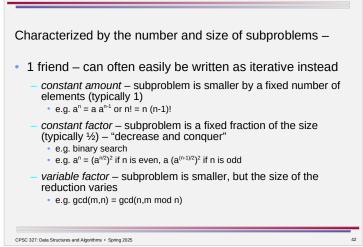
Developing Algorithms

Strategies -

- realize your problem is another well-known problem in disguise
 - it is searching or sorting
 - there's a data structure for that
 - it is a graph problem
- develop a new algorithm
 - divide and conquer
 - series of steps iterative
 - series of choices greedy, backtracking, branch and bound, dynamic programming

Recursive Patterns

CPSC 327: Data Structures and Algorithms . Spring 2025



Algorithmic Structures

Iterative algorithms proceed forward towards the solution one step at a time.

Recursive algorithms have friends solve subproblems.

- construct a complete solution out of complete solutions for smaller subproblems
 - induction lets you demonstrate that the solution for the bigger problem is correct
- base case defines when you stop
 - making progress ensures that you will get there (recursion will terminate)

CPSC 327: Data Structures and Algorithms • Spring 2025

Recursive Patterns

Characterized by the number and size of subproblems -

- 2+ friends
 - divide-and-conquer split into $b \ge 2$ subproblems of size n/b (b is typically 2)
 - case analysis each friend gets a subproblem resulting from a different alternative

CPSC 327: Data Structures and Algorithms • Spring 2025

How to Design Algorithms

- establish the problem
- identify avenues of attack
- define the algorithm
- show termination and correctness
- determine efficiency

CPSC 327: Data Structures and Algorithms . Spring 2025

How to Design (Divide-and-Conquer) Algorithms

Identify avenues of attack. Determine the algorithmic approach(es) to try.

• Targets.

Identify any applicable time and space requirements for your solution. This might be stated as part of the problem ("find an $O(n \log n)$ algorithm"), come from having an algorithm you are trying to improve on, or stem from the expected input size (e.g. you need to work with very large inputs so you need $O(n \log n)$ or better).

• Targets.

Divide-and-conquer algorithms often seek to improve on a polynomial-time iterative brute-force running time. If there aren't other targets, identify the brute force algorithm and its running time.

Approach.

CPSC 32

Identify applicable approach(es): divide and conquer or series of choices? For each, consider briefly what that approach would look like for this problem — does it even make sense?

• Paradigms and patterns.

Based on the targets and approach(es) identified, identify the applicable paradigm(s) and specific pattern(s) within each paradigm. For each, consider briefly what that paradigm and pattern would look like for this problem - does it even make sense?

2+-friend solutions, where there is more than one subproblem at each level divide-and-conquer Each friend gets a portion of the input: the problem is split into b subproblems of size n/b where b ≥ 2 (typically 2). easy split, where the input is divided in half in a straightforward way (such as "first half" and "second half"); the work in the main case is primarily in combining the results from the friends to produce the solution easy merge, where each friend produces some of the output (typically one friend produces the first

 easy inerge, where each mend produces some or the output (typically one mend produces the inst part of the output and the other friend produces the second part); the work in the main case is primarily in splitting the input

48

How to Design (Divide-and-Conquer) Algorithms

Establish the problem. Defining — and understanding — the precise task to be solved is an essential prerequisite for algorithm development.

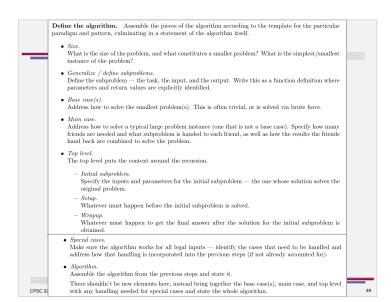
Specifications.

State complete specifications for the problem. What is the problem? What do you start with (input) and what is the end result (output)? What are the legal input instances and the required output for each?

• Examples.

If needed, give examples (specific inputs and the corresponding outputs) of typical and special cases to clarify the specifications.

CPSC 327: Data Structures and Algorithms • Spring 2025



How to Design (Divide-and-Conquer) Algorithms

Show termination and correctness. Show that the algorithm produces a correct solution.

- Termination. Show that the recursion and thus the algorithm always terminates.
 - Making progress.
 - Explain why what each of your friends get is a smaller instance of the problem. – The end is reached.
 - Explain why a base case is always reached.
- Correctness. Show that the algorithm is correct.
 - Establish the base case(s).
 - Explain why the solution is correct for each base case.
 - Show the main case.

Assume that the friends return the correct results for their subproblem, and explain why the correct answer is then produced from those results.

Final answer.

Explain why the top level — the setup plus a correct solution to the initial subproblem followed by the wrapup — means that the final result is a correct answer to the problem.

CPSC 327: Data Structures and Algorithms • Spring 2025

How to Design (Divide-and-Conquer) Algorithms

Determine efficiency. Evaluate the running time and space requirements of the algorithm.

• Implementation

Identify data structures and, as necessary, specific implementations of those data structures to efficiently support the algorithm. Also fill in any algorithmic details that are needed in order to establish the running time.

Time and space.
 Assess the running time and space requirements of the algorithm given the implementation identified.

Recursive algorithms tend to lead to recurrence relations in one of two forms:

split off b elements

T(n) = a T(n-b) + f(n) where f(n) = 0 or $\Theta(n^c \log^d n)$

divide into subproblems of size n/bT(n) = a T(n/b) + f(n) where $\Theta(n^c \log^d n)$

Room for improvement.

Are the targets met? Is it necessary to do better? If improvements in running time and/or space are needed, identify possible avenues for improvement.

CPSC 327: Data Structures and Algorithms • Spring 2025