NTNU

TDT4121 Introduction to Algorithms

Assignment 8
Practice Exam

Curriculum:
Lectures, Previous Assignments and their relevant chapters

Instructions:

This assignment is meant as practice for the final exam, the assignment is constructed the same way the exam will be, with the same type if questions.

There will not be programming on the final exam, and hence, there is no programming part for this assignment.

1. To solve the single source shortest paths problem on weighted directed graphs with integral weights, one can first transform every directed edge (u, v) with weight $w(u, v) = k > 1$ into a directed path $\langle u, x1, x3, ..., xk - 1, v\rangle$ with length k, and then use BFS to find the shortest paths. What are the advantages or disadvantages of this method? Discuss relationships with algorithms studied in the class. (Is this always possible to solve any instance of the problem using this method? Does it give a faster/slower algorithm? Why? Discuss various possible scenarios.)

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

   - $n^{\log 6}$
   - $6^{\log n}$
   - $(n \log n)^6$
   - $(6 \log 6)^n$
   - $(n \log 6)^6$
   - $(6 \log n)^6$

3. Given a set of n positive integers whose sum is $S \leq n^c$ for some small constant $c > 1$, describe an efficient algorithm to check if there exists a partition of the set into two subsets such the sum of all the integers in each subset is the same. Analyze the running time of your algorithm.

4. (a) Suppose someone finds a polynomial time algorithm for *factoring*. What would be the consequences for the relationship between *P*, *NP* and *co-NP*?

   (b) Independently (i.e., without having a polynomial time algorithms for factoring), what would be the consequences if someone showed a polynomial time reduction from *subsetsum* to *factoring*?

5. What is a priority queue? What operations does it support? If we use a priority queue to sort an input sequence of n elements, what is the runtime? (Express it in terms of the complexities of the priority queue operations.)

6. Given the string "`SHESELLSSEASHELLSONSEASHORE`", encode it using a variable-length prefix code such that the total length of the encoding is minimized (assuming that the alphabet contains only the characters that appear in the string).

7. Let A be an array consisting of n distinct integers. An inversion in A is a pair (i, j) such $i < j$ and $A[i] > A[j]$. Describe the algorithm (that is discussed in the class) to compute the number of inversions in A. Show its runtime analysis briefly.

8. There are n professors at NTNU, each having some joint research projects with m professors (from various universities) in Denmark - each professor can be part of multiple joint projects, and each joint project has exactly one Norwegian and one Danish professor associated with it. For a workshop discussing Norway-Denmark research cooperation, we would like to choose the minimum number of professors from either of these countries, such that all the joint projects are represented at the workshop (i.e., for each joint project, at least one of the professors associated with it is chosen).

   Is it possible to solve this problem in polynomial time? Justify your answer.

   [**Note:** You don't need to design an algorithm. Once you identify the right problem, you can lookup to check if there is an efficient algorithm to solve it.]

1. To solve the single source shortest paths problem on weighted directed graphs with integral weights, one can first transform every directed edge (u, v) with weight $w(u, v) = k > 1$ into a directed path $\langle u, x1, x3, ..., xk - 1, v\rangle$ with length k, and then use BFS to find the shortest paths. What are the advantages or disadvantages of this method? Discuss relationships with algorithms studied in the class. (Is this always possible to solve any instance of the problem using this method? Does it give a faster/slower algorithm? Why? Discuss various possible scenarios.)

1. To solve the single source shortest paths problem on weighted directed graphs with integral weights, one can first transform every directed edge (u, v) with weight $w(u, v) = k > 1$ into a directed path $\langle u, x1, x3, ..., xk - 1, v\rangle$ with length k, and then use BFS to find the shortest paths. What are the advantages or disadvantages of this method? Discuss relationships with algorithms studied in the class. (Is this always possible to solve any instance of the problem using this method? Does it give a faster/slower algorithm? Why? Discuss various possible scenarios.)

Assignment 4

Greedy Algorithms

4.4 Shortest Paths in a Graph

Some of the basic algorithms for graphs are based on greedy design principles. Here we apply a greedy algorithm to the problem of finding shortest paths, and in the next section we look at the construction of minimum-cost spanning trees.

