Algorithm Strategies

CSCI 1030U - Intro to Computer Science
@IntroCS

Randy J. Fortier
@randy_fortier

Ontario Tech University

Outline

- Algorithm strategies
  - Divide and conquer
    - Binary search
  - Greedy Algorithms
    - Fractional knapsack problem
  - Dynamic Programming
    - Fibonacci

Algorithm Strategies

- There are three common strategies used by many algorithms:
  - Divide and conquer
  - Greedy
  - Dynamic programming

Divide and Conquer

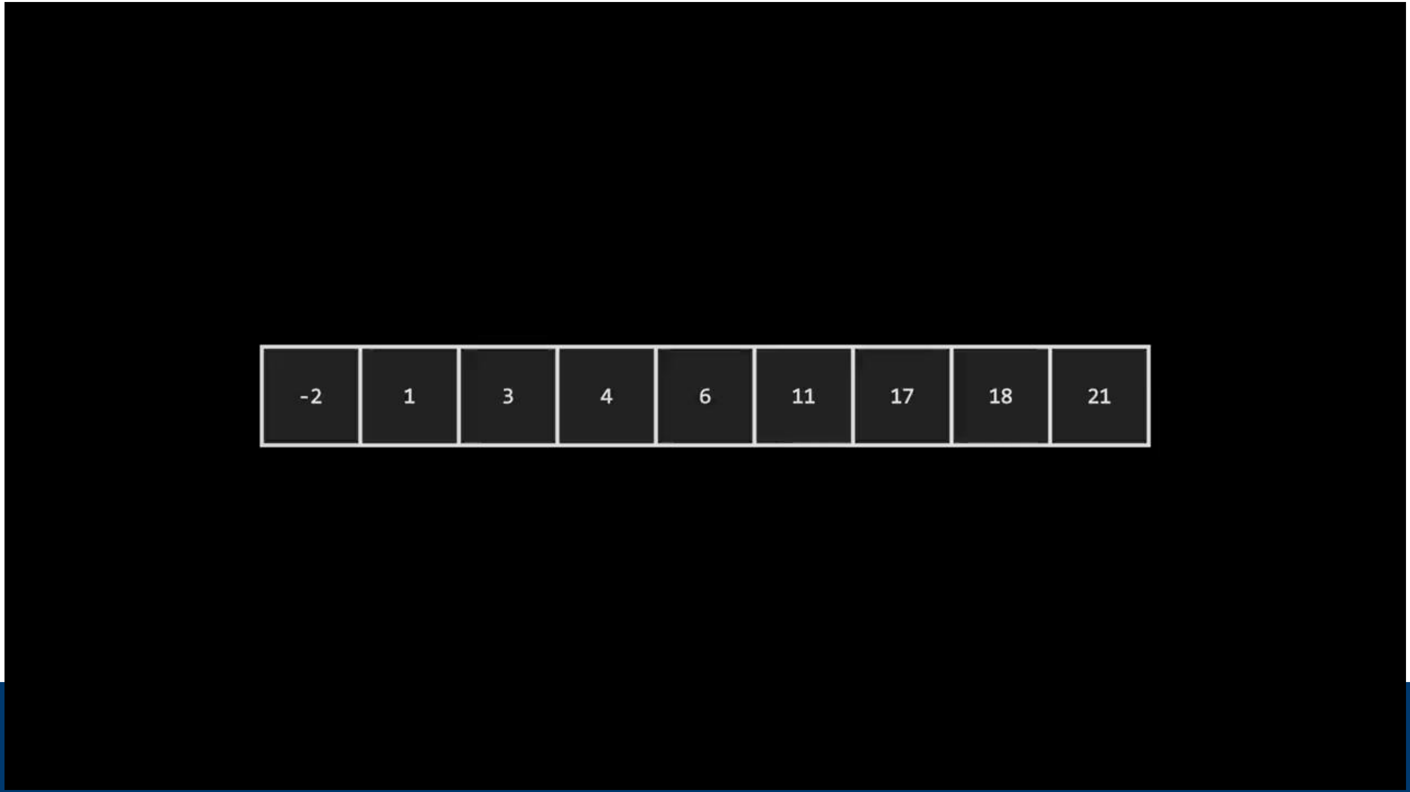
Divide and Conquer

- Divide and conquer uses the following strategy:
  - Divide the problem into one or more *smaller* subproblems
  - Recursively solve (conquer) the subproblem(s)
  - Combine the solutions to those subproblem(s)
- Clearly, a base case is needed
  - Generally, there is some size of subproblem that is trivial to solve (without recursion)

