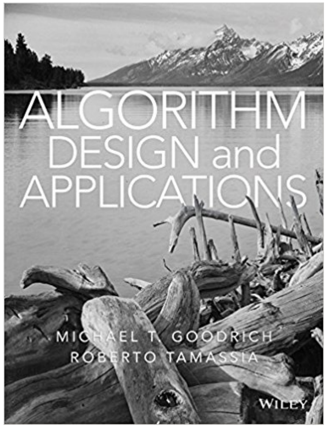
COMP26120: Divide and Conquer (2020/21)

Lucas Cordeiro
lucas.cordeiro@manchester.ac.uk

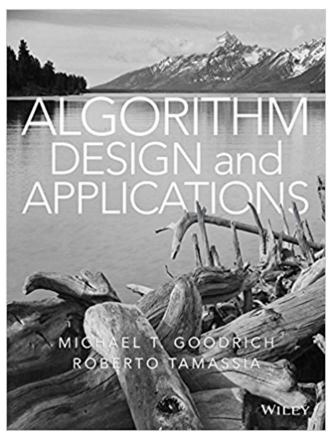
Divide-and-Conquer (Recurrence)

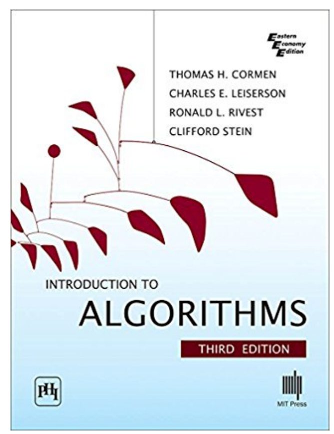
- References:
  - *Algorithm Design and Applications,* Goodrich, Michael T. and Roberto Tamassia (Chapter 8)



Divide-and-Conquer (Recurrence)

- References:
  - *Algorithm Design and Applications,* Goodrich, Michael T. and Roberto Tamassia (Chapter 8)
  - *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein (Chapters 2 and 4)





Intended Learning Outcomes

- **Understand** the **divide-and-conquer** paradigm and how **recurrence** can be obtained

Intended Learning Outcomes

- **Understand** the **divide-and-conquer** paradigm and how **recurrence** can be obtained
- Solve recurrences using **substitution** method

Intended Learning Outcomes

- **Understand** the **divide-and-conquer** paradigm and how **recurrence** can be obtained
- Solve recurrences using **substitution** method
- Describe **various examples** to analyse divide-and-conquer algorithms and how to solve their recurrences

Divide-and-Conquer

- The **divide-and-conquer** paradigm involves three steps at each level of the **recursion**:

Divide-and-Conquer

- The **divide-and-conquer** paradigm involves three steps at each level of the **recursion**:
  - **Divide** the problem into some subproblems that are smaller instances of the same problem *(D(n))*

Divide-and-Conquer

- The **divide-and-conquer** paradigm involves three steps at each level of the **recursion**:
  - **Divide** the problem into some subproblems that are smaller instances of the same problem *(D(n))*
  - **Conquer** the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just straightforwardly solve the subproblems *(aT(n/b))*

Divide-and-Conquer

- The **divide-and-conquer** paradigm involves three steps at each level of the **recursion**:
  - **Divide** the problem into some subproblems that are smaller instances of the same problem *(D(n))*
  - **Conquer** the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just straightforwardly solve the subproblems *(aT(n/b))*
  - **Combine** the solutions to the subproblems into the solution for the original problem *(C(n))*

Divide-and-Conquer

- The **divide-and-conquer** paradigm involves three steps at each level of the **recursion**:
  - **Divide** the problem into some subproblems that are smaller instances of the same problem *(D(n))*
  - **Conquer** the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just straightforwardly solve the subproblems *(aT(n/b))*
  - **Combine** the solutions to the subproblems into the solution for the original problem *(C(n))*

$$T(n) = \Theta(1) \quad \text{if } n \le c,$$
$$T(n) = D(n) + aT(n/b) + C(n) \text{ otherwise.}$$

Illustrative Example (Merge Sort)

- The merge sort algorithm closely follows the **divide-and-conquer** paradigm

Illustrative Example (Merge Sort)

- The merge sort algorithm closely follows the **divide-and-conquer** paradigm
  - **Divide:** Divide the *n*-element sequence to be sorted into two subsequences of $n/2$ elements each ($\Theta(1)$)

Illustrative Example (Merge Sort)

- The merge sort algorithm closely follows the **divide-and-conquer** paradigm
  - **Divide:** Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each ($\Theta(1)$)
  - **Conquer:** Sort the two subsequences recursively using merge sort ($2T(n/2)$)

Illustrative Example (Merge Sort)

- The merge sort algorithm closely follows the **divide-and-conquer** paradigm
  - **Divide:** Divide the *n*-element sequence to be sorted into two subsequences of $n/2$ elements each ($\Theta(1)$)
  - **Conquer:** Sort the two subsequences recursively using merge sort ($2T(n/2)$)
  - **Combine:** Merge the two sorted subsequences to produce the sorted answer ($\Theta(n)$)

Illustrative Example (Merge Sort)

- The merge sort algorithm closely follows the **divide-and-conquer** paradigm
  - **Divide:** Divide the *n*-element sequence to be sorted into two subsequences of $n/2$ elements each ($\Theta(1)$)
  - **Conquer:** Sort the two subsequences recursively using merge sort ($2T(n/2)$)
  - **Combine:** Merge the two sorted subsequences to produce the sorted answer ($\Theta(n)$)

$$T(n) = \Theta(1) \quad \text{if } n = 1,$$
$$T(n) = \Theta(1) + 2T(n/2) + \Theta(n) \quad \text{otherwise.}$$

How Does Merge Sort Work?

- **Idea:** suppose we have two piles of cards face up on a table; **each pile is sorted**, with the smallest cards on top

How Does Merge Sort Work?

- **Idea:** suppose we have two piles of cards face up on a table; **each pile is sorted**, with the smallest cards on top
- We wish to **merge the two piles** into a single sorted output pile, which is to be face down on the table

How Does Merge Sort Work?

- **Idea:** suppose we have two piles of cards face up on a table; **each pile is sorted**, with the smallest cards on top
- We wish to **merge the two piles** into a single sorted output pile, which is to be face down on the table
    1. choose the smaller of the two cards on top of the face-up piles

How Does Merge Sort Work?

- **Idea:** suppose we have two piles of cards face up on a table; **each pile is sorted**, with the smallest cards on top
- We wish to **merge the two piles** into a single sorted output pile, which is to be face down on the table
  1. choose the smaller of the two cards on top of the face-up piles
  2. remove it from its pile (which exposes a new top card)

How Does Merge Sort Work?

- **Idea:** suppose we have two piles of cards face up on a table; **each pile is sorted**, with the smallest cards on top
- We wish to **merge the two piles** into a single sorted output pile, which is to be face down on the table
  1. choose the smaller of the two cards on top of the face-up piles
  2. remove it from its pile (which exposes a new top card)
  3. place this card face down onto the output pile

How Does Merge Sort Work?

- **Idea:** suppose we have two piles of cards face up on a table; **each pile is sorted**, with the smallest cards on top
- We wish to **merge the two piles** into a single sorted output pile, which is to be face down on the table
  1. choose the smaller of the two cards on top of the face-up piles
  2. remove it from its pile (which exposes a new top card)
  3. place this card face down onto the output pile
  4. repeat this until one input pile is empty, at which time we take the remaining input pile and place it face down onto the output pile