North South University

Department of Electrical & Computer Engineering

<u>Topic</u>: Rod-Cutting Problem

<u>Submitted By</u>:

Name: Tasnim Shahrin. (202 2025 042)

Abu Mukaddim Rahi. (202 2027 042)

Khondokar Nazmul Dihan (202 1894 642)

Group: 01

Section : 12

Faculty : MLE.

Course Code: CSE373.

Spring 2023.

<u>Submitted To:</u>

Mirza Mohammad Lutfe Elahi

<u>Submission Date: 11.06.2023</u>

Introduction:

In the course '**Design and analysis the Algorithms**' we have designed **Rod-Cutting Problem**. As our given instruction we need to use minimum 3 types of algorithms to solve the problem; they are- Dynamic, Greedy, Brute Force Algorithm.

Analysis of Algorithm:

The most basic part of this Rod-Cutting Problem is given below. No matter which types of algorithms we used in our project the basic format algorithm of Rod-Cutting problem will always remain same.

```
import random
import matplotlib.pyplot as plt
import time

```

```
def generate_random_rods(n):
    lengths = []
    prices = []
    for _ in range(n):
        length = random.randint(1, 10)
        price = random.randint(1, 100)
        lengths.append(length)
        prices.append(price)
    return lengths, prices
```

We have imported these libraries to perform specific tasks related to generating random numbers, plotting, or measuring time.

The **generate_random_rods** function generates **n** random rod lengths and prices by iterating **n** times and using the **random.randint** function to generate random values within the specified ranges. The function then returns the generated lengths and prices as separate lists.

1. **Dynamic Programming:** The dynamic programming algorithm for the rod-cutting problem has a time complexity of O(n^2), where n is the length of the rod. This algorithm builds a table of optimal solutions for subproblems and uses that information to find the optimal solution for the entire problem.

2. **Greedy Algorithm:** The greedy algorithm for the rod-cutting problem has a time complexity of O(n), where n is the length of the rod. This algorithm makes a locally optimal choice at each step by selecting the largest possible cut and repeats this process until the entire rod is cut. The greedy approach does not consider future choices or optimize the overall solution globally.

3. **Brute Force Approach:** The brute force approach involves exploring all possible combinations of cuts and finding the best solution. The time complexity of the brute force approach for the rod-cutting problem is exponential, specifically O(2^n), where n is the length of the rod. This is because the number of possible cuts grows exponentially with the length of the rod.

Among the three complexity notations, O(n) outperforms best, followed by O(n^2), and O(2^n).

O(n) > O (n^2) > O (2^n); [Greedy algo. > Dynamic algo. > Brute Force algo.]