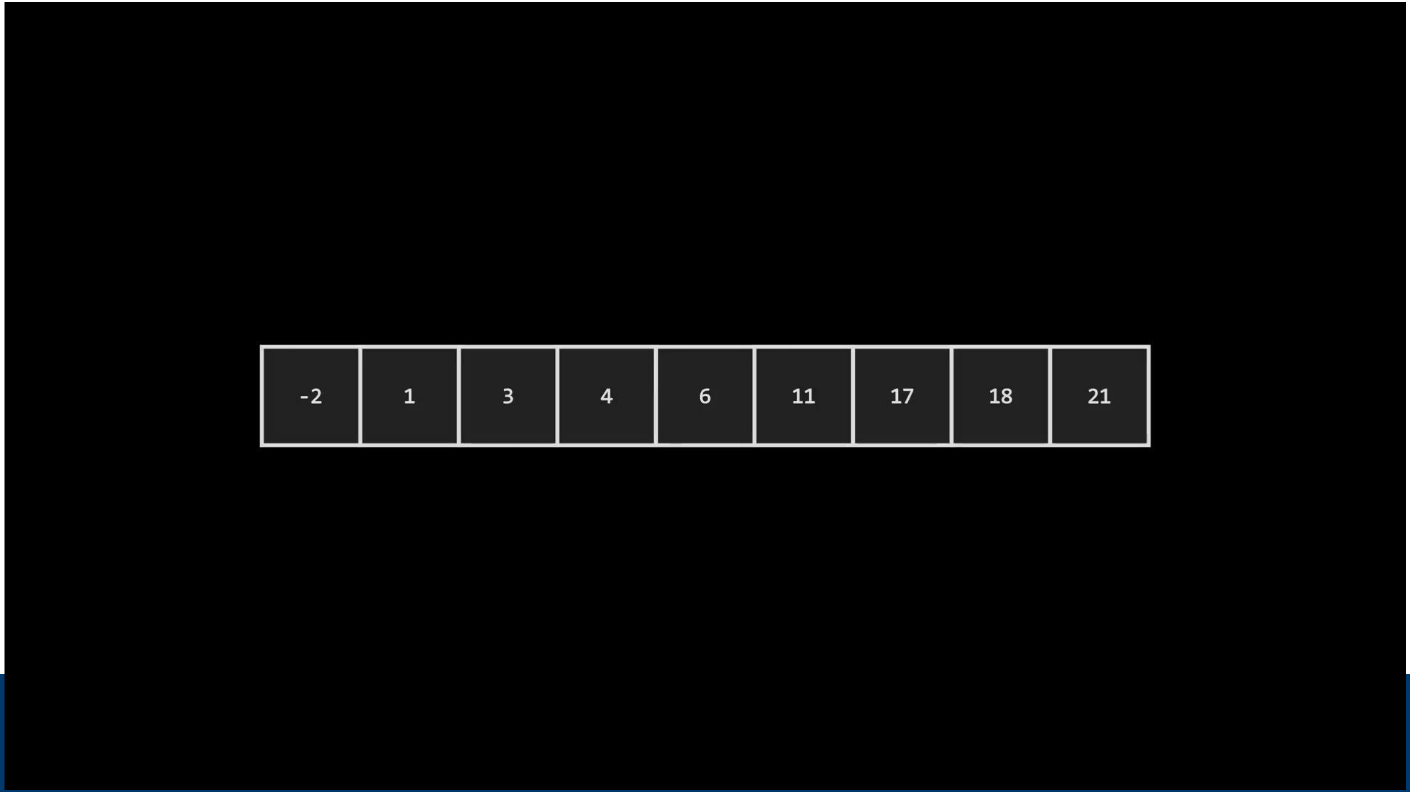
Binary Search

- Binary search is an example of a divide and conquer algorithm:
  - Start with a *sorted list*
  - If the list is empty, you are done (not found)
  - Divide the list into three parts:
    - First half of the list (A)
    - Middle element (M)
    - Second half of the list (B)
  - If `searchFor = M`, you are done (found)
  - If `searchFor < M`, recursively search A
  - If `searchFor > M`, recursively search B

