D0012E Lab2 Part 2 Grupp 11:

Elias Grundberg	eligru-3@student.ltu.se	010314
Hjalmar Norén	hjanor-3@student.ltu.se	041029
Pablo Zepada Garcia	pabzep-3@student.ltu.se	021107

2024-12-03

Description:

We have an array A of n = 4^k elements and need to sort it like this:

- If n <= 4 then sort A with insertion sort and return the array (Base Case)
- Sort the first $\frac{3n}{4}$ elements recursively
- Sort the last $\frac{3n}{4}$ elements recursively
- Sort the first $\frac{3n}{4}$ elements recursively
- Return the sorted array

Proof:

We solve this algorithm by using induction. In our hypothesis we assume that the algorithm will correctly sort any array of length n, where n is a power of 4 following the recursive steps described above. Let's assume that the algorithm works for arrays of size $\frac{3n}{4}$, in each recursive step the array is divided into smaller subarrays of size $\frac{3n}{4}$, $\frac{3n}{4}$ and $\frac{3n}{4}$. Each of these smaller subarrays are sorted recursively and the last merging steps ensure that the array is sorted.
Each of these subarrays are handled on their own in recursive calls. According to the hypothesis, each of these calls guarantee its subarray is correctly sorted.
The overlapping nature ensures that any mistake in the sorting will be corrected in the final step. The algorithm maintains the overall sorting invariance since the subarrays are sorted recursively and merged appropriately. For the base case where n = 4 the algorithm obviously works since insertion sort is known to work correctly for small arrays.

Number of Comparisons:

For the base case where n <= 4 the algorithm will use insertion sort which will require at most $O(n^2)$ comparisons which can be explained as: $T(4) = \Theta(4^2)=O(16)$
For the recursive case the algorithm splits the array into three parts of size $\frac{3n}{4}$ and sorts them recursively. The total number of comparisons is the sum of comparisons of each subarray and the comparisons required to merge them and can be expressed like: $T(n) = 3T(3n/4) + O(n)$. Using the master theorem we get 3, $\frac{4}{3}$ and 1

and since 3 is larger than $\frac{4}{3}$1 it means that the complexity is determined by the recursive term, the solution then becomes: $T(n) = \Theta(n^{\log_{\frac{4}{3}} 3})$ or roughly $\Theta(n^{3.8})$