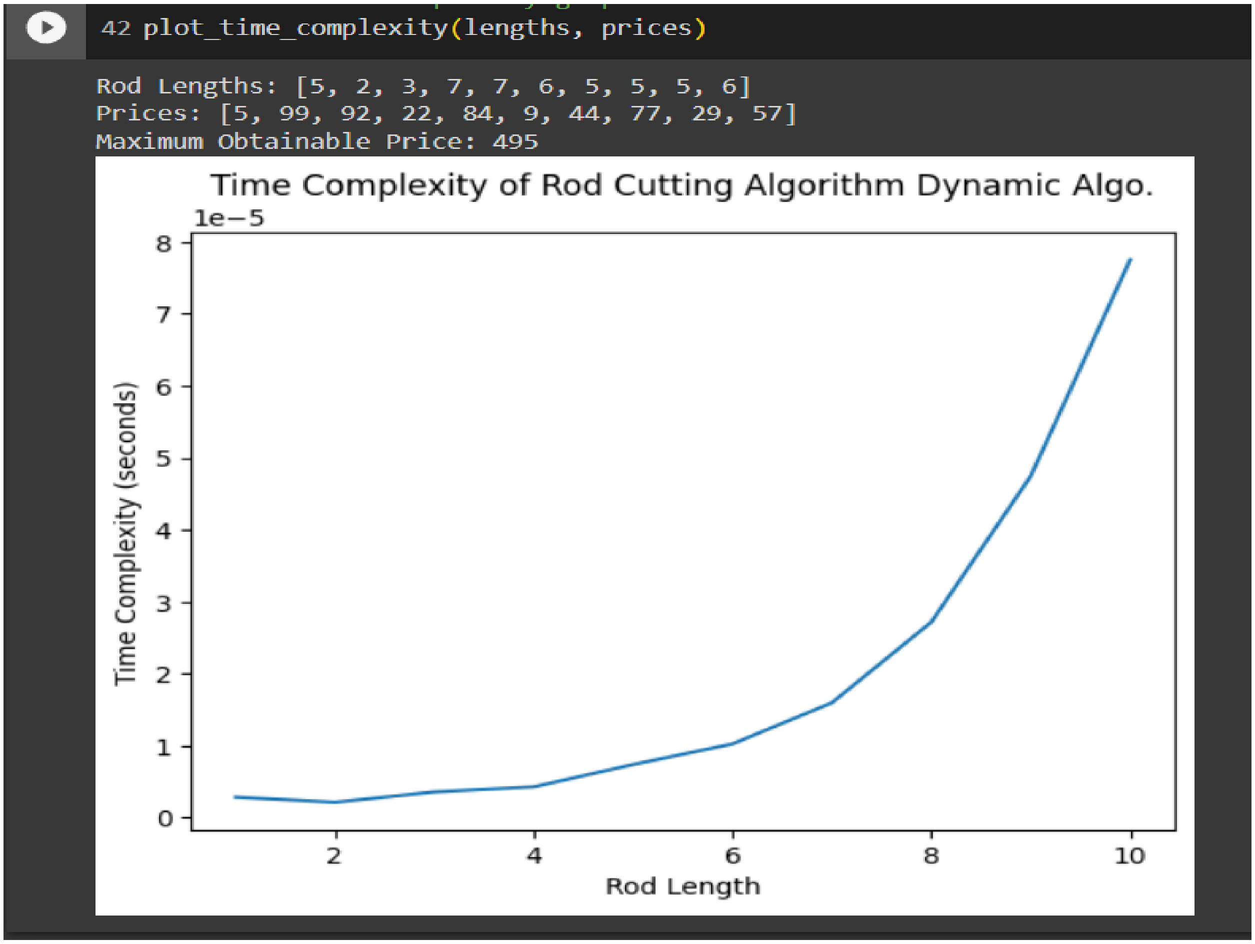
O(n) denotes linear time complexity, which means an algorithm's running time increases linearly with the input size. This is considered efficient and works well in most practical cases.

O(n^2) denotes quadratic time complexity, which means that the running time of an algorithm grows quadratically with the size of the input. It is less efficient than linear time complexity, but it is still suitable for small input sizes. However, as the size of the input increases, the running time can become significantly slower.

O(2^n) stands for exponential time complexity, which means that an algorithm's running time grows exponentially with input size. Exponential algorithms are notoriously inefficient, especially for larger input sizes. The running time quickly increases and can become unmanageable, making it impractical for most scenarios.

