COMP3121/9101
Algorithm Design

Tutorial 2
Divide and Conquer

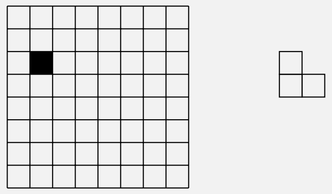
Before the Tutorial

Before coming to the tutorial, try and answer these discussion questions yourself to get a firm understanding of how you're pacing in the course. No solutions to these discussion points will be officially released, but feel free to discuss your thoughts on the forum. Your tutor will give you some comments on your understanding.

- Algorithms that use divide and conquer tend to involve three main steps. Briefly describe each of these steps.
  - **Divide**, **Conquer**, **Combine**.
- Identify these steps in the Merge Sort example in lectures. Remember to discuss how these steps are linked to obtain the overall algorithm.
- How can you confirm the correctness of a divide and conquer algorithm? Use Merge Sort (from Lecture 1) as an example.
- Recall the statement of the Master Theorem. Briefly discuss what the three cases of the Master Theorem entail, and what the conditions are for each case.
- Read through the problem prompts and think about how you would approach the problems.

Tutorial Problems

Problem 1. Let n be a power of two. You are given an $n \times n$ board with one of its cells missing (i.e., the board has a hole). The position of the missing cell can be arbitrary. You are also given a supply of "trominoes", each of which can cover three cells as below.



(a) Design a divide and conquer algorithm to fill the $n \times n$ board (with the missing cell) with trominoes.
  - Be sure to discuss what the divide, conquer, and combine steps are.
  - How do each of these steps together construct the overall algorithm?
  - It might help to look at small values of n, and use previous constructions to build your algorithm.

(b) Justify the correctness of your algorithm.

- What is the base case? Does your algorithm correctly solve the base case?
- If smaller instances are solved correctly, does the "combine" step ensure the correctness for the parent instance?

(c) Analyse the time complexity of your algorithm using the Master Theorem.

Problem 2. An element x is a *majority element* of A if x appears more than $n/2$ times in A. We will design an algorithm using divide and conquer to find x (if it exists). You cannot assume that the elements in A can be sorted (i.e. you cannot answer queries of the form "is $A[i] > A[j]$?"); however, you can answer queries of the form "is $A[i] = A[j]$?". You may also assume that n is a power of two.

(a) If a majority element exists, how many elements *could* be a majority element?

(b) Let A and B be two arrays containing $n/2$ elements. Let C be the array obtained when we merge A and B together.

- If a majority element of A and B is the same, what can you say about a majority element of C?
- If neither arrays contain a majority element, what can you say about a majority element of C?
- If both arrays contain different majority elements, how can we check if a majority element exists in C?

(c) Hence, design a divide and conquer algorithm to find a majority element.

- Be sure to discuss the divide, conquer, and combine steps.
- If smaller instances are solved correctly, how should you combine the solutions together to solve the original problem?

(d) Justify the correctness of your algorithm.

(e) Analyse the time complexity of your algorithm using the Master Theorem.

Problem 3. Consider the recurrence $T(n) = a \cdot T(n/b) + f(n)$. When working with recurrences like $T(n)$, we can view them as recursion trees. At each level, we perform $f(n)$ amount of work and then divide the problem into a subproblems, each of size n/b.

Construct the recursion tree for each of the recurrences. Analyse the overall time complexity and compare this with the result you obtain from the Master Theorem.

- Be sure to keep track of the amount of work required at each level.
- Does the number of leaves grow faster than the $f(n)$? What does this tell you about the overall time complexity?

(a) $T(n) = 4T(n/2) + n$.

(b) $T(n) = 4T(n/2) + n^2$.

(c) $T(n) = 4T(n/2) + n^3$.

After the Tutorial

After your allocated tutorial (or after having done the tutorial problems), review the discussion points. Reflect on how your understanding has changed (if at all).

- In your own time, try and attempt some of the practice problems marked [K] for further practice. Attempt the

[H] problems once you're comfortable with the [K] problems. All practice problems will contain fully-written solutions.

- If time permits, try and implement one of the algorithms from the tutorial in your preferred language. How would you translate high-level algorithm design to an implementation setting?