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

   - $n^{\log 6}$
   - $6^{\log n}$
   - $(n \log n)^6$
   - $(6 \log 6)^n$
   - $(n \log 6)^6$
   - $(6 \log n)^6$

Upper bound

$O(n) < c \cdot n$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

   - $n^{\log 6}$
   - $6^{\log n}$
   - $(n \log n)^6$
   - $(6 \log 6)^n$
   - $(n \log 6)^6$
   - $(6 \log n)^6$

Lower bound

$\Omega(n) > c \cdot n$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

   - $n^{\log 6}$
   - $6^{\log n}$
   - $(n \log n)^6$
   - $(6 \log 6)^n$
   - $(n \log 6)^6$
   - $(6 \log n)^6$

Tight bound

$c_1 n < \Theta(n) < c_2 n$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

   - $n^{\log 6}$
   - $6^{\log n}$
   - $(n \log n)^6$
   - $(6 \log 6)^n$
   - $(n \log 6)^6$
   - $(6 \log n)^6$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Asymptotic notation and arithmetic:

$O(n) + \Omega(n^2)$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Asymptotic notation and arithmetic:

$O(n) + \Omega(n^2)$

Total upper bound: ∞

Total lower bound: $n^2 + n$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Asymptotic notation and arithmetic:

$O(n) + \Omega(n^2)$

Total upper bound: ∞

Total lower bound: $n^2 + n$

$\Big\}\ \Omega(n^2)$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Asymptotic notation and arithmetic:

$O(n) \cdot \Theta(n^2)$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Asymptotic notation and arithmetic:

$O(n) \cdot \Theta(n^2)$

Total upper bound: n^3

Total lower bound: 0

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Asymptotic notation and arithmetic:

$O(n) \cdot \Theta(n^2)$

Total upper bound: n^3

Total lower bound: 0

$\Big\}\ O(n^3)$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Asymptotic notation and arithmetic:

$$\frac{\Theta(n^3)}{\Omega(n)}$$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Asymptotic notation and arithmetic:

$$\frac{\Theta(n^3)}{\Omega(n)}$$

Total upper bound: n^2

Total lower bound: 0

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

   (b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Asymptotic notation and arithmetic:

$$\frac{\Theta(n^3)}{\Omega(n)}$$

Total upper bound: n^2

Total lower bound: 0

$\Big\}\ O(n^2)$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

(b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

(b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Logarithmic rules:

- $a\log(b) = \log(b^a)$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

(b) Sort the following terms in the increasing order of their asymptotic growth:

- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Logarithmic rules:

- $a \log(b) = \log(b^a)$
- $a^{\log(b)} = b^{\log(a)}$ for $a > 1$, $b > 1$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

(b) Sort the following terms in the increasing order of their asymptotic growth:

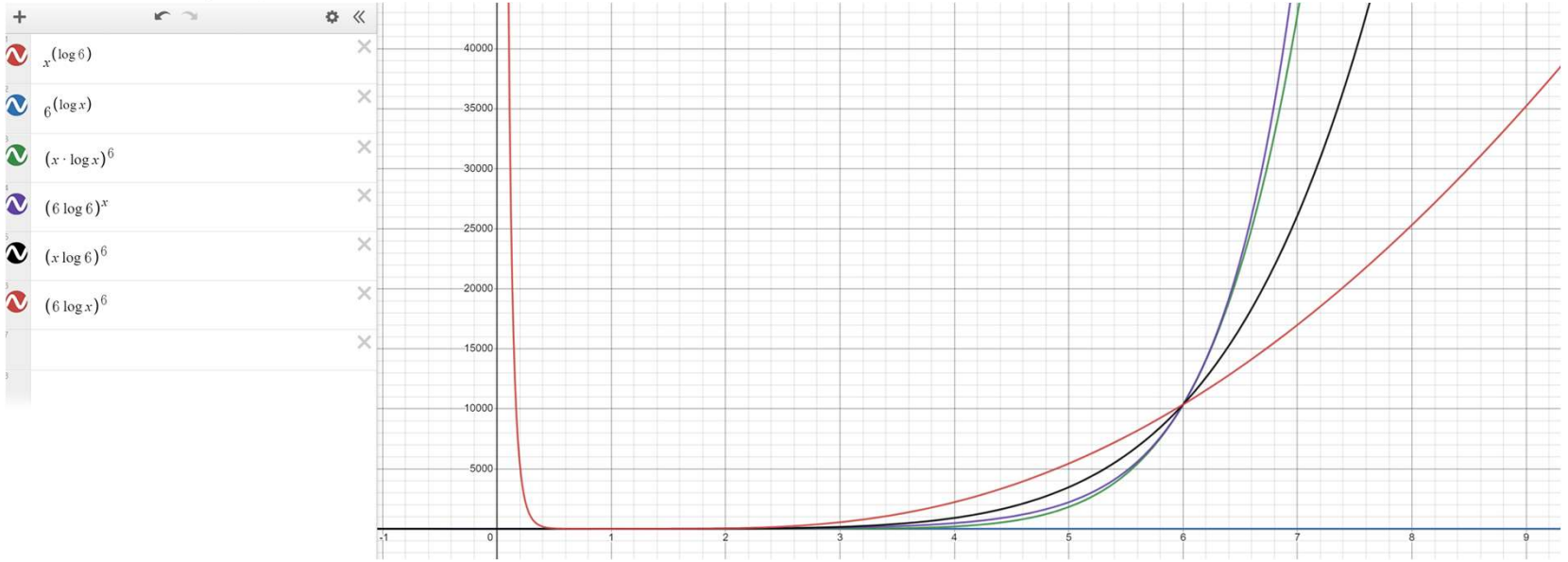
- $n^{\log 6}$
- $6^{\log n}$
- $(n \log n)^6$
- $(6 \log 6)^n$
- $(n \log 6)^6$
- $(6 \log n)^6$

