EC 504 – Fall 2020 –1 Homework 3

Due Thursday, Oct. 8, 2020 in the beginning of class. Written and Coding problems submitted in the directory `/projectnb/alg504/yourname/HW3` on your SCC account by Thursday Oct 8, 11:59PM.

Reading Chapter 6-10 on SortingDataStructures.

1. (10 pts) CRLS Exercise 9.3-1. Note that this refers to the general Select problem with finds the i-th element in size for an unsorted list of N objects $a[N]$ That it find the element $x \in a[N]$ so that exactly i-1 are larger. You may answer this problem for the spacial case of **median** discussed in class: That is element such that there are N/2 larger elments.

   Solution: It is easiest to first review the scaling argument for 5 groups of N/5. The argument is a follow to find the meadan with complexity $T(N)$. First sort the columns N/5 columns with cost $O(N)$ and then recursively find the median for central row costing $T(N/5)$, This partitions the problem in $O(3N/10)$ smaller value than the median and $O(7N/10)$ larger than the median as in class notes or Fig. 9.1 The worst case is to have to look for the median in the larger $O(7N/10)$. So the recursion is $T(N) \leq T[N/5] + T[7N/10] + c_o N$. Try to satisfy with $T \simeq cN$ find

   $$cN \leq cN/5 + c7N/10 + c_0 = cN(1/5 + 7/10) + c_0 N = c(9/10)N + c_0 N$$

   This is satisfied and the $c_0 N$ drives it forcing $c = 9/10c + c_0$ or $c = 10c_0$ for worst case.

   Now try this for 7 groups. The same logic divides the group small vs large into $O(4N/14)$ and $O(10N/14)$ so the the recursion is $T(N) \leq T[N/7] + T[10N/14] + c_o N$ with

   $$cN \leq cN/7 + c4N/14 + c_0 = cN(2/14 + 10/14) + c_0 N = c(12/14)N + c_0 N$$

   works as before.

   But with 3 group of N/3 the split is into $O(2N/6)$ and $O(4N/6)$ so the the recursion is $T(N) \leq T[N/3] + T[4N/6] + c_o N$ so we try

   $$cN \leq cN/3 + c4N/6 + c_0 = cN(1/3 + 2/3) + c_0 N = ccN + c_0 N$$

   Can't solve $c = c + c_0$ with $c_0 > 1$.

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 D(int n) {
       if (n <= 1) return 1;
       int prod = 0;
       for (int ii = 0; ii < n; ii++)
           prod *= D((int) sqrt(n));
       return prod;
   ```

```
}
```

Solution: Given n, the code will call n versions of the function of size $\sqrt{n}$, and perform a linear cn amount of operations (multiplications, square roots) to combine the results.
The expression for run time is

$$T(n) = nT(\sqrt{n}) + cn$$

To solve this, draw a recursion tree starting from a node of size n, with n children of size $\sqrt{n}$, each of which has $n^{1/2}$ children of size $n^{1/4}$, etc. Note that the work in each node is linear in the size of the node. The number of children from level n is n, from level $n^{1/2}$ is $n * n^{1/2}$, at level $n^{1/4}$ is $n^{1+1/2+1/4}$, ...
Thus, the total amount of work at each level is equal to the number of nodes at that level times the amount of work at the node, which is linear in the size of the node. There is one node of size n, n nodes of size $n^{1/2}$, $n^{3/2}$ nodes of size $n^{1/4}$, ... This yields the following expression, summing across the different levels:

$$T(n) = cn + cn^{3/2} + cn^{7/4} + \ldots + cn^{\frac{2^k-1}{2^{k-1}}} = cn^2(\frac{1}{n} + \frac{1}{n^{1/2}} + \ldots + \frac{1}{n^{1/2^k}})$$

where $k \approx \log_2 n$ (the number of square roots required to get $n^{1/k} < 2$.)
Note that there are $\log_2(n)$ terms in the summation. Each of the terms in parentheses is less than 1, so we know that this is $O(n^2 \log_2(n))$ as an upper bound. Can we show that this is also $\Theta(n^2 \log_2(n))$? Consider that half the terms are greater than $\frac{1}{n^{\frac{1}{\sqrt{n}}}}$, and that

$$\lim_{n\to\infty} \frac{1}{n^{\frac{1}{\sqrt{n}}}} = 1$$

because, by taking logarithms, we have

$$\lim_{n\to\infty} \log \frac{1}{n^{\frac{1}{\sqrt{n}}}} = -\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = 0$$

This shows that there is a lower bound on the complexity with $\frac{\log_2(n)}{2}$ terms in the summation close to 1, so the complexity is $\Theta(n^2 \log n)$.

(b)
```
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);
}
```

Solution: As before, let's derive a recursion: There are two calls to $E(n/2)$, and there are two loops, that do $O(n \log_2(n))$ operations. Hence, the recursion is

$$T(n) = 2T(n/2) + cn \log(n)$$

Note that this does not fit any of the 3 cases of the Master's theorem! We can try enumerating the recursion tree to see that, at each level, we do the following amount of work: $cn \log(n/k)$,

where k counts the level from 1 (the call of size n) to $\log_2(n)$, the calls to the function of size 1. Hence, the total work is

$$\sum_{k=1}^{\log_2 n} cn(\log(n) - \log(k)) \in O(n \log^2(n))$$

3. (50 pts) Suppose you are given two sorted arrays A, B of integer values, in increasing order, sizes n and m respectively, and $m+n$ is an odd value (so we can define the median value uniquely). Develop an algorithm for finding the median of the combined sorted arrays, and analyze the complexity of your algorithm. It is straight forward to develop an algorithm that is $O(min(n, m))$. Explain in words. You should attempt to construct a code that is worst case complexity is $O(min(n, m))$ to get good performance.

   Implement your algorithm as function completing the code `mergeAP.cpp` in the code in `HW3_codes`. It will read in an input file form the command line. There is large input file called `input_AB.txt`

   I have provides a code `makeABlist.cpp` You can use this to generate other input files. It is recommended to make a very small input to develop and test your solution. The solution should run on `input_AB.txt` or any example that we test it on.

   Put final source code with Makefile with the output file called `input_AB.txt_out` on your top level CCS account in directory (e.g. folder!)

   `/projectnb/alg504/username/HW3`