University of Nevada, Las Vegas Computer Science 477 Spring 2023

Review 2

1. Prove that no comparison based sorting algorithm can sort n items in fewer than $\log_2(n!)$ comparisons.

2. From the previous problem, we know that five items cannot be sorted with fewer than seven comparisons. However, The actual number of comparisons needed is more than seven. Show that the median of five numbers can be found with seven comparisons.

3. Consider the following recursive function

```
int F(int n)
 {
  if(n <= 0) return 1;
  else return F(4*n/5)+F(3*n/5)+n*n;
 }
```

   (a) Write a recursive definition of $F(n)$ This recurrence is exact, not asymptotic.

   (b) Find the asymptotic complexity of $F(n)$ byjA first stating, and then solving, a recurrence. This recurrence is asymptotic, not exact.

   (c) Find the asymptotic **time** complexity of the recursive code by first stating, and then solving, a recurrence.

   (d) Using the standard dynamic program for this problem, which is bottom-up, What is the time complexity of computing $F(n)$?

   (e) What is the asymptotic time complexity of the top-down memoization method of computing $F(n)$?

Solutions:

1. The computation diagram of any comparison based algorithm is a binary tree. The height of the binary tree is the worst case number of comparisons needed. Each possible outcome of the problem must correspond to at least one leaf of the tree. The height of a binary tree is no less than the base two logarithm of the number of leaves.

   A sorting algorithm computes a permutation of the sorted items, and that number is n, the number of permutations is $n!$, hence the computation tree must have at least $n!$ leaves, hence height at least $\log_2(n!) = \Theta(n \log n)$.

2. I will postpone giving the answer to this one.

3. (a) $F(n)$ is defined by the recursion:

$$F(n) = \begin{cases} 0 \text{ if } n \leq 0 \\ F(4n/5) + F(3n/5) + n^2 \text{ if } n > 0 \end{cases}$$

   (b) $F(n) = F(4n/5) + F(3n/5) + n^2$.

   Use the Akra-Bazzi method. Let γ be the solution to the equation $(4/5)^\gamma + (3/5)^\gamma = 1$. Then $\gamma = 2$, hence $F(n) = \Theta(n^\gamma \log n) = \Theta(n^2 \log n)$.

(c) Let $T(n)$ be the time executed by the computation of $F(n)$, using the recursive code. It takes $O(1)$ time to compute n^2, hence the recurrence is

$T(n) = T(4n/5) + T(3n/5) + 1$

We use the Akra-Bazzi method again, and $\gamma = 2$, Thus $T(n) = \Theta(n^2)$.

(d) The standard (bottom-up) DP computation is

```
F[0] = 1;
for(int i = 1; i <= n; i++)
  F[i] = F[4*i/5] + F[3*i/5] + n*n;
cout << F[n];
```

The time complexity of this computation is $\Theta(n)$.

(e) Not all possible subproblems are used in the computation, and unlike the bottom-up technique, only the needed subproblems are needed. Memos of the form $(m, F(m))$ will be computed and stored for only those which are needed. The reason we work top-down is that it is not always practical to determine which memos are needed when we work bottom-up.

Since each m is achieved by multiplying n by some combination of $4/5$ and $3/5$ in some order, Thus, each m should be of the form $3^i 4^i n/5^{i+j}$ for integers i, j. The choices of i, j are bounded by the fact that $3^i 4^j n^{i+j}/5^{i+j}$ cannot be less than 1. There are $\Theta(\log^2 n)$ such choices of i, j. The time complexity is proportional to the number of memos stored, namely $\Theta(\log^2 n)$.

Results of the Program. I wrote a program implementing the memoization technique for this problem, using $n = 1000$. The number of memos created was 101, which is close to $\log^2 n$. The result was $F(1000) = 16610822$, which is close to $n^2 \log n$.