Algorithms and data structures – sheet 4

Valdrin Smakaj

Problem 4.1 *Merge Sort* (12 points)

(a) (4 points) Implement a variant of Merge Sort that does not divide the problem all the way down to subproblems of size 1. Instead, when reaching subsequences of length k it applies Insertion Sort on these n/k subsequences.

(b) (3 points) Apply it to the different sequences which satisfy best case, worst case and average case for different values of k. Plot the execution times for different values of k.

(c) (3 points) How do the different values of k change the best-, average-, and worst-case asymptotic time complexities for this variant? Explain/prove your answer.

(d) (2 points) Based on the results from (b) and (c), how would you choose k in practice? Briefly explain.

~~~~~~Answers~~~~~~

Answers to sub-problem (a and b) are solved by code in the main.cpp file included in the folder I submitted.

c) We consider the cases for worst, best and average.

- Worst case ➔ considering the time complexity for insertion sort for worst case which is $\Theta(k^2)$. Since we have to sort n/k sublists and then merge them, the time complexity tends in a different form which is represented as $\Theta(k^2 \cdot \frac{n}{k}) = \Theta(nk)$. Then the overall merging formula is $\Theta(n \log(\frac{n}{k}))$. Now if we optimize the merge and insert sort time complexities the formula tends:

  $\Theta(nk + n\log(\frac{n}{k})) = \Theta(n \log n)$. Assuming $k = \Theta(\log n)$:

$$\begin{aligned}\Theta(nk + n\log(\frac{n}{k})) &= \Theta(nk + n\log n - n\log k)\\ &= \Theta(n\log n + n\log n - n\log(\log n))\\ &= \Theta(2n\log n - n\log(\log n))\\ &= \Theta(n\log n)\end{aligned}$$

- Best case ➔ From the graph we can observe and conclude that with higher values of k the algorithm is getting faster because the time complexity of insertion sort is smaller (linear) than the one of merge sort.
- Average case ➔ it is slightly decreasing compared to the worst case for a small factor since at least 1 element can be in its position in the rarest case but generally some can be in their right place. $\Theta(n^2)$.
~~~~~~

d) From my perspective, analyzing the time complexities using the graphs with values provided, with a given size k from 10 to 1000, I'd say to pick k value around 400. So Pretty much for general sizes I think that the best way or sort to say size of k to pick is half of the overall size (n/2) or (n/2 – x) for x being a small factor which lowers the half size of the array a bit (saying this based on my result of getting 400 as approximately best result out of size 1000 ➔ 1000/2 – x = 400 ➔ x=100)➔ just a specific case I got to calculate but I think that the overall idea flows through this concept point.

But overall the idea is that the new modified one has to be less in time complexity than the merge sort one. Thinking of that and by the formula we got on the c) point we can see that k has to be less than $\Theta(\log n)$ or $\log_2(n)$.

Problem 4.2 *Recurrences* (10 points)

Use the substitution method, the recursion tree, or the master theorem method to derive upper and lower bounds for $T(n)$ in each of the following recurrences. Make the bounds as tight as possible. Assume that $T(n)$ is constant for $n \leq 2$.

(a) (2 points) $T(n) = 36T(n/6) + 2n$,

(b) (2 points) $T(n) = 5T(n/3) + 17n^{1.2}$,

(c) (2 points) $T(n) = 12T(n/2) + n^2 \lg n$,

(d) (2 points) $T(n) = 3T(n/5) + T(n/2) + 2^n$,

(e) (2 points) $T(n) = T(2n/5) + T(3n/5) + \Theta(n)$.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Answers~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~