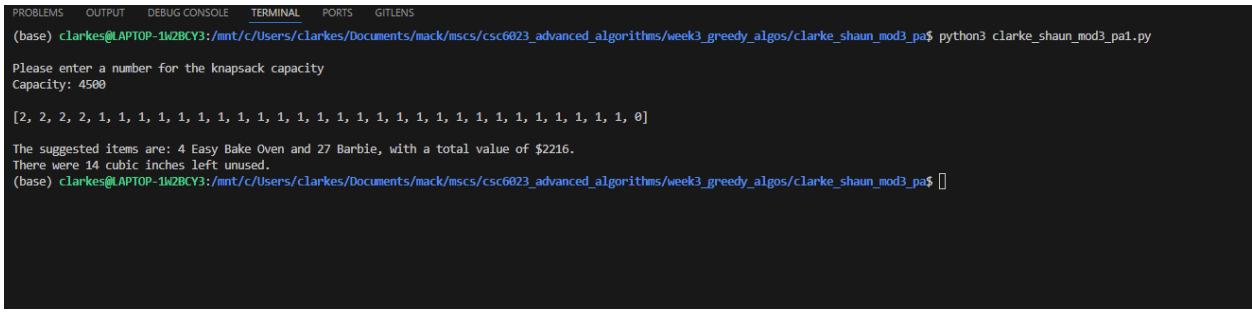
1. What is the Big Oh
    a. Factorial1: O(n)
    b. Factorial2: O(n)
    c. Fibonacci1: O(n)
    d. Fibonacci2: O(2^n)
2. Towers of Hanoi
    a. Worst case number of moves would be E, O(2^n)
    b. Reasoning:
        i. The recurrence T(n) = 2T(n−1) + 1 means each time we add a disk, the problem size doubles because we have to move all the smaller disks twice. So solving it gives T(n) = 2^n − 1, which basically blows up.
3. Greedy Algorithms
    a. 

4. Amortized Analysis of Quicksort Algorithm
    a. Quicksort is *usually* O(n log n) because, on average, you could say the pivot divides the table into equal halves each time. That means we do log n levels of work, and each level handles n elements.
    b. The worst case O(n^2) happens only when a good pivot was not chosen, like when the array is already sorted, and we always pick the first or last item as pivot. But when the pivots are chosen randomly, the chance of selecting a bad one repeating many times is extremely low.
5. Linear Programming

    x = boxes of cookies
    y = boxes of muffins

    Objective Function
    $P = 150x + 100y$

    Constraints
    $4x + 3y \leq 80$ (flour)
    $5x + y \leq 80$ (milk)
    $x \geq 0$
    $y \geq 0$

Cookies Only
$y = 0$
$5x \leq 80$
$x = 16$
$P = 150 \times 16 = 2400$

Muffins Only
$x = 0$
$3y \leq 80$
$y = 26$
$P = 100 \times 26 = 2600$

Balanced Options
$5x + y = 80$
$y = 80 - 5x$
$4x + 3(80 - 5x) = 80$
$4x + 240 - 15x = 80$
$-11x = -160$
$x = 14$
$y = 80 - 5(14)$
$y = 10$
$P = 150(14) + 100(10)$
$P = 2100 + 1000 = 3100$

Best outcome is the balanced option
Only cookies: Profit for 16 boxes of cookies = $2400
Only muffins: Profit for 26 boxes of muffins = $2600
Cookies and Muffins: Profit for 14 boxes of cookies and 10 boxes of muffins = $3100

6. Randomized algorithm to find an element in an array:
    a. Reused most of the code from the assignment.

```
from typing import List
import random

class GenerateArrays:
    # This method generates a list with a specific count of a number
    @staticmethod
    def generate_list_of_nums(num: int, total_items: int) -> List[int]:
        # number of items to generate
        number_of_items: int = total_items
        # Generating list of items
        list_of_itemss: List[int] = [num] * number_of_items
        return list_of_itemss
```

```
class ArrayTools:
    def __init__(self, array1: List[int], array2: List[int]):
        self.array1: List[int] = array1
        self.array2: List[int] = array2
        self.joined_array: List[int] = []

    # This method joins two lists
    def join_lists(self) -> List[int]:
        # Joining lists
        self.joined_array: List[int] = self.array1 + self.array2
        return self.joined_array

    # This method shuffles the list.
    def shuffle_list(self) -> List[int]:
        # Shuffling joined list.
        random.shuffle(self.joined_array)
        return self.joined_array



# 1) This function creates and returns a randomized array of 1000 elements,
# As per question requierment, each being 2, 3, or 4 with roughly the same
probability.
def alea() -> List[int]:
    twos   = GenerateArrays.generate_list_of_nums(2, 334)
    threes = GenerateArrays.generate_list_of_nums(3, 333)
    fours  = GenerateArrays.generate_list_of_nums(4, 333)

    # Using ArrayTools to join and shuffle
    tools_1 = ArrayTools(twos, threes)
    first_join = tools_1.join_lists()

    tools_2 = ArrayTools(first_join, fours)
    full_array = tools_2.join_lists()
    tools_2.shuffle_list()

    return tools_2.joined_array


# 2) Monte Carlo, pick one random index and
#    print the index only if the the value is 2.
def pick(a: List[int]) -> None:
    i = random.randrange(len(a))
    if a[i] == 2:
        print(i)
```

```
def main():
    arr = alea()
    pick(arr)


if __name__ == "__main__":
    main()
```

7. Advanced Data Structures

```
8. # UFO/UAP adjacency matrix
9. [ [0,442,  4,  0,  0,1344,  0,  0],
10.  [442, 0,  0,  0,842, 932,  0,  0],
11.  [  4, 0,  0,437,  0,   0,  0,  0],
12.  [  0, 0,437,  0,  0, 399,337,  0],
13.  [  0,842, 0,  0,  0, 376,  0,345],
14.  [1344,932, 0,399,376,  0,  0,154],
15.  [  0,  0, 0,337,  0,  0,  0,221],
16.  [  0,  0, 0,  0,345,154,221,  0] ]
17.
```

a. Shortest distance from Earth to Orion is 994 lightyears.

b. Stops on that route:

i. Earth to Alpha Centauri to Vela Molecular Ridge to Orion Nebula to Orion.

8. a. For computing the value of pi

An approximated solution is reasonable because the exact value of π cannot be found it's an irrational and infinite number. Computers can only handle finite precision, so we can only compute an approximation that's close enough.

b. Machine learning algorithm based on decision trees

Machine learning always gives an approximate best effort model. It's impossible to test or train on every possible input since the number of possible situations is infinite, so decision trees aim to capture useful patterns that work most of the time.

c. For the traveling salesperson problem

An exact solution would be too costly. For large numbers of cities, that's impossible to solve quickly. Approximated or heuristic algorithms give a near optimal route much faster, which is good enough for real world use cases.

d. For a strategy game available on a mobile device

An approximated algorithm is reasonable because a perfect strategy would really tax the device's resources. The game doesn't exactly need to be perfect, just good enough to keep the users engaged and run efficiently to keep them happy.