For the *mergeSort* D&C algorithm, we obtained the following recurrence relation:

$$T_r(\hat{n}) = \begin{cases} 1 & n = 1 \\ 2T_r\left(\frac{\hat{n}}{2}\right) + \hat{n} + 1 & n > 1 \end{cases}$$

we also, obtained:

$$T_c(\hat{n}) = \hat{n}\,(\lg\hat{n}) + 2\hat{n} - 1$$

where $\hat{n} = 2^k$

- **Example 65.** Prove $T_r(\hat{n}) = T_c(\hat{n})$.

- **Example 66.** In a few recent proofs, we used the fact that $\lg n$ is monotonically increasing. Let's prove it here. Prove $\lg(n-1) < \lg n \quad \forall n > 1 \in \mathbb{N}$.

- **Example 67.** Demonstrate how to obtain the time complexity of *binSearch* using the approach we used in class for *mergeSort*.

- **Example 68.** Demonstrate how to obtain the time complexity of *mergeSort* using the approach we used in class for *binarySearch*.

- **Example 69a.** In this week's lecture, we proved that $T_{mergeSort}(n) \in \Omega(n\lg n)$ without using induction. As yet another practice on induction, prove $T_{mergeSort}(n) \in \Omega(n\lg n)$ by induction.

- **Example 69b.** In this week's lecture, there is a non-inductive proof for $T_{mergeSort}(n) \in O(n\lg n)$. As yet another practice on induction, prove $T_{mergeSort}(n) \in O(n\lg n)$ by induction.

- **Example 70.**

  Part A) Develop a divide and conquer algorithm similar to *mergeSort*, but let's call it *merge3Sorted*, in which a large list splits to 3 sublists and each sublist splits to 3 sublists until there is only one element in the sublist (note that a sublist with one element is considered sorted). Then, sorted sublists merge together to make a bigger sorted list.

  Part B) Provide a recurrence relation for the time complexity of your *merge3Sorted*.

  Part C) Using either the approach that we used for *binSearch* or the one we used for *megeSort*, demonstrate what the time complexity of *merge3Sorted* is.

  Note 1. You can find a fair comparison of these approaches here.

  Note 2. In the closed form you find in Part B, the base of *log* is perhaps 3.