Discrete Structures. CSCI-150. Summer 2015.

Homework 6. Due Mon. Jun 22, 2015.

Problem 1

In this problem, we would like to prove by induction that the number of diagonals in a convex polygon is equal to $\frac{n(n-3)}{2}$.

Let $D(n)$ be the number of diagonals of a convex n-sided polygon.

(a) First, demonstrate that $D(3) = 0$. In addition, you also can show that $D(4) = 2$ and $D(5) = 5$.

(b) Then show that for for convex polygons with $n \geq 3$ sides: $D(n+1) = D(n) + n - 1$.

(c) Using these results, prove by induction that $\forall n \geq 3$:

$$D(n) = \frac{n(n-3)}{2}$$

Problem 2 (Graded)

Given the recurrence

$$\begin{aligned} S(0) &= 0, \\ S(n+1) &= 3S(n) + 1, \end{aligned}$$

prove by induction that for all $n \geq 0$:

$$S(n) = \frac{3^n - 1}{2}.$$

Problem 3 (Graded)

Given the recurrence

$$\begin{aligned} T(1) &= 2, \\ T(n) &= T(n-1) + 2n \qquad (\text{for } n > 1), \end{aligned}$$

first, find the closed form expression for $T(n)$. You may apply the method we used in class, where we repeatedly substitute $T(n)$ in terms of $T(n-1)$, then $T(n-1)$ in terms of $T(n-2)$, and so on, eventually identifying the pattern. This method is also described in Lehman and Leighton's book (p.147), where it is called "Plug-and-Chug" method.

After that, prove by induction that the closed form expression you've found is correct.

Problem 4 (Graded)

Solve the linear recurrence (for $n \geq 0$)

$$f(0) = 1, \quad f(1) = -1,$$
$$f(n) = f(n-2).$$

Problem 5 (Graded)

Solve linear recurrence

$$f(0) = 3, \quad f(1) = 1,$$
$$f(n) = 4f(n-1) + 21f(n-2).$$

Problem 6. We have not learned yet, how to deal with the roots of multiplicity two, so you can wait until Monday before trying to solve this recurrence.

First, verify that $x^3 - 3x^2 + 4 = (x^2 - 4x + 4)(x + 1)$.

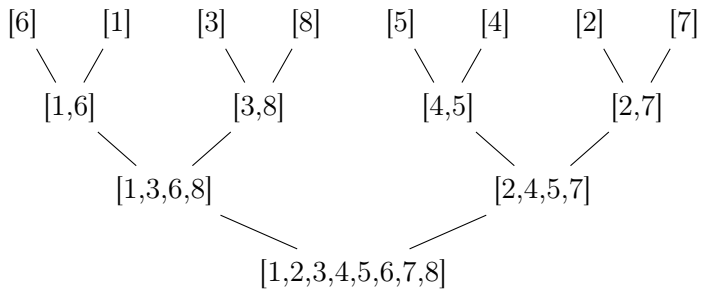
Then, solve the linear recurrence

$$f(0) = 1, \quad f(1) = 0, \quad f(2) = 14,$$
$$f(n) = 3f(n-1) - 4f(n-3).$$

Problem 7. Try it only if you have time.

Conceptually, the merge sort algorithm consists of two stages:

1. In the first stage, the input list is broken down into small lists of unit length. It takes 0 comparisons, so this stage does not contribute anything to the total time complexity of the algorithm.
2. In the second stage, we merge the lists:

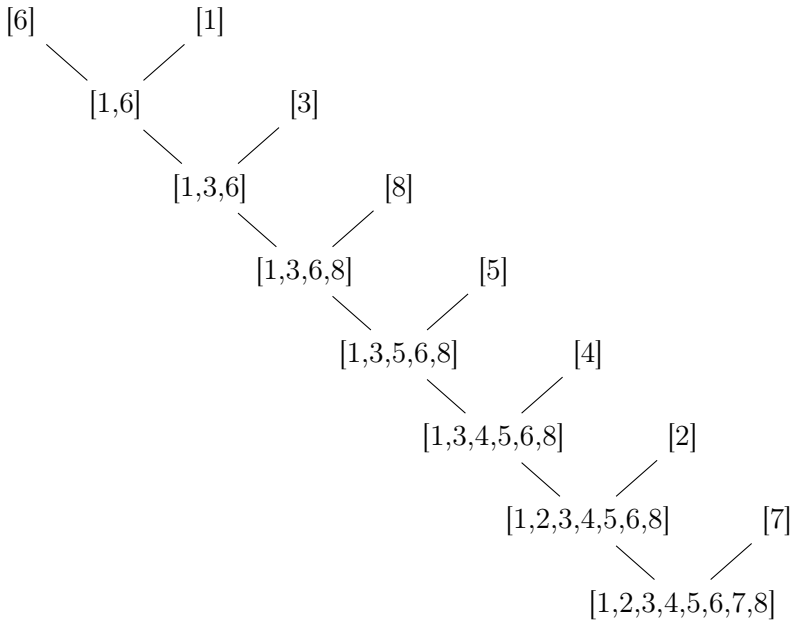


The time complexity of merging. When merging two lists l_1 and l_2 with the procedure we discussed in class, the number of comparisons is at most

$$\text{lenght}(l_1) + \text{length}(l_2) - 1.$$

Using this fact, we proved that, in the worst case, the merging stage takes $T(n) = n \log_2 n - n + 1$ comparisons.

Consider a similar sorting algorithm, call it INCREMENTAL SORT. It is using the same merge function, but the input is split differently. Because of that, on the input $[6, 1, 3, 8, 5, 4, 2, 7]$, the lists are merged in the following order:



(a) What is the worst input for this algorithm? That is, suggest an input such that the algorithm makes the maximum number of comparisons.

(b) In the worst case, what is the number of comparisons for this algorithm?

(c) Discuss the results. Compare the new algorithm with the merge sort.

(d) How would you implement INCREMENTAL SORT? We know that the merge function is the same as before. Try to define the splitting function. (You don't have to write actual code, explain how it works).