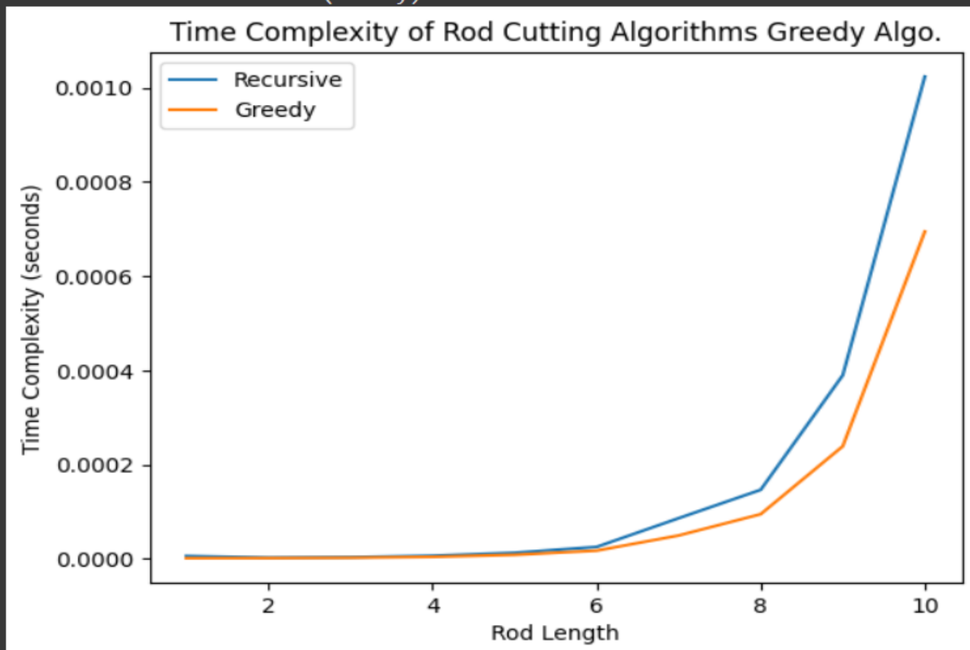
In summary, the most efficient is O(n), the least efficient is O(n^2), and the least efficient is O(2^n). The complexity level chosen is figured out by the Rod-Cutting problem, and acceptable performance trade-offs.

Experimental Result (Time Complexity Graph):

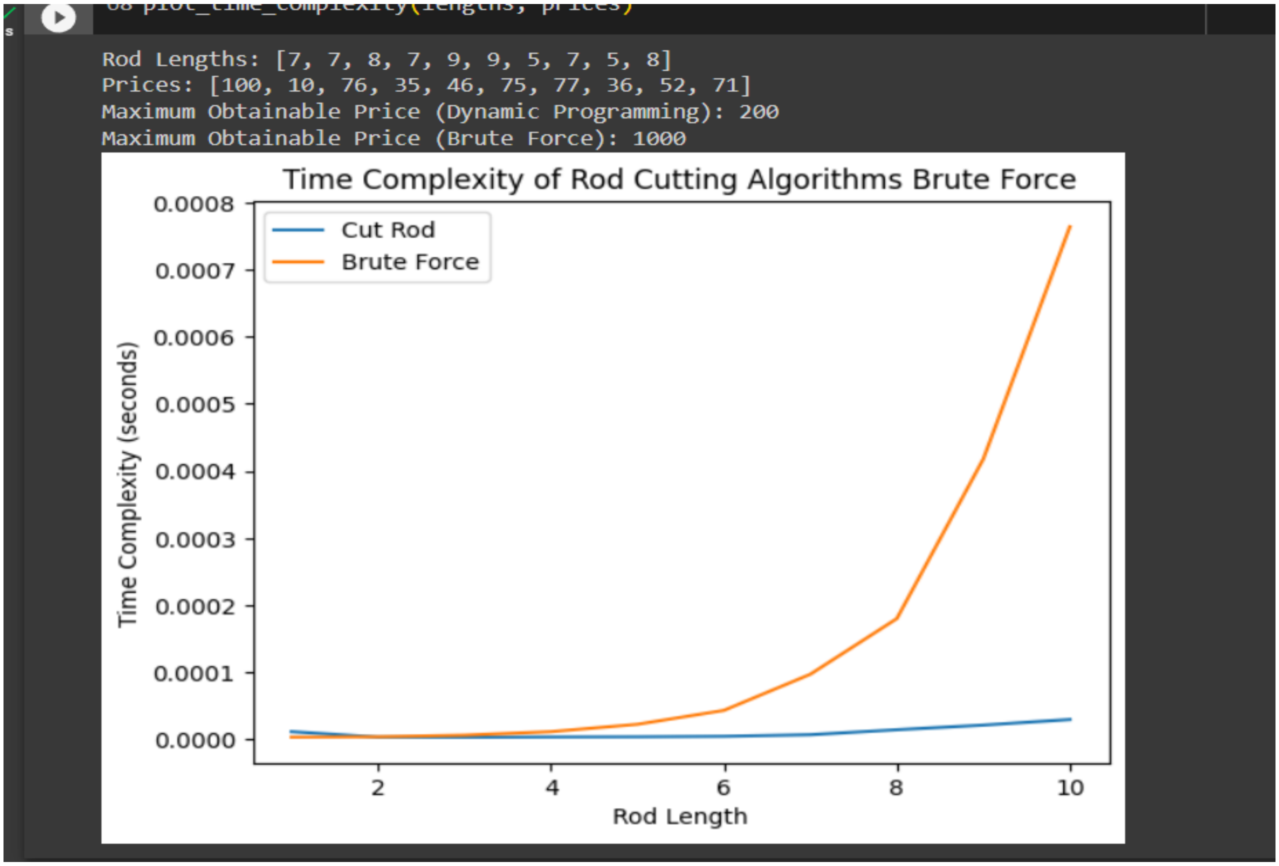


```
plot_time_complexity(lengths, prices)
```

```
Rod Lengths: [6, 2, 1, 2, 7, 9, 2, 7, 3, 2]
Prices: [12, 76, 54, 100, 82, 19, 36, 42, 8, 65]
Maximum Obtainable Price (Recursive): 508
Maximum Obtainable Price (Greedy): 508
```