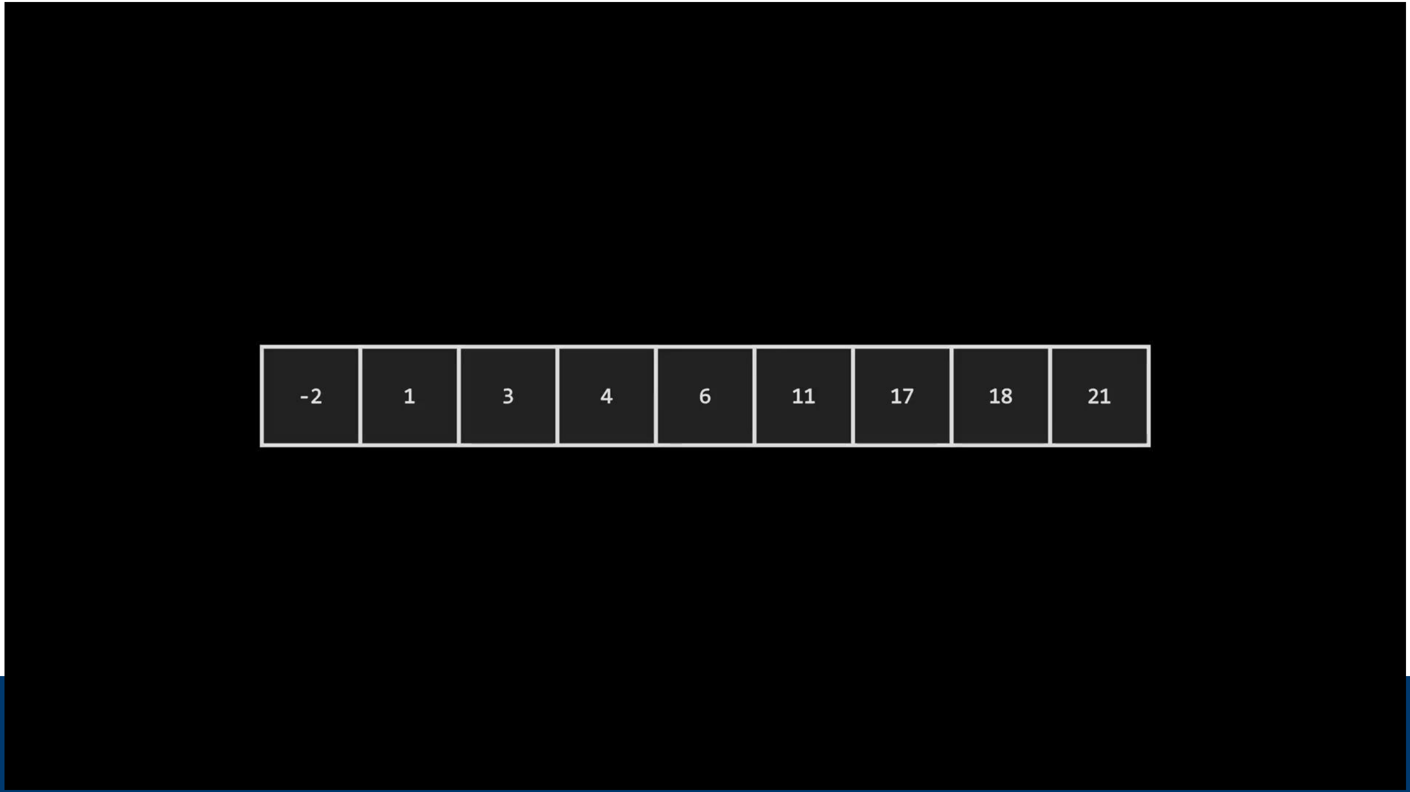
Binary Search - Video Example



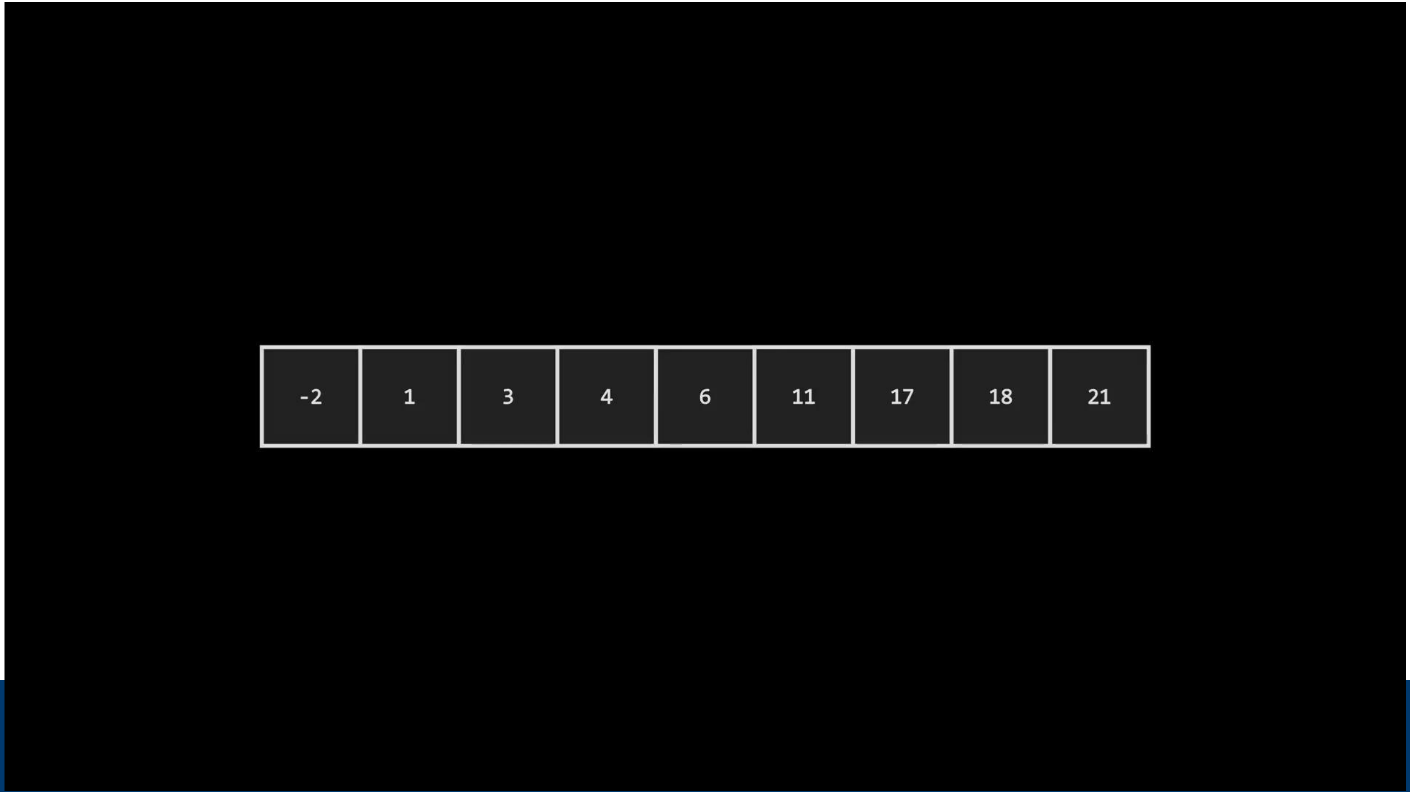
Binary Search - Video Example



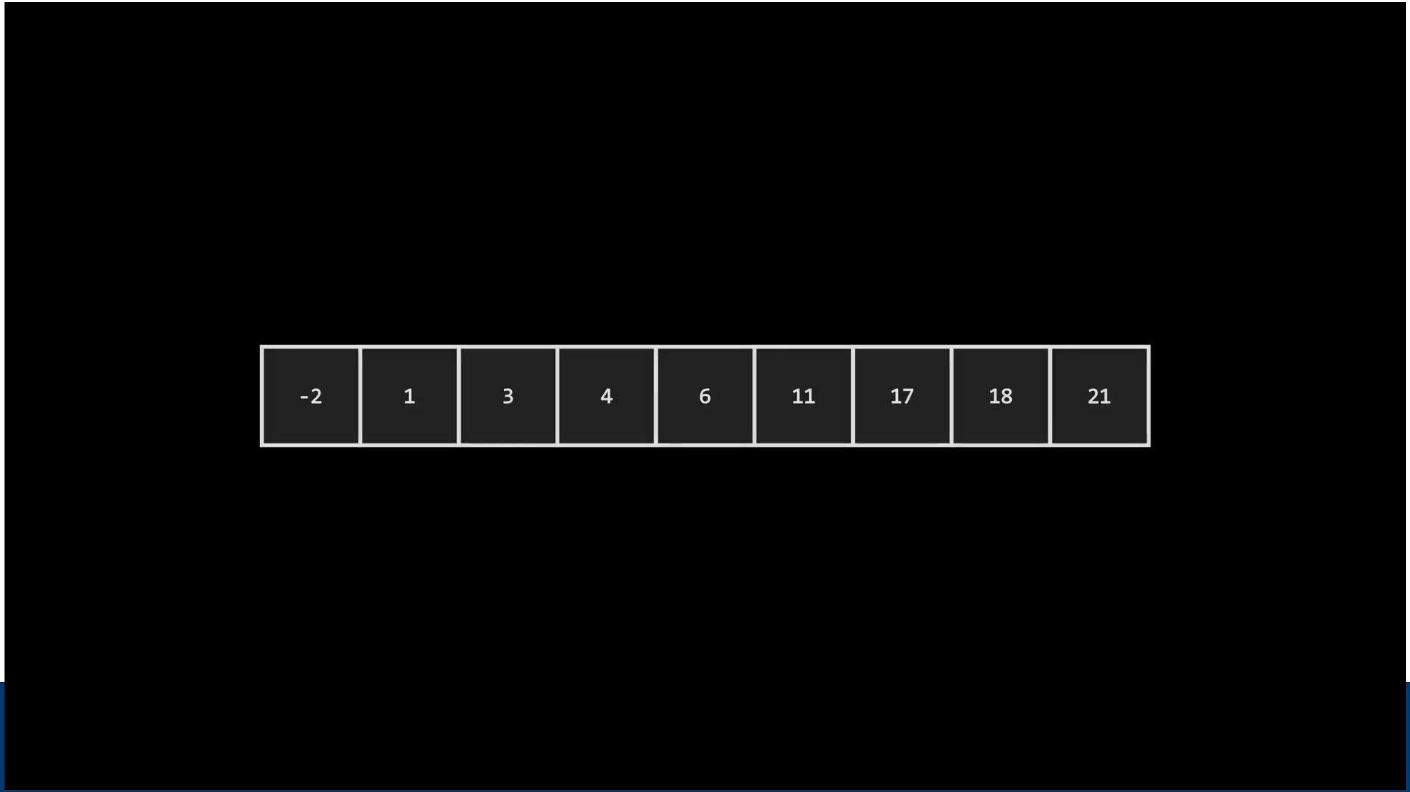
Binary Search - Video Example



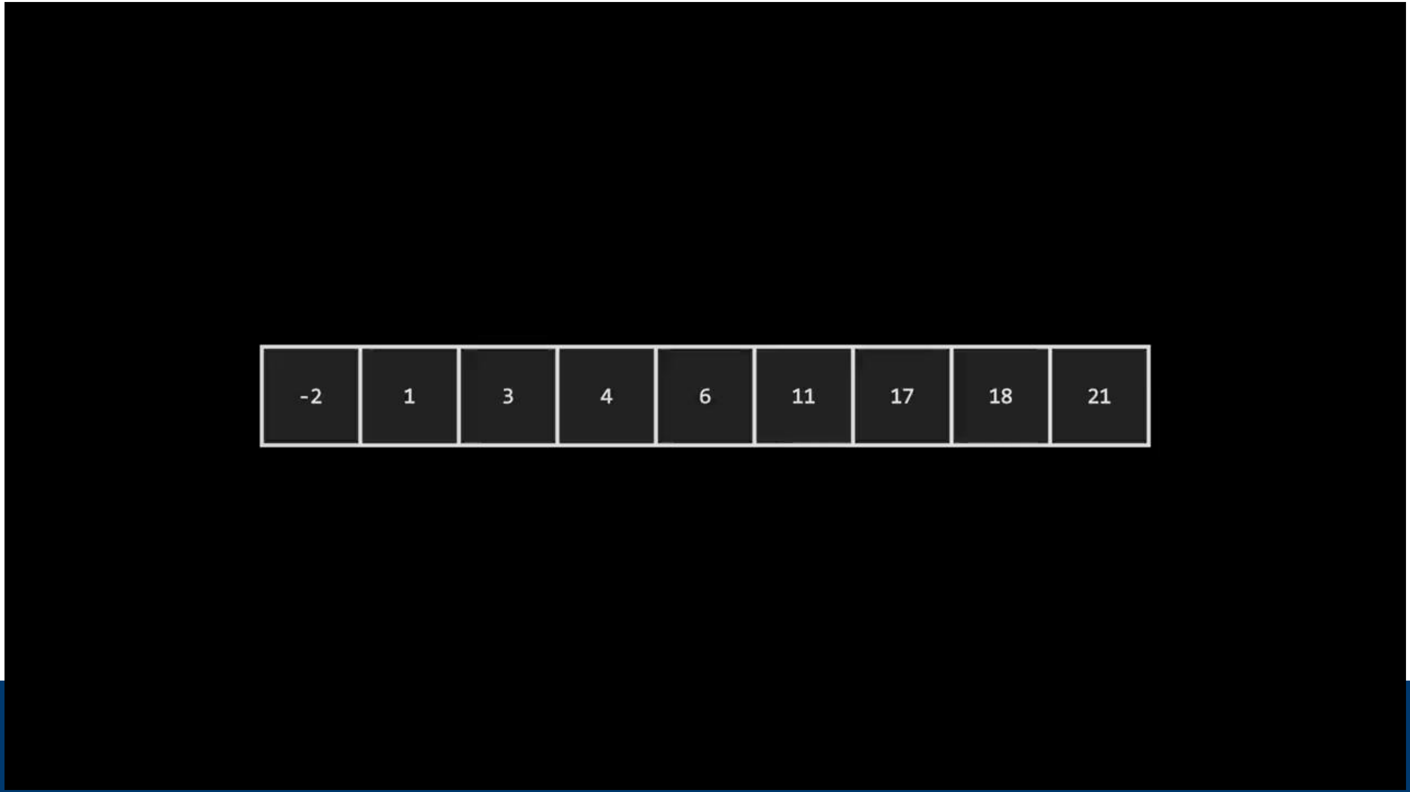
Binary Search - Video Example



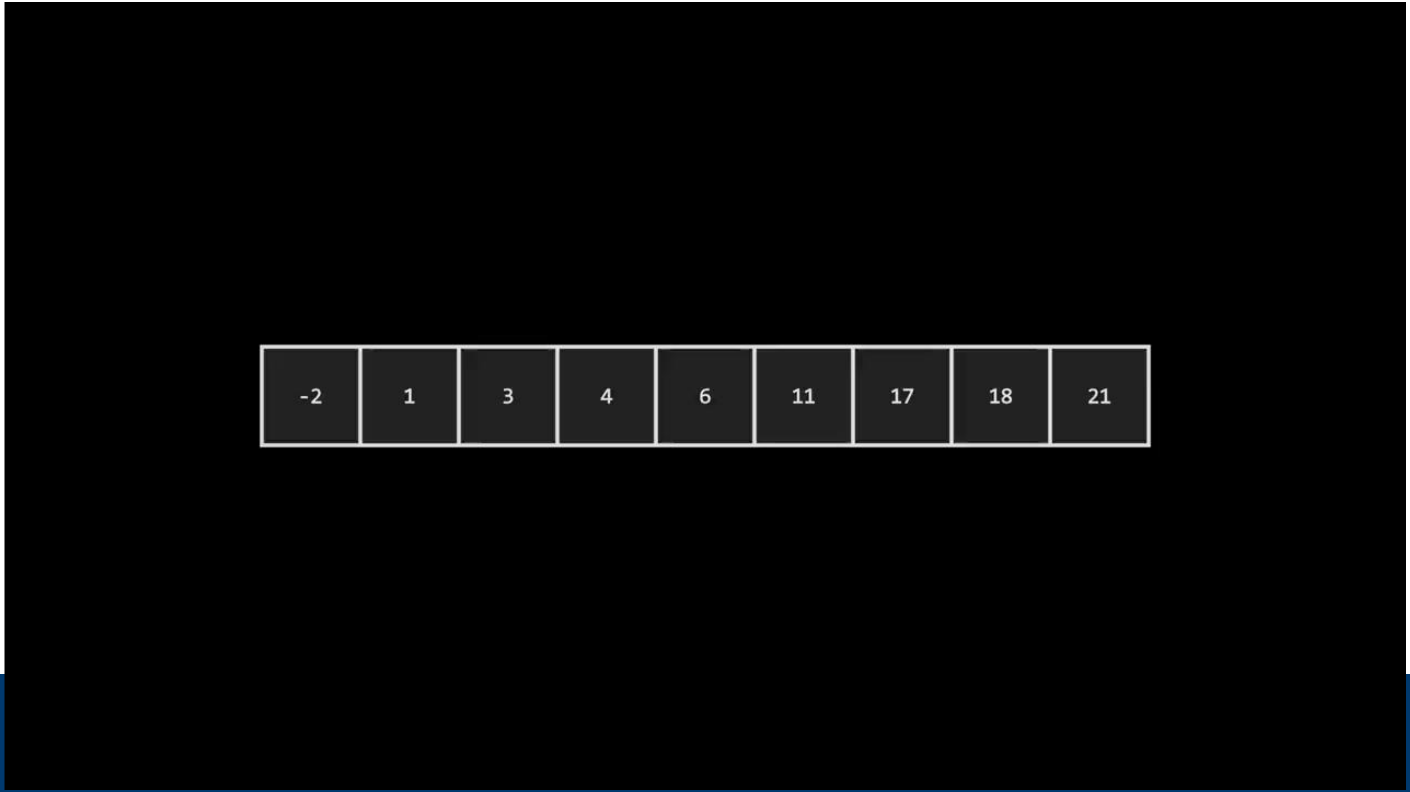
Binary Search - Video Example



Binary Search - Video Example



