# Abstract:

## Instructions:

* + - * “In a paragraph, give a summary of what the problem is trying to solve, what kinds of approaches there are to solving the problem (naive divide-and-conquer, dynamic programming, backtracking, etc.), and what kind of Big O time complexities it may have for those given approaches.”

## the problem is trying to solve

* + - a

## kinds of approaches there are to solving the problem

* + - a

## kind of Big O time complexities it may have for given approaches

* + - a

# Problem Statement:

## Instructions:

* + - “Describe what the knapsack problem is in detail. Describe its history and why its such an important problem in computer science. If there are any variants to the primary knapsack problem, shortly list and describe what the variant is and why it’s included.”

## Introduction

## What is the knapsack problem

## What is the knapsack problem

* + - High Level
      * Modified Thief example from [Bhargava]
        + Imagine a thief attempting to steal valuable items from a treasure trove. He breaks in with only his knapsack, which has a weight limit of 5kg. He sees three items:

A priceless artifact [100k, 4kg]

a rare antique watch [80k, 2kg]

Ornate silver vase [90k, 3kg]

* + - * + He grabs most expensive item, A priceless artifact, and realizes he cannot carry any more and makes a break for it. However, as he is making his getaway, he realizes his mistake – while the single item has more value, it took up more than half of the weight. He should have grabbed the other two items, which would have been worth more collectively, while still being within the weight limit.
      * *While in this example, size and dimensions are not considered, ….*
      * “In the Knapsack Problem, a knapsack has a specific maximum weight that it can hold. Several items are available to be stored in the knapsack, and each item has a different weight and value. The goal is to fit as many items into the knapsack as possible so that the total value is maximized and the total weight does not exceed the knapsack’s limit. The physical size and dimensions of the items are ignored in the simplest variation of the problem” [Hurbans]
      * Has a set total weight capacity, and can hold any of the items
      * “the possibilities explode as the number of potential items increases.” [Hurbans]
      * “It will also be computationally expensive to try to brute-force every combination of items when the variables grow; thus, we look for algorithms that are efficient at finding a desirable solution.” [Hurbans]

## History

* + - A

## Importance

* + - “used in computer science to explore how algorithms work and how efficient they are.” [Hurbans]

## Variants

* + - a

# Algorithms:

## Instructions:

* + - “You must present pseudocode for at least two different algorithms in your report that solve the knapsack problem. The algorithms must be from different paradigms (dynamic programming, backtracking, naive Divide and Conquer, etc.). New and obscure implementations are welcomed, but not necessary.”

## Introduction

* + - Describe Brute Force briefly and why it isn’t a good option

## Algorithm 1: Dynamic Programming

* + - Dynamic Programming
      * Solves problems by breaking them down into smaller subproblems and solving those [Bhargava]
    - Knapsack [Bhargava]
      * Break the knapsack into smaller knapsacks, solve those, and work up towards the original one
      * To do this, the algorithms starts with a grid with items as the rows and columns as weights

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 2 | 3 | 4 | 5 |
| Artifact |  |  |  |  |
| Watch |  |  |  |  |
| Vase |  |  |  |  |

* + - * + Each column represents a smaller knapsack
        + Since our smallest item weighs 2kg, we don’t include column 1
      * The table is filled out, row by row, asking at each cell “will stealing this item give me the most value”?
      * Once you continue to the next row, you can either steal the item at that row or from any of the rows above it
    - Psuedo Code

## Algorithm 2:

* + - A

## Misc

* + - “Performance is defined as how well a specific solution solves a problem, not necessarily computational performance.” [Hurbans]

# Time Complexity:

## Instructions:

* + - “For each algorithm presented above, cite your sources and you may use their justification for the time-complexity. Feel free to utilize online resources to help in this, but be comfortable enough that if you were asked questions about the work, you could answer them. Huge leaps in logic or math will likely be met with questions. YOU DO NOT NEED TO PROVE ALGORITHM CORRECTNESS OR TIME COMPLEXITY. We will assume correctness for now.”

## Algorithm 1:

* + - a

## Algorithm 2:

* + - A
* MISC

# Code:

## Instructions:

* + - “Write up the algorithms you presented above in the coding language of your choice. Using similar input, note the difference in their respective observed runtimes in your report and why there might be deviations from our expectations of the time complexity given above. Provide screenshots and instructions on how to run your programs in your report. Also, submit these code files along with your report.”

## A

* + - a

# Sources [Citations]:

## Hurbans

## Grokking Artificial Intelligence Algorithms

* + - Rishal Hurbans
    - https://learning.oreilly.com/library/view/grokking-artificial-intelligence/9781617296185/

## Bhargava

* + - Grokking Algorithms
    - Aditya Bhargava
    - https://learning.oreilly.com/library/view/grokking-algorithms/9781617292231/
  + G