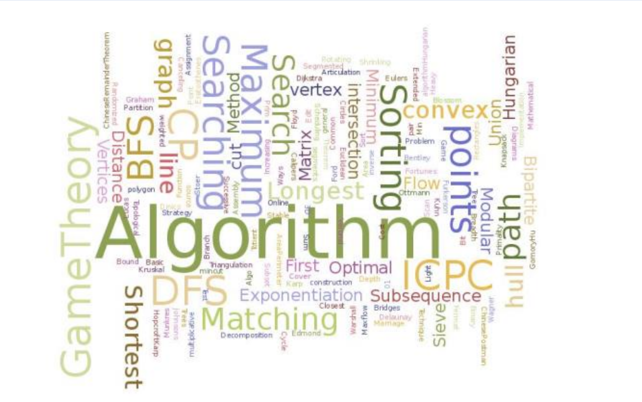
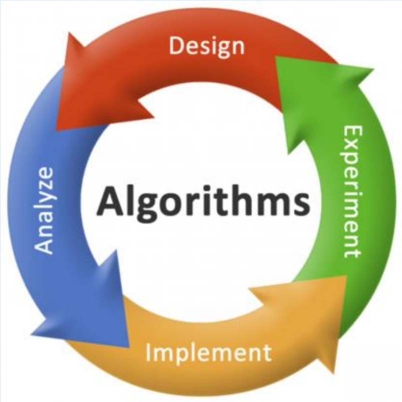
Dynamic Programming Introduction

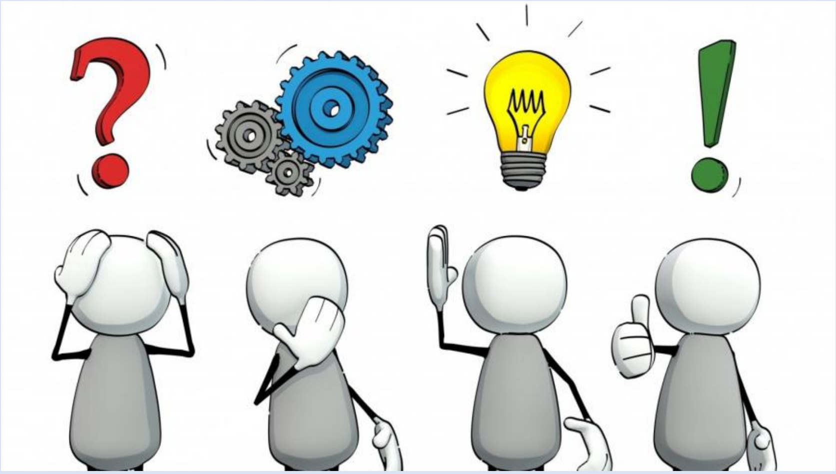
COP 3503
Fall 2021
Department of Computer Science
University of Central Florida
Dr. Steinberg

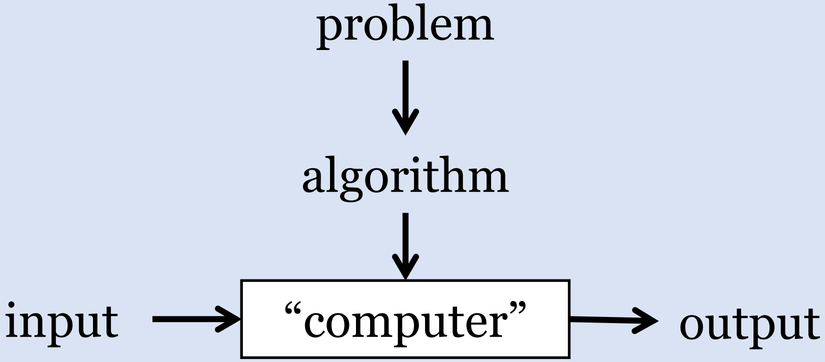




What is an Algorithm? (review)

