Week 5 Day 1 (Tuesday) Lecture Notes

Prep:

- **Question Set 3:** DVP Chapter 6. Dynamic Programming
- Recitation on Wednesday to review for Quiz on Thursday.
- Read DVP **Chapter 6.4:** The Knapsack Problem.

Dynamic Programming: Knapsack

Planning for Mars One Mission (Knapsack)

The 0-1 Knapsack Problem

- Each item has its own weight and value.
- For 0-1 **You can only have ONE** of each type of item.
- How do you maximize the value you can take?

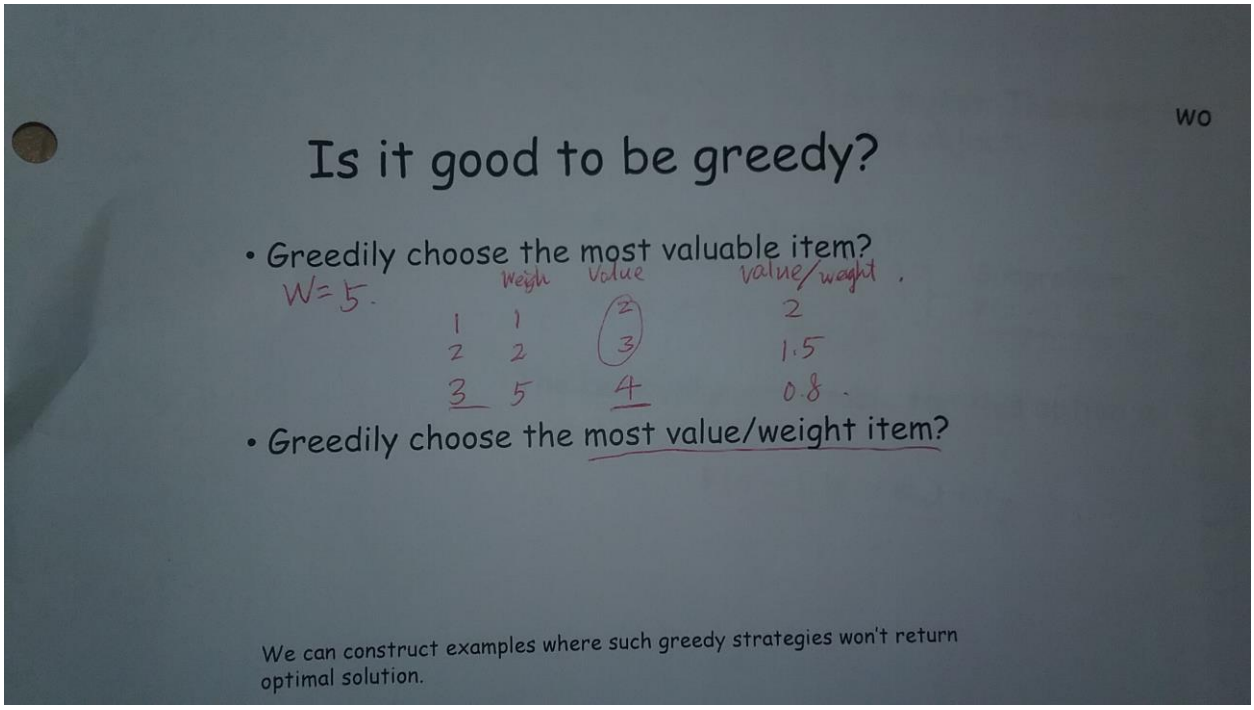
Example: I found this **YouTube Video** a very helpful visualization of Dynamic Programming with the Knapsack problem.
https://www.youtube.com/watch?v=8LusJS5-AGo

(0/1 means you can only pick at most 1 of each item)

Can we brute force this problem?

- Runtime of 2^n possible choices. Enumerate all possible subsets.
- Very expensive to do this brute force method. Exponential complexity means any problem of significant size is unrealistic to solve in a meaningful timeframe.

Being Greedy?



Most valuable item won't work because the weight might be very expensive.

- What about the most value/weight items? This will give the best ratio for a single item. But will not always be optimal because you must consider the combinations of items.
- You might have lots of "extra weight."

Optimal Substructure

Our problem is determined by two things.

1. Number of items to choose from.

2. The maximum Weight Limit.

Option 1: use the object.

If we choose to pull the last object in our set. We reduce the object input size (n) by 1 and we reduce the Weight we have available by how much the object n weights.

Option 2: don't use this object:

If we don't pick the object n. We still remove it from our sub problem list but the weight will not change.

