Sreeram Danda
908 190 4576

#8a:

Base Case $P(1)$:

$n = 1$. The array only has one element so $j = i$. The algorithm correctly returns the original array. (A one element array is always sorted).

Inductive Case $(\forall(x < y)\ P(x) \Rightarrow P(y))$:

Inductive Hypothesis: The algorithm outputs correctly for any array of size less than *n*.

$n > 1$. Consider the two following cases:

Case #1: $n = 2$

If there are two elements in the array, $j = i + 1$ so the algorithm performs a swap of the two elements if the first element is larger than the second element. This operation would result in a correctly sorted array of two elements.

Case #2: $n > 2$

Let $k = \lfloor (j - i + 1)/3 \rfloor$.

The algorithm then makes three recursive calls:

SuperSort $(A,\ i,\ j - k)$

SuperSort$(A,\ i + k,\ j)$

SuperSort$(A,\ i,\ j - k)$

Since each recursive call uses a sub-array of size j - k. Each call is strictly less than the size of the original array. By the inductive hypothesis we can assume that each recursive call returns the correct output. That is that each recursive call returns a sorted array of size j - k. First for elements *i* to *j-k*. Then elements *i+k* to *j.* Then again for elements *i* to *j-k* to account for elements in the third portion that may have lower values in comparison to elements in the first portion.

$\therefore$ By induction the algorithm returns correctly an array of sorted elements. ■

#8b:

Consider this recursive relation where *n* represents the number of elements in the array:

$$T(n) = 3T(2n/3) + f(n)$$

The work done at any given level is the two comparisons done to check if *i = j* and if *j* = 1. Each of these comparisons take a constant amount of work, C which is done at every level so we can rewrite our recurrence relation as the following:

$$T(n) = 3T(2n/3) + Cn$$

$$T(n) = 3T(2n/3) + O(n)$$

Using the master theorem we note the following:

$$a = 3,\ b = 3,\ d = 1$$

$$d = log_3 3$$

Therefore, the asymptotic number of comparisons made by the algorithm is: $O(n(log 2n))$