CS124 Homework 2 Out: Feb. 2, 2017
Due: Feb. 10, 2017

For all homework problems where you are asked to give an algorithm, you must prove the correctness of your algorithm and establish the best upper bound that you can give for the running time. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail. As always, try to make your answers as clear and concise as possible.

1. Buffy and Willow are facing an evil demon named Stooge, living inside Willow's computer. In an effort to slow the Scooby Gang's computing power to a crawl, the demon has replaced Willow's hand-designed super-fast sorting routine with the following recursive sorting algorithm, known as StoogeSort. For simplicity, we think of Stoogesort as running on a list of distinct numbers. StoogeSort runs in three phases. In the first phase, the first 2/3 of the list is (recursively) sorted. In the second phase, the final 2/3 of the list is (recursively) sorted. Finally, in the third phase, the first 2/3 of the list is (recursively) sorted again.

   Willow notices some sluggishness in her system, but doesn't notice any errors from the sorting routine. This is because StoogeSort correctly sorts. For the first part of your problem, prove rigorously that StoogeSort correctly sorts. (Note: in your proof you should also explain clearly and carefully what the algorithm should do and why it works even if the number of items to be sorted is **not** divisible by 3. You may assume all numbers to be sorted are distinct.) But StoogeSort can be slow. Derive a recurrence describing its running time, and use the recurrence to bound the asymptotic running time of Stoogesort.

2. (Part A) Solve the following recurrences exactly, and then prove your solutions are correct by induction. (Hint: Graph values and guess the form of a solution: then prove that your guess is correct.)

   - $T(1) = 1, T(n) = T(n-1) + 4n - 4$.
   - $T(1) = 1, T(n) = 2T(n-1) + 2n - 1$.

   (Part B) Give asymptotic bounds for $T(n)$ in each of the following recurrences. Hint: You may have to change variables somehow in the last one.

   - $T(n) = 4T(n/2) + n^3$.
   - $T(n) = 17T(n/4) + n^2$.
   - $T(n) = 9T(n/3) + n^2$.
   - $T(n) = T(\sqrt{n}) + 1$.

3. Explain how to solve the following two problems using heaps. (No credit if you're not using heaps!) First, give an $O(n \log k)$ algorithm to merge k sorted lists with n total elements into one sorted list. Second, say that a list of numbers is k-close to sorted if each number in the list is less than k positions from its actual place in the sorted order. (Hence, a list that is 1-close to sorted is actually sorted.) Give an $O(n \log k)$ algorithm for sorting a list of n numbers that is k-close to sorted.

4. Design an efficient algorithm to find the *longest* path in a directed acyclic graph. (Partial credit will be given for a solution where each edge has weight 1; full credit for solutions that handle general real-valued weights on the edges, including *negative* values.)

5. Giles has asked Buffy to optimize her procedure for nighttime patrol of Sunnydale. (This takes place sometime in Season 2.) Giles points out that proper slaying technique would allow Buffy to traverse all of the streets of Sunnydale in such a way that she would walk on each side of each street, exactly once, going up the street in one direction and down the street in the other direction. Buffy now has slayer homework: how can it be done? (If you have to assume anything about the layout of the city of Sunnydale, make it clear!)

6. Consider the shortest paths problem in the special case where all edge costs are non-negative integers. Describe a modification of Dijkstra's algorithm that works in time $O(|E|+|V|m)$, where m is the maximum cost of any edge in the graph.

7. (Note: this problem is somewhat difficult.) We found that a recurrence describing the number of comparison operations for a mergesort is $T(n) = 2T(n/2)+n-1$ in the case where n is a power of 2. (Here $T(1) = 0$ and $T(2) = 1$.) We can generalize to when n is not a power of 2 with the recurrence

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n - 1.$$

   Exactly solve for this more general form of $T(n)$, and prove your solution is true by induction. (Additional note: you will not receive full credit if your answer is in the form of a summation over say n terms. We are looking for a "closed form" answer.) Hint: plot the first several values of $T(n)$, graphically. What do you find? You might find the following concept useful in your solution: what is $2^{\lceil \log_2 n \rceil}$?

8. **Do not turn this in. This is a suggested exercise.** This exercise will be based on the 2SAT problem. The input to 2SAT is a logical expression of a specific form: it is the conjunction (AND) of a set of clauses, where each clause is the disjunction (OR) of two literals. (A literal is either a Boolean variable or the negation of a Boolean variable.) For example, the following expression is an instance of 2SAT:

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1}).$$

   A solution to an instance of a 2SAT formula is an assignment of the variables to the values T (true) and F (false) so that all the clauses are satisfied– that is, there is at least one true literal in each clause. For example, the assingment $x_1 = T, x_2 = F, x_3 = F, x_4 = T$ satisfies the 2SAT formula above.

   Derive an algorithm that either finds a solution to a 2SAT formula, or returns that no solution exists. Please carefully give a complete description of the entire algorithm and the running time.

   (Hint: Reduce to an appropriate problem. It may help to consider the following directed graph, given a formula I in 2SAT: the nodes of the graph are all the variables appearing in I, and their negations. For each clause $(\alpha \vee \beta)$ in I, we add a directed edge from $\overline{\alpha}$ to β and a second directed edge from $\overline{\beta}$ to α. How can this be interpreted?)

9. **Do not turn this in. This is a suggested exercise.** Give a complete proof that $\log(n!)$ is $\Theta(n \log n)$. Hint: you should look for ways to bound $(n!)$. Fairly loose bounds will suffice.