DRAFT

EC 504 – Fall 2019 – Homework 1

Due Thursday , Sept. 19, 2019 in the beginning of class. Coding problems submitted in HW1 directory on your SCC account by Friday Sept 20, 11:59PM

Background Reading in CLRS Chapters 2, 3 and 4

1. (20 pts) Place the following functions in order from asymptotically smallest to largest. When two functions have the same asymptotic order, put an equal sign between them.

$$\prod_{k=1}^{n}(1-\frac{1}{k^2}) \ , \ n^2+n^{-2} \ , \ n^{n^2} \ , \ n^{\frac{1}{n}} \ , \ \ln n \ , \ 10^{\ln(\ln n)} \ , \ xn(1+\cos(\pi n/7)), \log_{10}(n!) \ ,$$

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

   Extra Credit (5 pts): 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. If you are eager do it more generally for $T(n) = aT(n/b) + n^k$ with $k = \gamma$ and $b^\gamma = a$ assumed.

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 <= 1) return 1;
      int prod = 0;
      for (int ii = 0; ii < n; ii++)
          prod *= C((int) sqrt(n));
      return prod;
}
```

(d)
```
int D(int n) {
      if (n == 0) return 1;
      if (n== 1) return 3;
      return D(n-1) + D(n-2)*D(n-2);
}
```

(e)
```
int E(int n) {
      if (n<= 1) return 1;
      int count = 3;
      int tmp = E(n/2);
      for (int k = 0; k < n; k++)
           for (int m = 1; m <n; m*=2)
                if (tmp < exp(k+m))
                     count++;
      return E(n/2)*(count%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 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, outline in clear steps 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 $O(n^2)$.

4. (20 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 `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. In class we will discuss how to use `gnuplot` to show this behavior graphically – much more fun that a bunch of numbers!

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 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.

5. (20 points) Average performance of $O(N^2)$ search algorithm. Standard $O(N^2)$ search algorithms involve local (nearest neighbor) exchanges of element of the given list

$$A_{list} = a[0], a[1], a[2], a[3], \cdots, a[N-1] \tag{1}$$

The result is that you need to slide (or push) each of N elements into its right position exchanging it with all that are out of order. It is easy to imagine (and proven in class) that on average each of N elements need to exchange on 1/2 if the others so the on average the algorithm would have

$$\text{Number of Exchanges} \quad = \quad \frac{N(N-1)}{2} \tag{2}$$

This exercise to *numerically prove* by counting the exchanges averaged over a large set of permutation of the list few values of N. To do this you need to set up a list of N objects put it into a random order and count the exchanges. The code to do this is on GitHub. Place your final code in directory HW1 so that it is compiled with `makeRanSort` .

Make histogram of the number of exchanges for a few values of N and sample sizse. *In class suggestion will be given to make a simple histogram and graph it.* **Keep it simple!** You may pass in the histogram graph with the hard copy or put it in the HW1 directory, which ever is easier.

Extra Credit: This exercise illustrate two significant issues. #1. Average performance is rather than worse case if often more relevant in real applications: When and Why? #2. Counting operations is often a more interesting metric for an algorithm than time it: When and Why? Pass in with the written part one sentence comments on these issues.