
You're expected to work on the problems before coming to the lab. Discussion session is not meant to be a one-way lecture. The TA will lead the discussion and correct your solutions if needed. For many problems, we will not release 'official' solutions. If you're better prepared for discussion, you will learn more. The TA is allowed to give some bonus points to students who actively engage in discussion and report them to the instructor. The bonus points earned will be factored in the final grade.

1. (basic) Consider the pseudo-code of Merge-Sort with line 1 removed. What goes wrong?

2. (advanced) Let's slightly tweak Merge Sort. Instead of partitioning the array into two subarrays, we will do into three subarrays of an (almost) equal size. Then, each recursive call will sort each subarray, and we will merge the three sorted subarrays. Assuming that merging can be done in $O(n)$ time, write a recurrence for the running time of this new version of Merge Sort, and solve it.

   Sol. $T(n) = 3T(n/3) + O(n)$. $T(n) = O(n \log n)$.

3. (basic) Solve $T(n) = 4T(n/2) + \Theta(n)$ using the recursion tree method. Cleary state the tree depth, each subproblem size at depth d, the number of subproblems/nodes at depth d, workload per subproblem/node at depth d, (total) workload at depth d.

   Sol. The tree visualization is omitted.
   For simplicity, drop Θ and $T(1) = 1$; there are hidden constants c_1 and c_2 such that the function is sandwiched by $c_1 n$ and $c_2 n$, but we can ignore such constants n the asymptotic world. Make sure you specify all the following key quantities to be 100% safe.
   Tree depth $D = \log_2 n$.
   Each subproblem size at depth d: $n/2^d$.
   Number of nodes at depth d: 4^d
   WL per node at depth d: $n/2^d$
   WL at depth d: $2^d n$.
   So, $\sum_{d=0}^{D} 2^d n = \Theta(2^D n) = \Theta(n^2)$. (bottom level dominates).

4. (basic) Solve $T(n) = 4T(n/2) + \Theta(n^2)$ using the recursion tree method.

   Sol. The tree visualization is omitted.
   Tree depth $D = \log_2 n$.
   Each subproblem size at depth d: $n/2^d$.
   Number of nodes at depth d: 4^d
   WL per node at depth d: $(n/2^d)^2$
   WL at depth d: n^2.
   So, $\sum_{d=0}^{D} n^2 = \Theta(n^2 \log n)$. (All levels are equally important).

5. (basic) Solve $T(n) = 4T(n/2) + \Theta(n^3)$ using the recursion tree method.

   Sol. The tree visualization is omitted.
   Tree depth $D = \log_2 n$.
   Each subproblem size at depth d: $n/2^d$.
   Number of nodes at depth d: 4^d
   WL per node at depth d: $(n/2^d)^3$
   WL at depth d: $n^3/2^d$.
   So, $\sum_{d=0}^{D} n^3/2^d = \Theta(n^3)$. (top level dominates).

6. (basic) Solve the above recursions using the Master Theorem. You must specify which case and how you set the variables such as a, b, ϵ.

   Sol.

   $T(n) = 4T(n/2) + \Theta(n)$: Case 1. $a = 4$ and $b = 2$, and $f(n) = \Theta(n) = O(n^{\log_2 4 - \epsilon})$, where you can choose any $\epsilon > 0$ as long as $0 < \epsilon \leq 2$. Therefore, we have $T(n) = \Theta(n^2)$
   $T(n) = 4T(n/2) + \Theta(n^2)$: Case 2. $a = 4$ and $b = 2$, and $f(n) = \Theta(n^2) = \Theta(n^{\log_2 4})$. Therefore, we have $T(n) = \Theta(n^2 \log n)$
   $T(n) = 4T(n/2) + \Theta(n^3)$: Case 3. $a = 4$ and $b = 2$, and $f(n) = \Theta(n^3) = \Omega(n^{\log_2 4 + \epsilon})$, where $\epsilon = 1$ (any choice of ϵ works as long as $0 < \epsilon \leq 4$). Further, $af(n/b) = 4(n/2)^3 \leq (1/2)n^3 = cn^3$; here $c = 1/2$. Therefore, we have $f(n) = \Theta(n^3)$.

7. (basic/intermediate) Solve $T(n) = 2T(n/2) + 1$ using the substitution method. You just need to give an asymptotic upperbound (preferably tight, though).

8. (basic/intermediate) In the Max-Subarray problem, explain how to compute $\max_{low \leq i \leq mid < j \leq high} S[i, j]$ in $O(high - low)$ time. Here, $S[i, j]$ is defined as $A[i] + A[i + 1] + \ldots + A[j]$.

9. (basic) In the Max-Subarray problem, state the recurrence on the running time of the algorithm you learned in class and solve it using your favorite method.

10. (basic) We learned a divide-and-conquer-based algorithm for Matrix multiplication (not Strasssen's algorithm) for multiplying two n by n matrices. The recurrence on the running time of the algorithm was $T(n) = 8T(n/2) + O(n^2)$ – can you explain why? Solve the recurrence.

11. (basic) In Strassen's algorithm, what was the recurrence we obtained? Solve the recurrence.

12. (Very Challenging) The Restricted-Max-Sub-Array problem, we are given another parameter k as input. Now the goal is to find i, j such that $1 \leq i \leq j \leq \min\{i + k, n\}$ that maximizes $A[i] + A[i + 1] + \cdots + A[j]$. In other words, we have another constraint that i and j differ by at most k; note that this problem is the original Max-Sub-Array problem if $k \geq n - 1$. Modify the divide-and-conquer algorithm for the Max-Sub-Array problem to derive an $O(n \log n)$ time algorithm for the Restricted-Max-Sub-Array problem.

    Sol. The crossing is where we have to make changes. Kind of DP.