Consider Both options: Then pick the maximum value between those two.

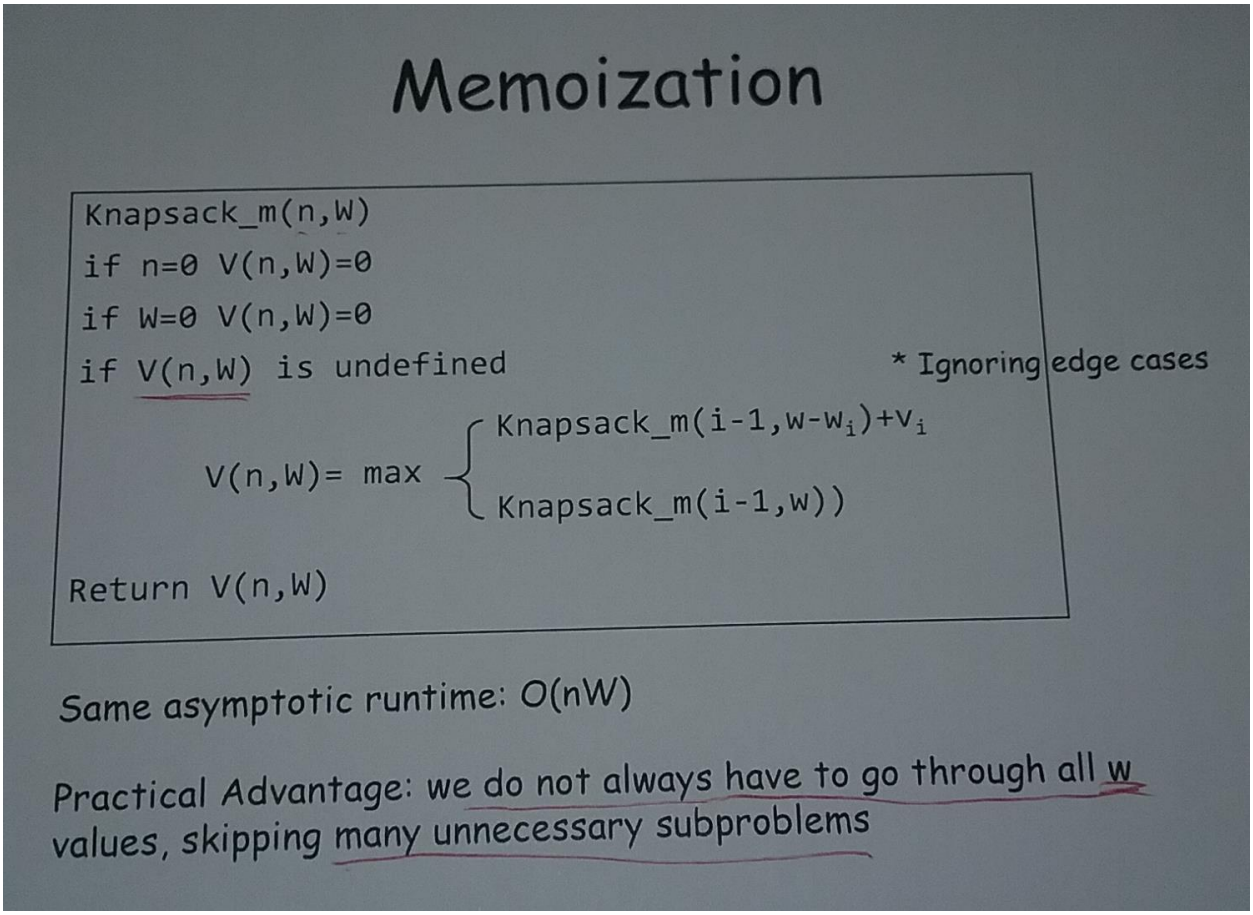
Base cases; when the weight is 0 or we have no objects to take.

Pseudocode for Iterative Version.

(See Pseudocode Picture) Runtime will be the loops 1 to n and 1 – w so O(n * W) where n is the number of items and W is the weight we can carry.

This iterative approach inefficient for enumerations that are smaller than the smallest item weight.

Memoization: Slight change to the Pseudocode.



- In Memoization we have a stored global space where we can look up values. Recursively solve the problem.

