3-2 on page 61, Problem 4.3-9 on page 88.

   (b) Place the following functions in order from asymptotically smallest to largest – using $f(n) \in O(g(n)$ notation. When two functions have the same asymptotic order, put an equal sign between them.

$$n^2 + 3n\log(n) + 5\ ,\ \ n^2 + n^{-2}\ ,\ n^{n^2} + n!\ ,\ n^{\frac{1}{n}}\ ,\ n^{n^2-1}\ ,\ \ln n\ ,\ \ln(\ln n)\ ,\ 3^{\ln n}\ ,\ 2^n\ ,$$
$$(1+n)^n\ ,\ n^{1+\cos n}\ ,\ \sum_{k=1}^{\log n}\frac{n^2}{2^k}\ ,\ 1\ ,\ n^2+3n+5\ ,\ log(n!)\ ,\ \sum_{k=1}^{n}\frac{1}{k}\ ,\ \prod_{k=1}^{n}(1-\frac{1}{k^2})\ ,\ (1-1/n)^n$$

2. (25 pts)

   (a) Substitute
   $$T(n) = c_1 n + c_2\ nlog_2(n)$$
   into $T(n) = 2T(n/2)+n$ to find the values of c_1, c_2 to determine the exact solution.

   (b) Generalize this to the case for $T(n) = aT(n/b) + n^k$ with trial solution
   $$T(n) = c_1 n^\gamma + c_2 n^k$$
   using $b^\gamma = a$. What happens when $\gamma = k$?
   Now set $\gamma = k$ but start over with the guess $T(n) = c_1 n^\gamma + c_2 n^\gamma log_2(n)$ to determine new values of c_1, c_2.

3. (50 pts) **Binary search** of a large sorted array is a classic devide and conquer algorithms. Given a value called the `key` you search for a match in an array `int a[N]` of N objects by searching sub-arrays iteratively. Starting with `left = 0` and `right = N-1` the array is divides at the middle `m = (right + left)/2`. The routine, `int findBisection(int key, int *a, int N)` returns either the index position of a match or failure, by returning `m = -1`. First write a function for bisection search. The worst case is `O(log N)` of course. Next write a second function,

`int findDictionary(int key, int *a, int N)` to find the `key` faster, using what is called, **Dictionary search**. This is based on the assumption of an almost uniform distribution of number of in the range of `min = a[0]` and `max = a[N-1]`. Dictionary search makes a better educated search for the value of `key` in the interval between `a[left]` and `a[right]` using the fraction change of the value, $0 \leq x \leq 1$:

```
x = double(key - a[left])/(double(a[right]) - a[left]);
```

to estimate the new index,

```
m =  int(left + x * (right - left)); // bisection uses x = 1/2
```

Write the function `int findDictionary(int key, int *a, int N)` for this. For a uniform sequence of numbers this is with **average** performance: `(log(log( N))`, which is much faster than `log(N)` bisection algorithm.

Implement your algorithm as a C/C++ functions. On the class GitHub there is the main file that reads input and writes output the result. You only write the required functions. Do not make any changes to the `infield` reading format. Place your final code in directory HW1. The grader will copy this and run the `makeFind` to verify that the code is correct [1]. There are 3 input files
`Sorted100.txt , Sorted100K.txt, Sorted1M.txt` for $N = 10^2, 10^5, 10^6$ respectively. You should report on the following:

`(1)The code should run for each file on the command line.`

Next for extra credit do your best to modify the code. (This modified code does not have to be turned in but as a placeholder I have copied `find.cpp` into `findGraph.cpp` which is a placeholder for the modified code. In class let's discuss together what how you did this. This will give you practice on methods need in future exercise. This code should be used to graph the performance as function of N to see this scaling behavior as we did in HW0 – much more fun that a bunch of numbers! To extend the range of sizes there is a code `makeSortedList.cpp` that can generate more sorted input lists any size N. For example you might 6 sizes: $N = 10^2, 10^4, 10^5, 10^6, 10^7, 10^8$. Then collect on Time and OpCount and graph them as function of N. You may use any graphing routine you like but in class we will discuss how to use `gnuplot` which is a basic unix tool. **(See gnuplot instructions on GitHub at** `GeneralComputingInfo/Plotting_and_Fitting` **)**, Any hack is ok to get some rough graphs. You will find there is an art to measuring these slow growth rates! If you want to automate the size you can integrate `makeSortedList.cpp` as subroutine into your code.

(In the future exercises, we will develop these basic procedure to do performance analysis by running for many values of N, averaging over many cases (e.g. *keys*) with to

[1] Note, I have introduced two different makefiles: `makeFind` and `makeGraph`. Now you can compile two different codes in HW1. That is `make -k -f makeFind` compiles `find.cpp` and `make -k -f makeGraph` compiles `findGraph.cpp` The second one is placeholder for the extra credit option below.

get mean values for Time and OpCount even including error bars and fitting routines to determine average scaling. This is an important art of engineering analysis. This require more instruction and experimentation with how to plot data and do curve fitting and even error analysis. These skills may also be useful later in the class project phase depending on what you topic you choose.)