Logarithmic rules:

- $a \log(b) = \log(b^a)$
- $a^{\log(b)} = b^{\log(a)}$ for $a>1$, $b>1$
- $a < b \iff \log a < \log b$

2. (a) Simplify the expression: $O(n) + \Omega(n) + \Theta(n)$

(b) Sort the following terms in the increasing order of their asymptotic growth:



(Verify by plots, but not on exam)

3. Given a set of n positive integers whose sum is $S \leq n^c$ for some small constant $c > 1$, describe an efficient algorithm to check if there exists a partition of the set into two subsets such the sum of all the integers in each subset is the same. Analyze the running time of your algorithm.

3. Given a set of n positive integers whose sum is $S \leq n^c$ for some small constant $c > 1$, describe an efficient algorithm to check if there exists a partition of the set into two subsets such the sum of all the integers in each subset is the same. Analyze the running time of your algorithm.

Example: $\{8, 2, 4, 1, 5, 6, 2\}$ $(n = 7)$

3. Given a set of n positive integers whose sum is $S \leq n^c$ for some small constant $c > 1$, describe an efficient algorithm to check if there exists a partition of the set into two subsets such the sum of all the integers in each subset is the same. Analyze the running time of your algorithm.

Example: $\{8, 2, 4, 1, 5, 6, 2\}$

$\{8, 1, 5\}$ $\{2, 4, 6, 2\}$

Sum: 14 Sum: 14

3. Given a set of n positive integers whose sum is $S \leq n^c$ for some small constant $c > 1$, describe an efficient* algorithm to check if there exists a partition of the set into two subsets such the sum of all the integers in each subset is the same. Analyze the running time of your algorithm.

* Polynomial time ($O(n^k)$)

4. (a) Suppose someone finds a polynomial time algorithm for *factoring*. What would be the consequences for the relationship between P, NP and *co-NP*?

   (b) Independently (i.e., without having a polynomial time algorithms for factoring), what would be the consequences if someone showed a polynomial time reduction from *subsetsum* to *factoring*?

4. (a) Suppose someone finds a polynomial time algorithm for *factoring*. What would be the consequences for the relationship between P, NP and *co-NP*?

   (b) Independently (i.e., without having a polynomial time algorithms for factoring), what would be the consequences if someone showed a polynomial time reduction from *subsetsum* to *factoring*?

Polynomial-Time Reduction (Assignment 7)

- A method for solving one problem using another
- Proves that the first problem is no more difficult than the second one
- Whenever an efficient algorithm exists for the second problem, one exists for the first problem as well
- $X \leq_p Y$

5. What is a priority queue? What operations does it support? If we use a priority queue to sort an input sequence of n elements, what is the runtime? (Express it in terms of the complexities of the priority queue operations.)

5. What is a priority queue? What operations does it support? If we use a priority queue to sort an input sequence of n elements, what is the runtime? (Express it in terms of the complexities of the priority queue operations.)

Assignment 1

Algorithm Analysis

2.5 A More Complex Data Structure: Priority Queues

Our primary goal in this book was expressed at the outset of the chapter: we seek algorithms that improve qualitatively on brute-force search, and in general we use polynomial-time solvability as the concrete formulation of this. Typically, achieving a polynomial-time solution to a nontrivial problem

6. Given the string "SHESELLSSEASHELLSONSEASHORE", encode it using a variable-length prefix code such that the total length of the encoding is minimized (assuming that the alphabet contains only the characters that appear in the string).

6. Given the string "SHESELLSSEASHELLSONSEASHORE", encode it using a variable-length prefix code such that the total length of the encoding is minimized (assuming that the alphabet contains only the characters that appear in the string).