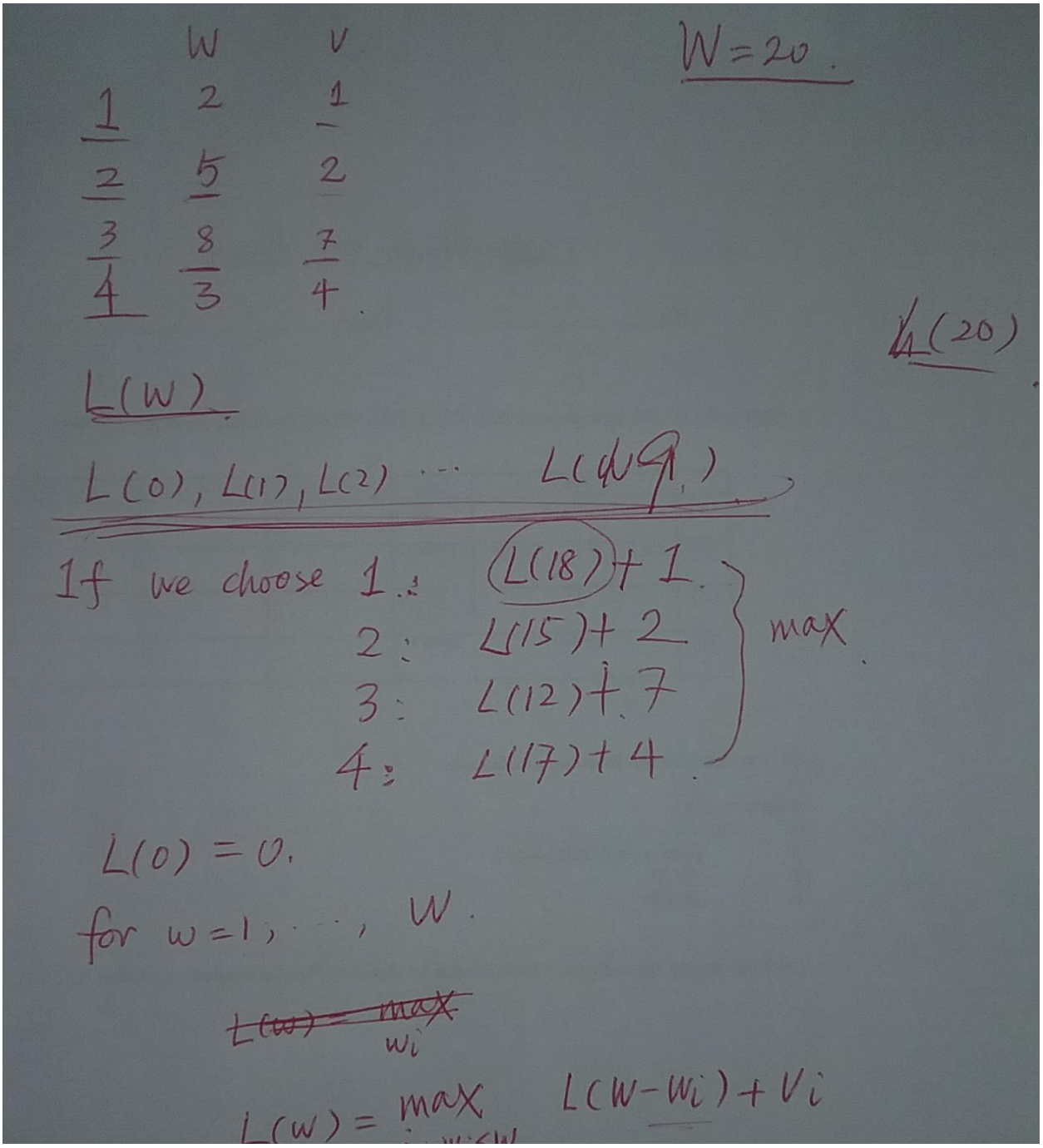
Avoids repeated computations but only computes sub-problems that need to be solved. Memoization has an edge is runtime speed over an iterative approach.

Unbounded Knapsack Problem: Unlimited Number of supply of each item.

- Compared to the 0-1 Knapsack we can't reduce the size of n. But we can change the weight.

- How will you choose which items to put into your Knapsack? You still want to have the maximum value/weight items.

Example in Class:



Ending Notes:

- **Quiz 3** will be next Thursday (Focus on Dynamic Programming and Knapsack)
- (Recitation for Quiz Review at normal location **Wednesday 5:45-6:45pm**)
- **Instructor is working on Assignment 1 Grades.**

Finished Knapsack Lecture (W5D1)

End of Week 4 Thursday Lecture Notes

~Information composed by Notetaker Scott Russell for CS 325 **DAS** student