0s completed at 12:22 AM



Conclusion:

In conclusion, we can see the time complexity of the rod-cutting problem is O(n^2) for dynamic programming, O(n) for the greedy algorithm, and O(2^n) for the brute force approach. The dynamic programming and greedy algorithms offer more efficient solutions compared to the brute force approach, with the dynamic programming approach providing optimal solutions.

Code: (Dynamic Algorithm)

```
import random

import matplotlib.pyplot as plt
```

```
import time


def generate_random_rods(n):
    lengths = []
    prices = []
    for _ in range(n):
        length = random.randint(1, 10)
        price = random.randint(1, 100)
        lengths.append(length)
        prices.append(price)
    return lengths, prices

def cut_rod(lengths, prices, n):
    if n <= 0:
        return 0

    max_price = float('-inf')
    for i in range(n):
        max_price = max(max_price, prices[i] + cut_rod(lengths, prices, n -
lengths[i]))

    return max_price


def plot_time_complexity(lengths, prices):
    n_values = []
    time_values = []
```

```
    for n in range(1, len(lengths) + 1):
        start_time = time.time()
        cut_rod(lengths, prices, n)
        end_time = time.time()
        elapsed_time = end_time - start_time

        n_values.append(n)
        time_values.append(elapsed_time)

    plt.plot(n_values, time_values)
    plt.xlabel('Rod Length')
    plt.ylabel('Time Complexity (seconds)')
    plt.title('Time Complexity of Rod Cutting Algorithm')
    plt.show()

# Generate random rod lengths and prices
lengths, prices = generate_random_rods(10)

# Print the generated rod lengths and prices
print("Rod Lengths:", lengths)
print("Prices:", prices)

# Calculate and print the maximum obtainable price
max_price = cut_rod(lengths, prices, len(lengths))
print("Maximum Obtainable Price:", max_price)
```

```
# Plot the time complexity graph
plot_time_complexity(lengths, prices)
```