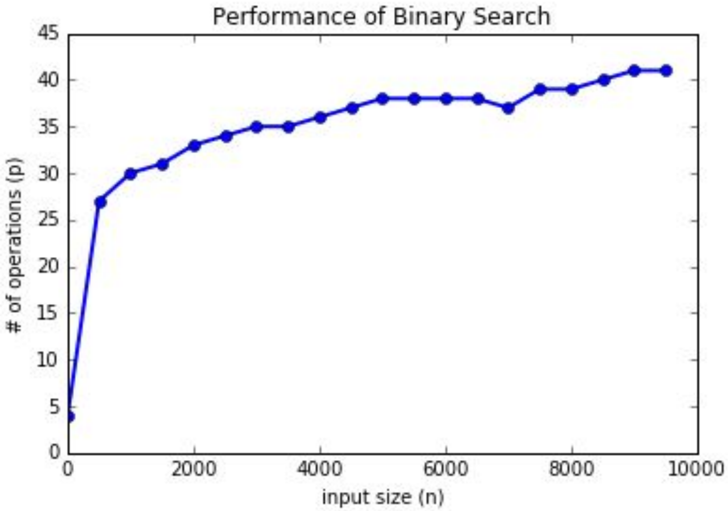
Binary Search - Video Example



Binary Search - Pseudo-code

```
BINARY-SEARCH(X, A, start, end)
  if start > end then
      return False
  middle = (start + end) / 2
  if X = A[middle] then
      return True
  else if X < A[middle] then
      return BINARY-SEARCH(X, A, start, middle - 1)
   else
       return BINARY-SEARCH(X, A, middle + 1, end)
```