Assignment 4
Greedy Algorithms

4.8 Huffman Codes and Data Compression

In the Shortest-Path and Minimum Spanning Tree Problems, we've seen how greedy algorithms can be used to commit to certain parts of a solution (edges in a graph, in these cases), based entirely on relatively short-sighted consid-

7. Let A be an array consisting of n distinct integers. An inversion in A is a pair (i, j) such $i < j$ and $A[i] > A[j]$. Describe the algorithm (that is discussed in the class) to compute the number of inversions in A. Show its runtime analysis briefly.

7. Let A be an array consisting of n distinct integers. An inversion in A is a pair (i, j) such $i < j$ and $A[i] > A[j]$. Describe the algorithm (that is discussed in the class) to compute the number of inversions in A. Show its runtime analysis briefly.

Assignment 3
Divide & Conquer

5.3 Counting Inversions

We've spent some time discussing approaches to solving a number of common recurrences. The remainder of the chapter will illustrate the application of divide-and-conquer to problems from a number of different domains; we will use what we've seen in the previous sections to bound the running times

8. There are n professors at NTNU, each having some joint research projects with m professors (from various universities) in Denmark - each professor can be part of multiple joint projects, and each joint project has exactly one Norwegian and one Danish professor associated with it. For a workshop discussing Norway-Denmark research cooperation, we would like to choose the minimum number of professors from either of these countries, such that all the joint projects are represented at the workshop (i.e., for each joint project, at least one of the professors associated with it is chosen).

   Is it possible to solve this problem in polynomial time? Justify your answer.

   [**Note:** You don't need to design an algorithm. Once you identify the right problem, you can lookup to check if there is an efficient algorithm to solve it.]

8. There are n professors at NTNU, each having some joint research projects with m professors (from various universities) in Denmark - each professor can be part of multiple joint projects, and each joint project has exactly one Norwegian and one Danish professor associated with it. For a workshop discussing Norway-Denmark research cooperation, we would like to choose the minimum number of professors from either of these countries, such that all the joint projects are represented at the workshop (i.e., for each joint project, at least one of the professors associated with it is chosen).

{ Is it possible to solve this problem in polynomial time? Justify your answer.

[**Note:** You don't need to design an algorithm. Once you identify the right problem, you can lookup to check if there is an efficient algorithm to solve it.]

Is this problem in P, or is it NP-complete?

8. There are n professors at NTNU, each having some joint research projects with m professors (from various universities) in Denmark - each professor can be part of multiple joint projects, and each joint project has exactly one Norwegian and one Danish professor associated with it. For a workshop discussing Norway-Denmark research cooperation, we would like to choose the minimum number of professors from either of these countries, such that all the joint projects are represented at the workshop (i.e., for each joint project, at least one of the professors associated with it is chosen).

   Is it possible to solve this problem in polynomial time? Justify your answer.

   [**Note:** You don't need to design an algorithm. Once you identify the right problem, you can lookup to check if there is an efficient algorithm to solve it.]

Prove X in P: Show $X \leq_p Y$ for some known Y in P.

8. There are n professors at NTNU, each having some joint research projects with m professors (from various universities) in Denmark - each professor can be part of multiple joint projects, and each joint project has exactly one Norwegian and one Danish professor associated with it. For a workshop discussing Norway-Denmark research cooperation, we would like to choose the minimum number of professors from either of these countries, such that all the joint projects are represented at the workshop (i.e., for each joint project, at least one of the professors associated with it is chosen).

   Is it possible to solve this problem in polynomial time? Justify your answer.

   [**Note:** You don't need to design an algorithm. Once you identify the right problem, you can lookup to check if there is an efficient algorithm to solve it.]

Prove X in P: show $X \leq_p Y$ for some known Y in P.

Or prove X NP-complete: show $Z \leq_p X$ for some known NP-complete Z.

Chapter 1 - Introduction: Some Representative Problems

> *We start with an introduction of several representative problems, which introduce you to the world of algorithms.*

Goals 🏁

The student should be able to:

- Understand the *Stable Matching Problem*
- Understand how the *Gale-Shapley algorithm* solves the stable matching problem
- Understand some representative problems

Chapter 2 - Basics of Algorithm Analysis

> *Analyzing algorithms involves thinking about how their resource requirements—the amount of time and space they use—will scale with increasing input size.*

Goals 🏁

The student should be able to:

- Understand the different definitions of an efficient algorithm ✓
- Understand asymptotic notation and the asymptotic bounds O, Ω, Θ as well as their properties ✓
- Understand how the stable matching problem is implemented using arrays and lists
- Have knowledge of common running-time bounds and some of the typical approaches that lead to them ✓
- Understand what a priority queue is and how heaps are used to implement them ✓