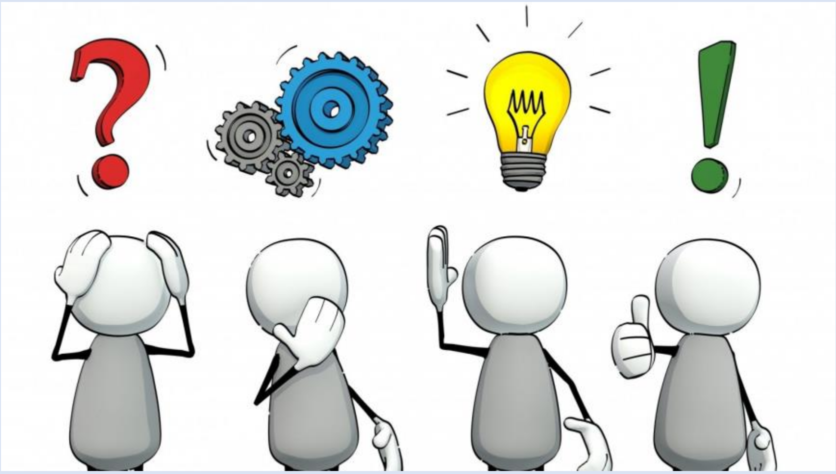
- A well-defined computational procedure which takes a value (or even set of values) as input and produces a value (or set of values) as output.
- An algorithm is said to be **correct** if, for every input instance, it halts with the correct output.





The Output Produced by our Algorithms

- Something to consider with our problems we are solving as programmers and computer scientists.
- Does there exist a group of solutions to a problem?



Greedy Algorithms (Review)

- Our objective is to produce the best output to a solution.
- Greedy algorithms incorporate the concept of making the best the decision at the current moment (without looking at the big picture overall).
- Greedy algorithms make a greedy choice
  - This results in looking at only one subproblem.
- Does a greedy algorithm produce the optimal solution always?
  - **NO!**

Dynamic Programming Introduction

- Dynamic Programming is a technique. Not an algorithm.
  - Like Divide and Conquer and Backtracking
- Dynamic Programming is applied to optimization problems.
  - Finding the Maximum
  - Finding the Minimum
- Dynamic Programming is applicable when the subproblems are not independent. The subproblems share subsubproblems.
  - Dynamic Programming solves the subproblem and stores the result to be used later.
- This allows for optimal solutions always.

Fibonacci Series

- The series
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- Many Approaches to solving this problem.
- Recursive
- Dynamic Programming

Recursive Approach

Algorithm 1 Fibonacci (n)

```
if n <= 1 then
    return n
end if
return Fibonacci (n − 1) +Fibonacci (n − 2)
```

Running Time Complexity: $O(2^n)$

Memoization

- Relieve the potential inefficiency of recursion by using the basic idea of dynamic programming.
- With Fibonacci, we were using recursion without storing the result.
  - This can be very bad in terms of large recursion calls.
- The idea is we can store a previous result that will be used in a later subproblem. (Dynamic Programming)
- Memoization can help reduce running time complexity

DP Approach with Memoization

Algorithm 1 memoized_fibonacci (n)

```
for i = 1 to n do
    results[i] = −1
end for
return memoized_fibonacci_recurs(results, n)
```

Algorithm 2 memoized_fibonacci_recurs (results, n)

```
if results[n] != −1 then
    return results[n]
end if
if n == 1 then
    val = 1
else if n == 2 then
    val = 1
else
    val = memoized_fibonacci_recurs (results, n − 2)
    val = memoized_fibonacci_recurs (results, n − 1)
end if
results[n] = val
return val
```

Remember making change?

Remember the Greedy Algorithm

<u>MakeChangeGreedy(n)</u>

$q = \left\lfloor \frac{n}{25} \right\rfloor$

$n_q = n \; mod \; 25$

$d = \left\lfloor \frac{n_q}{10} \right\rfloor$

$n_d = n_q mod \; 10$

$k = \left\lfloor \frac{n_d}{5} \right\rfloor$

$n_k = n_d \; mod \; 5$

$p = n_k$

Let’s Derive the Dynamic Programming Solution

Dynamic Programming Problems We Will Observe

- 0-1 Knapsack
- Longest Common Subsequence (LCS)
- Sequence Alignment
- Matrix Chain Multiplication