Binary Search - Performance



Other Divide and Conquer Algorithms

- QuickSort
  - Take the middle element 'pivot'
  - Put all the elements < pivot into one sublist (A)
  - Put all the elements ≥ pivot into another (B)
  - Recursively sort A and B
  - Put them back in order: A, pivot, B

Coding Exercise 09b.1

- Write a search algorithm in Python that uses the divide and conquer strategy, but doesn't require that the list be sorted

Greedy Algorithms

Greedy Algorithms

- The heart of greedy algorithms is to always make the greedy choice
  - e.g. When trying to find an optimal path, choose the direction that moves you closest to your destination
  - e.g. When translating English to Japanese, choose the longest sequence of words to look up

Fractional Knapsack Problem

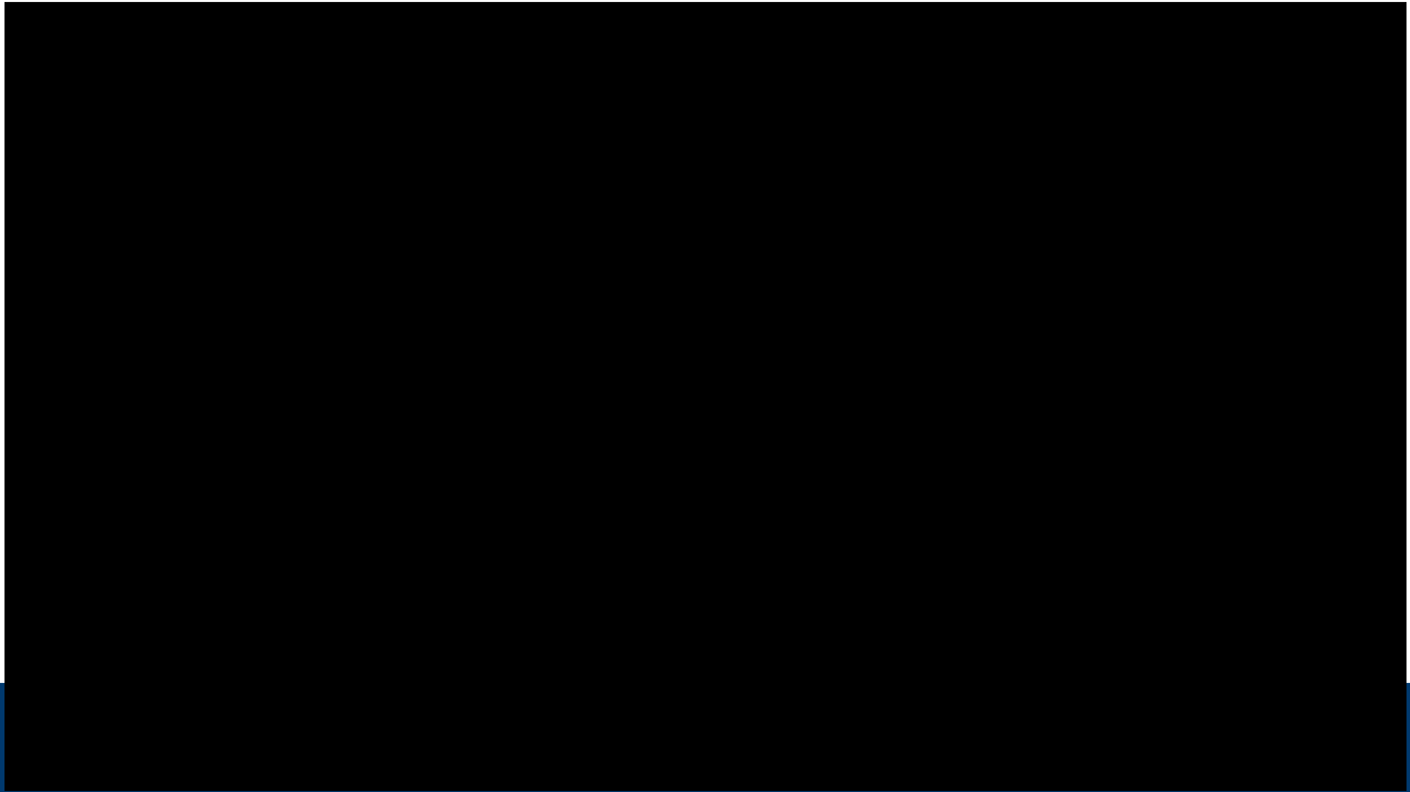
- A classic problem in Computer Science
  - You are a treasure hunter
  - You have a knapsack with capacity C
  - You have found a supply of various valuables
    - e.g. gold, diamonds, rubies, silver
  - You can take any amount of any valuable item
  - Each valuable item has a different *weight* and *value*
  - You can't take it all, how should you choose as to maximize your profit?
- Let's solve this problem with the greedy strategy

