Lecture 5: Divide and conquer

We study the merge sort algorithm in class. Refer to Section 2.3 for more details. Here are some key points about divide and conquer. You should decide how to divide a problem into smaller sub-problems. Then, you will solve these smaller sub-problems recursively. Finally, you combine these subproblems to get the answer for the original problem.

For the merge sort, its running time can be written as \( T(n) = 2T(n/2) + \Theta(n) \), when \( n > 1 \). When \( n = 1 \), \( T(n) = \Theta(1) \). There are several methods to estimate \( T(n) \). The first is called direct substitution. This method will repetitively get rid of the sub-problems by substituting with even smaller sub-problems. Here, \( T(n) = 2T(n/2) + cn \), \( T(n/2) = 2T(n/4) + cn/2 \), \( T(n/4) = 2T(n/8) + cn/4 \), and so on. Then, we get rid of \( T(n/2) \) term by substituting it with \( 2T(n/4) + cn/2 \), and so on. This will lead to \( T(n) = nT(1) + cn \log n \), because we stop after \( \log n \) substitutions and we get sub-problems of size 1. If you are unclear about this step, you should write down these recurrences and try it yourself.

A related approach is to construct the so-called recursion tree. The recursion tree is divided into levels. Nodes of a level are labeled with the work spent on this level (i.e. the divide and combine work). The conquer portion of the work is expressed its descendant nodes. For the merge sort, each node for problem size of \( n/2^k \) has work \( cn/2^k \). There are \( \Theta(\log n) \) levels in the tree, because each time we go down the tree by one level, the number of subproblems doubles. At the bottom, there are \( n \) subproblems (because in merge sort, the subproblems are disjoint).

A third approach of analyzing recurrences is to use the Master theorem. Refer to Section 4.3 for more details. Two things to remember: remember the small \( \epsilon \) in cases 1 and 3, and also the regularity condition for case 3.

Lecture 6: Divide and conquer (continued)

Our first example is the MaxProfit problem, where you are given \( n \) days' stock price \( P[1..n] \) and you want to make the most profit by buying a fixed number of shares at day \( b \) and later selling at day \( s \).

A naive algorithm would try all \( \binom{n}{2} \) pairs of days, which leads to an \( O(n^2) \) algorithm. We now use divide and conquer. We define \( \text{MaxProfit}(l, r) \) as the maximum profit we can earn by buying and selling within the period of \([l, r]\). Note we must first buy then sell. Then, the solution is simply \( \text{MaxProfit}(1, n) \). Here is the algorithm.

1. \( \text{MaxProfit}(l, r) \)
2. \( \text{if } l = r \) then
3. \( \quad \text{return } 0. \)
4. \( \text{end if} \)
5. \( m = [(l + r)/2] \)
6. \( m_1 \leftarrow \text{MaxProfit}(l, m) \)
7. \( m_2 \leftarrow \text{MaxProfit}(m + 1, r) \)
8. \( v_1 \leftarrow \text{MIN}(P[l], P[l + 1], \ldots, P[m]) \)
9. \( v_2 \leftarrow \text{MAX}(P[m + 1], P[m + 2], \ldots, P[r]) \)
10. \( \text{return } \text{MAX}(m_1, m_2, v_2 - v_1). \)

This algorithm is correct because lines 6 and 7 cover the cases where you buy and sell within the first (resp. second) half of the problem, while line 8 covers the case when you buy before day \( m \) and sell after day \( m \). The running time \( T(n) = 2T(n/2) + \Theta(n) \), which leads to \( O(\log n) \) time. Note, the combine step takes \( \Theta(n) \) to find \( v_1 \) and \( v_2 \).

Our next problem is on multiplying two \( n \) by \( n \) square matrices \( A \) and \( B \). We apply divide and conquer. See Section 28.2 for more details. One comment about analyzing the first divide and conquer algorithm. The combine step takes \( \Theta(n^2) \). This is because it involves adding up a constant number of \( n/2 \) by \( n/2 \) matrices (which takes \( \Theta(n^2) \)) time, and then copy these four resulting \( n/2 \) by \( n/2 \) matrices \( C_{i,j} \) into the \( n \) by \( n \) matrix \( C \), which also take \( \Theta(n^2) \) since there are \( \Theta(n^2) \) cells in \( C \) and each copying takes \( O(1) \) time. Thus, \( T(n) = 8T(n/2) + \Theta(n^2) \). We use the Master theorem to find out \( T(n) = \Theta(n^3) \), which is no better than the original naive algorithm.