

EC 504 – Fall 2020 – Homework 1

Written part due Thursday, Sept. 17, 2019 in the beginning of class. Coding problem due Monday Sept 21 at 11:59PM. Submit both in your directory /projectnb/alg504/yourname/HW1

Start reading Chapters 2, 3 and 4 in CLRS .

1. (20 pts)

- (a) In CLRS do Problem 3-2 on page 61.
- (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.

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

(c) Substitute

$$T(n) = c_1 n + c_2 n \log_2(n)$$

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

Extra Credit(5 pt): If you are eager do it more generally 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 .

2. (20 pts) Below you will find some functions. For each of the following functions, please provide:

- A recurrence $T(n)$ that describes the worst-case runtime of the function in terms of n as *provided* (i.e. without any compiler optimizations to avoid redundant work).
- The tightest asymptotic upper and lower bounds you can develop for $T(n)$.

(a)

```
int A(int n) {
    if (n == 0) return 1;
    else return A(n-1) * A(n-1);
}
```

(b)

```
int B(int n) {
    if (n == 0) return 1;
    if (B(n/2) >= 10)
        return B(n/2) + 10;
}
```

```

        else
            return 10;
    }
(c) int C(int n) {
    if (n == 0) return 1;
    if (n == 1) return 3;
    return D(n-1) + D(n-2)*D(n-2);
}

```

3. (20 pts) You are given n nuts and n bolts, such that one and only one nut fits each bolt. Your only means of comparing these nuts and bolts is with a function $\text{TEST}(x, y)$, where x is a nut and y is a bolt. The function returns +1 if the nut is too big, 0 if the nut fits, and -1 if the nut is too small. Design and analyze an algorithm for sorting the nuts and bolts from smallest to largest using the TEST function, such that the worst case performance of the algorithm has asymptotic complexity $O(n^2)$.

Pseudo-code.

You can also do a version of quicksort, by picking a bolt and pivoting the nut array, and similarly picking a nut and pivoting the bolt array.

4. (40 pts) **Binary search** of a large sorted array is a classic divide 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 divided 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. Graph the data to see this scaling behavior. – much more fun than a bunch of numbers! 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 CourseInformation/Plotting_and_Fitting**),

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 `infile` reading format or the `outfile` writing format in `main()` . Place your final code in directory HW1. The grader will copy this and run* the `makeFind` to verify that the

code is correct. 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.
- (2)The code should print to a file a table of the 100 random keys, execution time and the number of bisections.
- (3)And another simliar file for dictionary with "bisections" the number of dictionary sections

(In the future we use this basic procedure to some data analysis by running it for many values of N and many values of *keys* and to plot it to see as best we can the claimed scaling of $\log(N)$ and $(\log(\log(N)))$ respectively. This require more instruction in how to plot data and do curve fitting and even error analysis. These skills will be useful later in the class project phase.)