Fractional Knapsack Problem

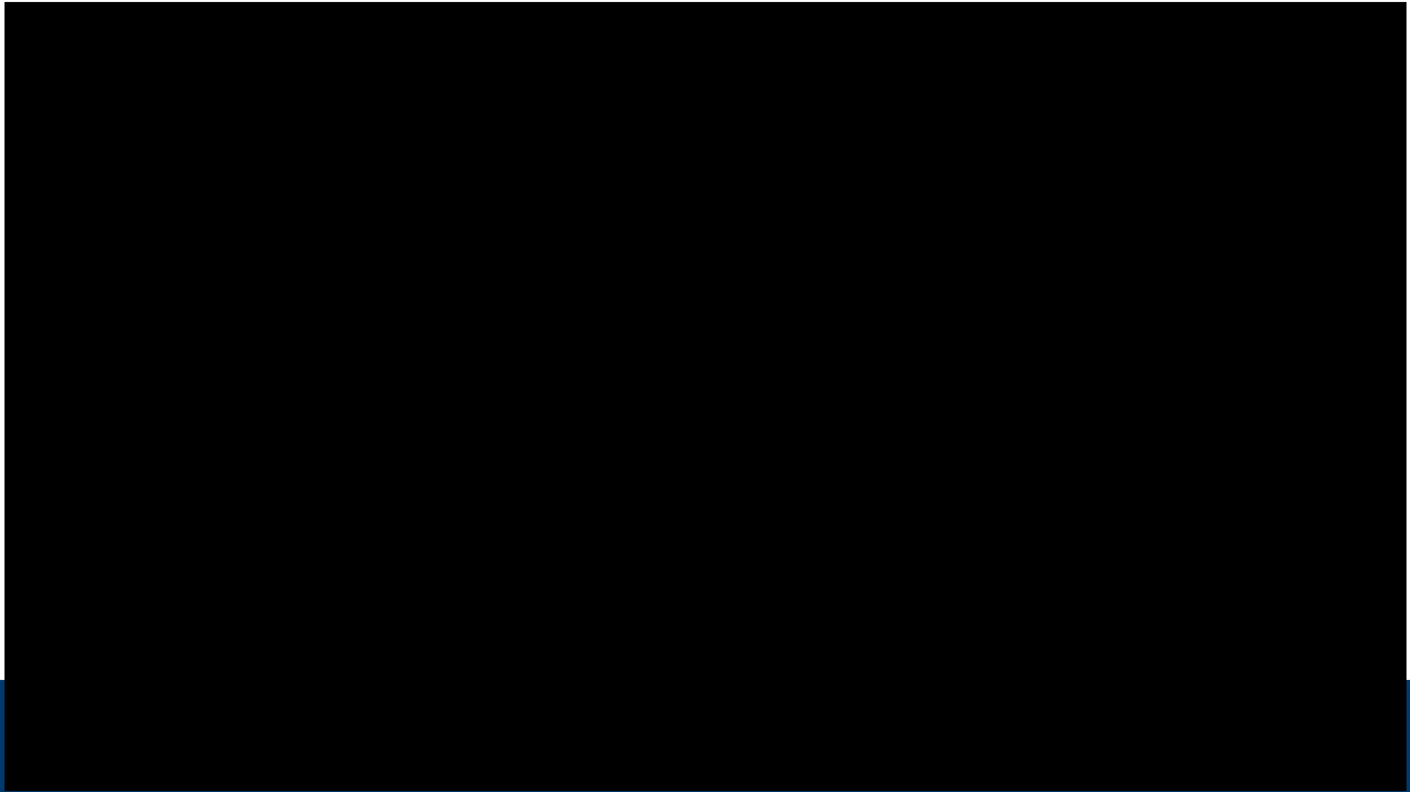
1. First, calculate the value/weight ratio for each valuable
2. Start by choosing the valuable with the *highest value/weight ratio*
   - If the weight is ≥ your knapsack's current capacity, then take the remaining capacity in weight of that valuable
   - If the weight is < than your knapsack's capacity, then take all the available weight of that valuable
   - Repeat step 2 until the knapsack has been filled

- It can be shown that this solution is optimal
  - It will always earn you the most money for your treasure!

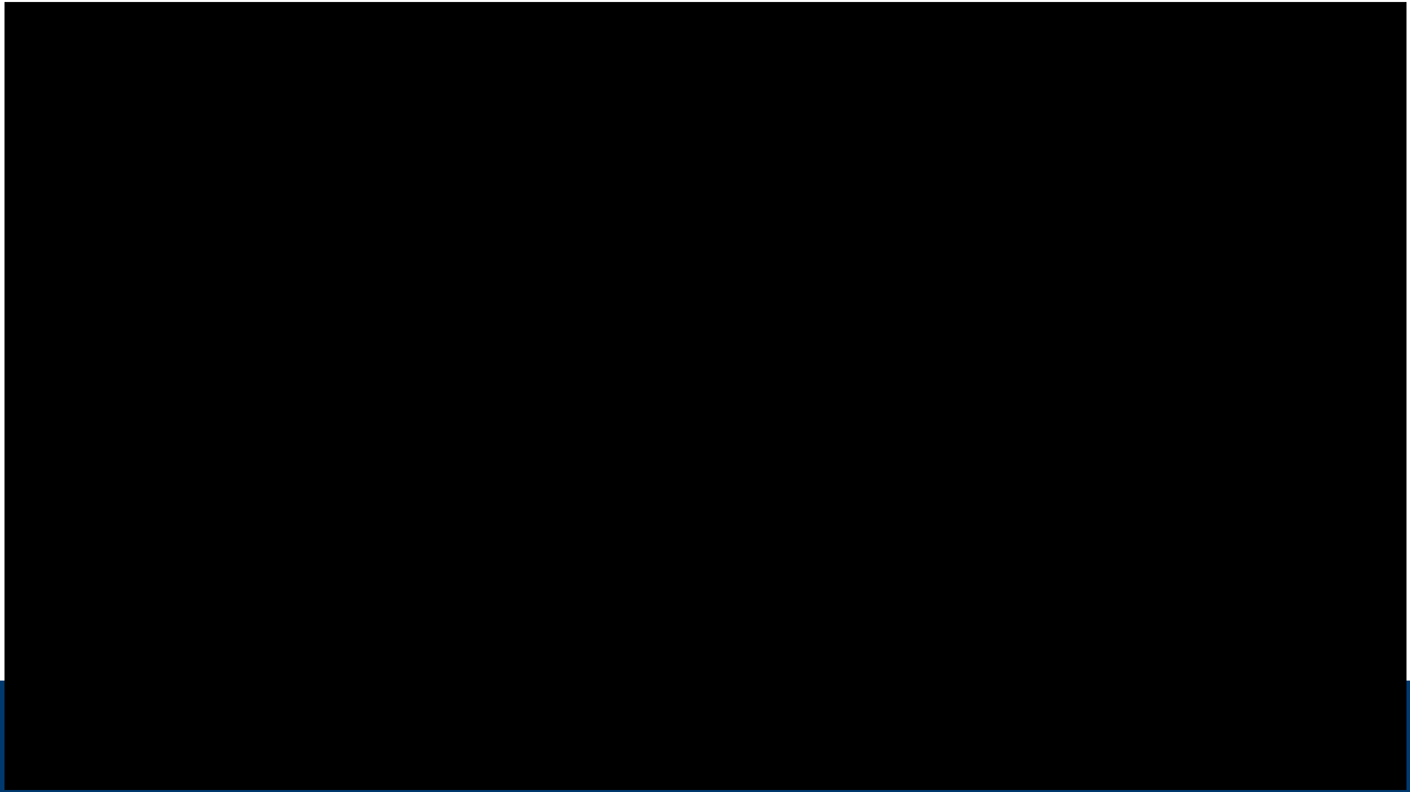
Fractional Knapsack - Video Example



0-1 Knapsack - Video Example



0-1 Knapsack - Optimal Solution



Coding Exercise 09b.2

- Write a solution to the fractional knapsack problem in Python that uses the greedy strategy