Chapter 3 - Graphs

> *One of the most fundamental and expressive data structures in discrete mathematics*

Goals 🏁

The student should be able to:

- Understand the basic definition of a *graph* ✓
  - The difference between a graph and a *directed graph* ✓
- Know of examples of applications of graphs
- Understand what a *tree* is and their properties ✓
- Understand *BFS* - Breadth-First Search ✓
- Understand *DFS* - Depth-First Search
- Understand how graphs are implemented
- Understand *Directed Acyclic Graphs* and *Topological Ordering*

Chapter 4 - Divide and Conquer

> *Divide and conquer refers to a class of algorithmic techniques in which one breaks the input into several parts, solves the problem in each part recursively, and then combines the solutions to these subproblems into an overall solution. In many cases, it can be a simple and powerful method.*

Goals 🏁

The student should be able to:

- Understand the divide and conquer algorithm design method ✓
- Understand mergesort
  - Understand how its ideas are used to solve the Counting Inversions problem ✓
- Understand the Finding the Closest Pair of Points problem
- Understand the Integer Multiplication problem
- Solve a recurrence by "unrolling" it
- Solve a recurrence by using *substitution*

Chapter 5 - Greedy Algorithms

> *An algorithm is greedy if it builds up a solution in small steps, choosing a decision at each step myopically to optimize some underlying criterion. One can often design many different greedy algorithms for the same problem, each one locally, incrementally optimizing some different measure on its way to a solution*

Goals 🏁

The student should be able to:

- Understand the different *Interval Scheduling* problems
- Understand the *Optimal Caching Problem*
- Understand the *Shortest Path Problem* and how *Dijkstra's algorithm* solves it ✓
- Know what a *Minimum Spanning Tree* is and understand the *Minimum Spanning Tree Problem*
- Understand *Prim's* and *Kruskal's algorithm* ✓
- Understand *Huffman* and *Huffman Codes* ✓

Chapter 6 - Dynamic Programming

> *The basic idea of Dynamic Programming is drawn from the intuition behind divide and conquer and is essentially the opposite of the greedy strategy: one implicitly explores the space of all possible solutions, by carefully decomposing things into a series of subproblems, and then building up correct solutions to larger and larger subproblems.*

Goals 🏁

The student should be able to:

- Understand the Dynamic Programming design method
  - Understand when and how dynamic programming could be used
- Understand how dynamic programming tables are used
- Understand *The Knapsack Problem*
- Understand how dynamic programming can be used to solve the *Longest Common Subsequence* problem
- Understand how dynamic programming can be used to find *Sequence Alignment*
- Understand how dynamic programming can be used to find the *Shortest path in a graph* ✓
- Find the asymptotic running time for Dynamic Programming Algorithms.

Chapter 7 - Network Flow

> *In graph theory, a flow network is a directed graph where each edge has a capacity and each edge receives a flow. The amount of flow on an edge cannot exceed the capacity of the edge. While the initial motivation for network flow problems comes from the issue of traffic in a network, we will see that they have applications in a surprisingly diverse set of areas and lead to efficient algorithms not just for Bipartite Matching, but for a host of other problems as well.*

Goals 🏁

The student should be able to:

- Understand the *Maximum-Flow Problem*
- Understand the *Ford-Fulkerson Algorithm*
- Understand and solve the problem of *Minimum Cut* in a graph.
- Use *Network Flow* to solve *The Bipartite Matching Problem*
- Identify problems that can be solved using network flow.

Chapter 8 - NP and Computational Intractability

> *So far we've developed efficient algorithms for a wide range of problems and have even made some progress on informally categorizing the problems that admit efficient solutions. But although we've often paused to take note of other problems that we don't see how to solve, we haven't yet made any attempt to actually quantify or characterize the range of problems that **can't be solved efficiently**.*

Goals 🏁

The student should be able to:

- Understand the difference between **P**, **NP**, **NP-Complete** and **NP-Hard** problems. ✓
- Prove whether a problem is **NP** or **NP-Complete**. ✓
- Understand the basis of *Polynomial-Time Reduction* ✓
- Understand the *Satisfiability Problems*
- Understand the *Traveling Salesman Problem*
- Understand the *Hamiltonian Cycle Problem*

Study guide suggestion

- Understand the fundamentals of each algorithm design paradigm
  - Know the curriculum algorithms of each paradigm
    - What problem do they solve?
    - How do they solve the problem?
      - Can you run the algorithm by hand?
    - What is their run-time (and why)?