Dynamic Programming

Dynamic Programming

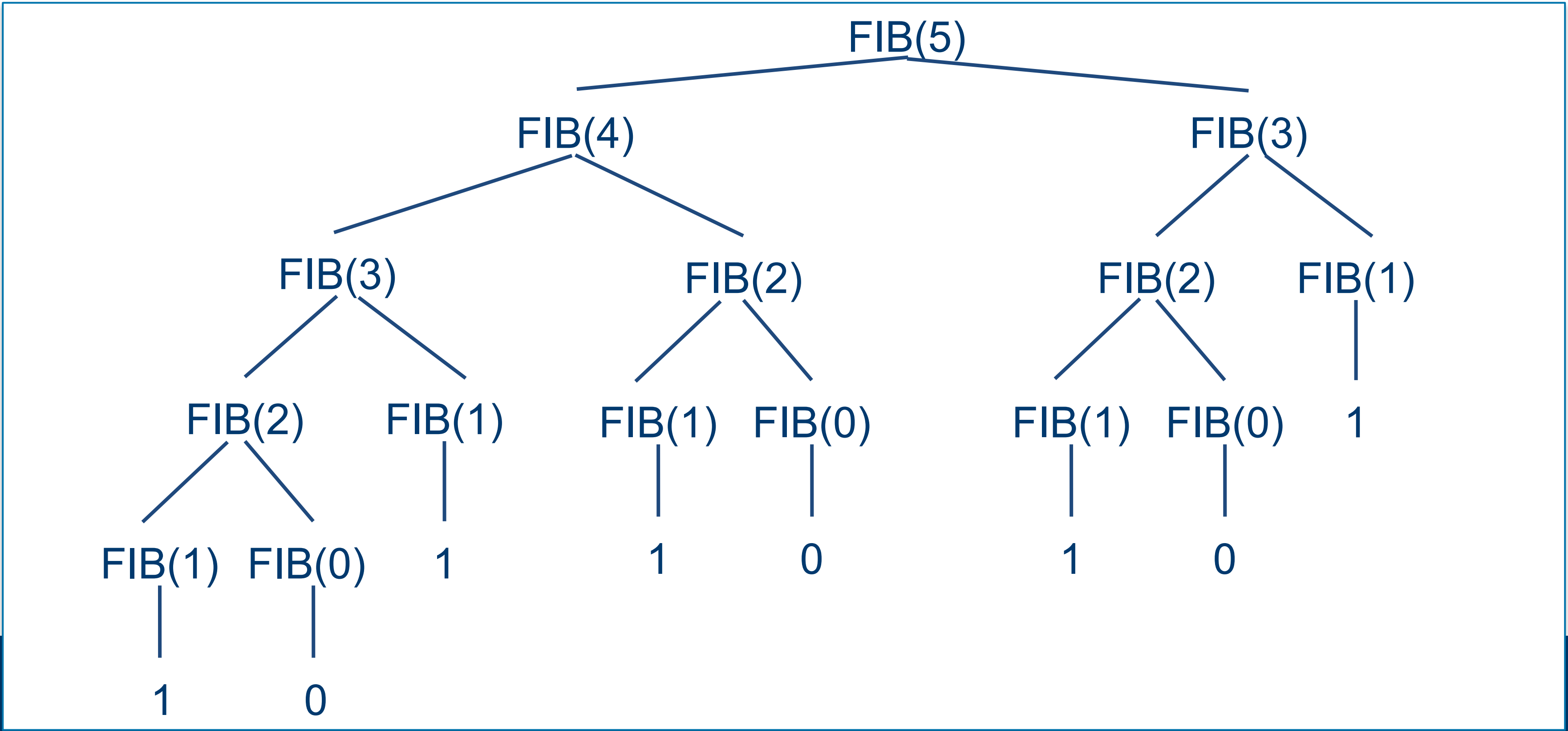
- Dynamic programming involves re-using stored solutions that you have calculated before
- DP is useful when an algorithm must calculate the same sub-solutions again and again
  - Essentially, you store the solutions in a table
  - When you encounter a sub-problem that has yet to be solved, solve it and put the solution into the table
  - When you encounter a sub-problem that has already been solved, look up the solution in the table

Fibonacci - Divide and Conquer

- Calculating the n^{th} Fibonacci number
- Divide and conquer approach:

```
FIBONACCI-DC(n)
if n <= 1 then
    return n
return FIBONACCI-DC(n – 1) + FIBONACCI-DC(n – 2)
```

Fibonacci - Divide and Conquer

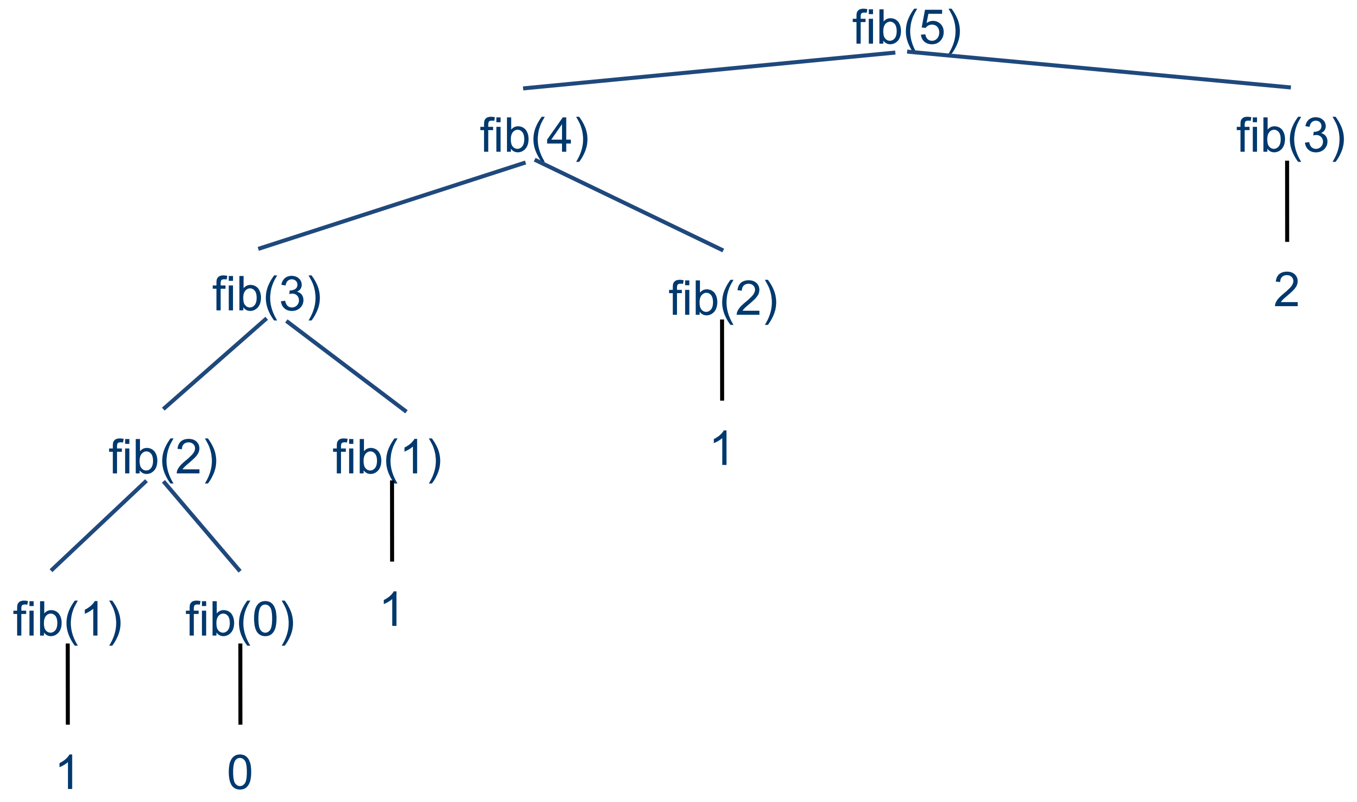


Fibonacci - Dynamic Programming

- We keep calculating the same numbers again and again
- Dynamic programming approach:

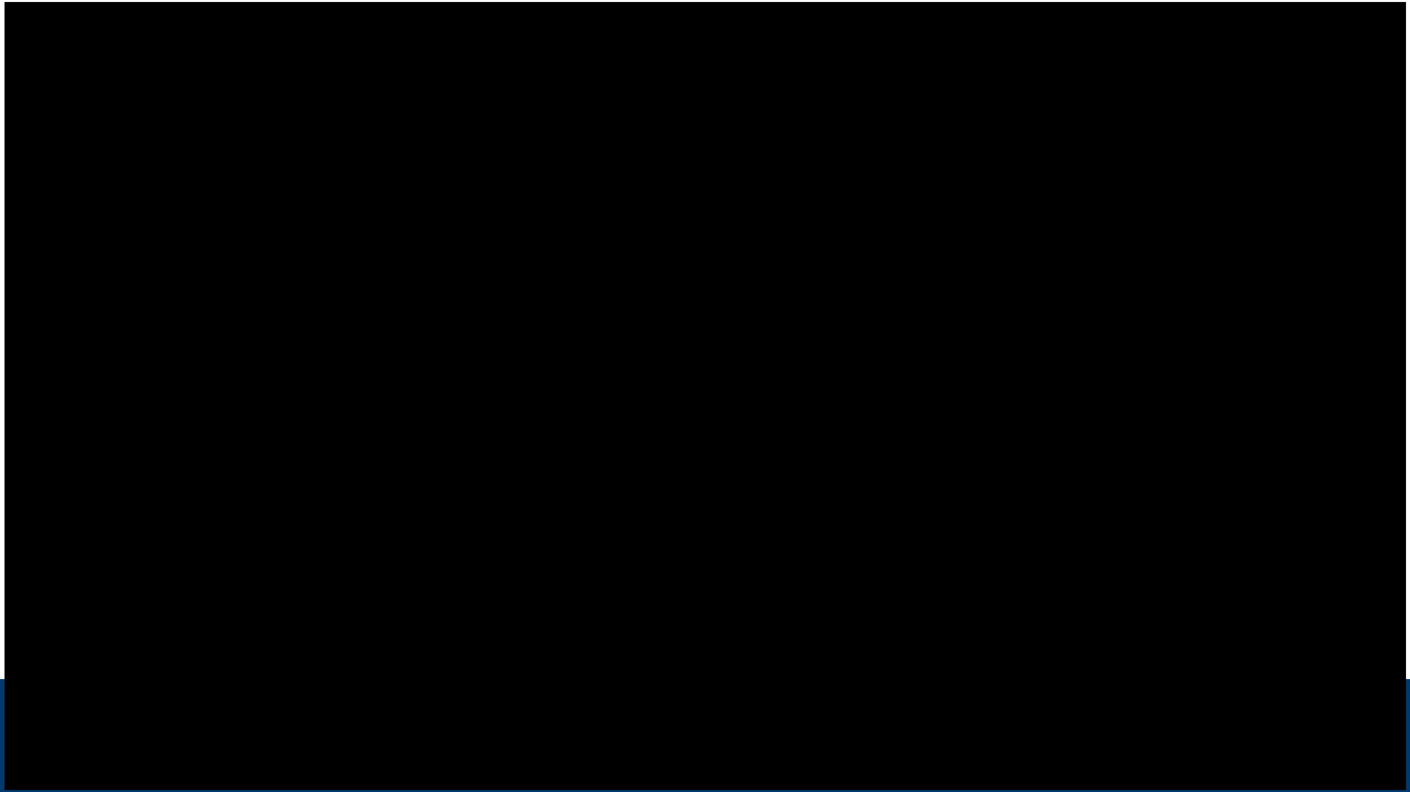
```
FIBONACCI-DP(n)
soln s = [0, 1]
if n < 2 then
    return solns[n]
for i = 2 to n do
    append (solns[i-1] + solns[i-2]) to solns
return solns[n]
```

Fibonacci - Dynamic Programming



Fibonacci - Dynamic Programming

Fibonacci - Dynamic Programming



Fibonacci - Dynamic Programming

- We keep calculating the same numbers again and again
- Dynamic programming approach:

```
FIBONACCI-DP(n)
1. solns = [0, 1]
2. if n < 2 then
3.     return solns[n]
4. for i = 2 to n do
5.     append (solns[i-1] + solns[i-2]) to solns
6. return solns[n]
```

Coding Exercise 09b.3

- Write the dynamic programming solution to the Fibonacci numbers problem in Python

Want to Learn More?

- An investigation into a famous CS problem (TSP), and various algorithms to solve it:
  - https://www.youtube.com/watch?v=GiDsjIBOVoA

Algorithm Strategy Discussion

- MergeSort
  - Divide the list into two equal-sized pieces (A and B)
  - Recursively sort A
  - Recursively sort B
  - Merge A and B together in the proper order

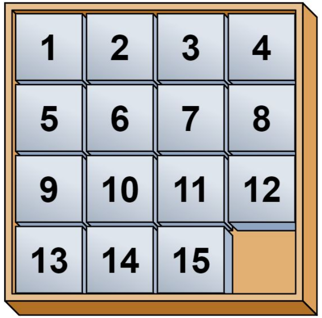
Algorithm Strategy Discussion

- MergeSort
  - Divide the list into two equal-sized pieces (A and B)
  - Recursively sort A
  - Recursively sort B
  - Merge A and B together in the proper order

- Which strategy is used by MergeSort?

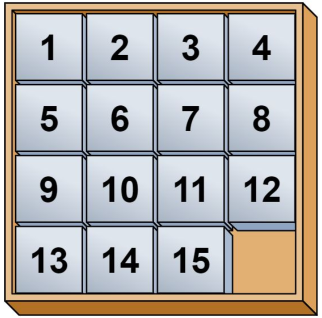
Algorithm Strategy Discussion

- A puzzle game
  - A grid of 4x4 pieces can be slid left/right and up/down
  - There is one open space
  - The goal is to put the pieces in order
  - The same game state can be reached in many different ways



Algorithm Strategy Discussion

- A puzzle game
  - A grid of 4x4 pieces can be slid left/right and up/down
  - There is one open space
  - The goal is to put the pieces in order
  - The same game state can be reached in many different ways



- Which strategy seems most appropriate for this problem?

Wrap-up

- Algorithm strategies
  - Divide and conquer
    - Binary search
  - Greedy Algorithms
    - Fractional knapsack problem
  - Dynamic Programming
    - Fibonacci

Ontario Tech UNIVERSITY

Coming Up

- Basic data structures:
  - Stacks
  - Queues