(Greedy Algorithm)

```
import random
import matplotlib.pyplot as plt
import time


def generate_random_rods(n):
    lengths = []
    prices = []
    for _ in range(n):
        length = random.randint(1, 10)
        price = random.randint(1, 100)
        lengths.append(length)
        prices.append(price)
    return lengths, prices
def cut_rod(lengths, prices, n):
    if n <= 0:
        return 0

    max_price = float('-inf')
    for i in range(n):
```

```
        max_price = max(max_price, prices[i] + cut_rod(lengths, prices, n -
lengths[i]))

    return max_price


def greedy_cut_rod(lengths, prices, n):
    if n <= 0:
        return 0

    max_price = 0
    for i in range(n):
        if lengths[i] <= n:
            max_price = max(max_price, prices[i] + greedy_cut_rod(lengths,
prices, n - lengths[i]))

    return max_price


def plot_time_complexity(lengths, prices):
    n_values = []
    time_values = []
    greedy_time_values = []


    for n in range(1, len(lengths) + 1):
        start_time = time.time()
        cut_rod(lengths, prices, n)
        end_time = time.time()
        elapsed_time = end_time - start_time
```

```
        start_time = time.time()
        greedy_cut_rod(lengths, prices, n)
        end_time = time.time()
        greedy_elapsed_time = end_time - start_time

        n_values.append(n)
        time_values.append(elapsed_time)
        greedy_time_values.append(greedy_elapsed_time)

    plt.plot(n_values, time_values, label='Recursive')
    plt.plot(n_values, greedy_time_values, label='Greedy')
    plt.xlabel('Rod Length')
    plt.ylabel('Time Complexity (seconds)')
    plt.title('Time Complexity of Rod Cutting Algorithms')
    plt.legend()
    plt.show()

# Generate random rod lengths and prices
lengths, prices = generate_random_rods(10)

# Print the generated rod lengths and prices
print("Rod Lengths:", lengths)
print("Prices:", prices)
```

```
# Calculate and print the maximum obtainable price using the recursive approach
max_price = cut_rod(lengths, prices, len(lengths))
print("Maximum Obtainable Price (Recursive):", max_price)


# Calculate and print the maximum obtainable price using the greedy approach
greedy_max_price = greedy_cut_rod(lengths, prices, len(lengths))
print("Maximum Obtainable Price (Greedy):", greedy_max_price)


# Plot the time complexity graph
plot_time_complexity(lengths, prices)
```

(Brute Force Algorithm)

```
import random
```

```
import matplotlib.pyplot as plt
import time



def generate_random_rods(n):
    lengths = []
    prices = []
    for _ in range(n):
        length = random.randint(1, 10)
        price = random.randint(1, 100)
        lengths.append(length)
        prices.append(price)
    return lengths, prices


def cut_rod(lengths, prices, n):
    if n <= 0:
        return 0

    max_price = float('-inf')
    for i in range(n):
        max_price = max(max_price, prices[i] + cut_rod(lengths, prices, n -
lengths[i]))

    return max_price



def brute_force_cut_rod(lengths, prices, n):
    if n <= 0:
        return 0
```

```
    max_price = 0
    for i in range(1, n + 1):
        max_price = max(max_price, prices[i - 1] + brute_force_cut_rod(lengths,
prices, n - i))

    return max_price


def plot_time_complexity(lengths, prices):
    n_values = []
    time_values_cut_rod = []
    time_values_brute_force = []

    for n in range(1, len(lengths) + 1):
        start_time = time.time()
        cut_rod(lengths, prices, n)
        end_time = time.time()
        elapsed_time_cut_rod = end_time - start_time

        start_time = time.time()
        brute_force_cut_rod(lengths, prices, n)
        end_time = time.time()
        elapsed_time_brute_force = end_time - start_time

        n_values.append(n)
        time_values_cut_rod.append(elapsed_time_cut_rod)
```

```
        time_values_brute_force.append(elapsed_time_brute_force)


    plt.plot(n_values, time_values_cut_rod, label='Cut Rod')
    plt.plot(n_values, time_values_brute_force, label='Brute Force')
    plt.xlabel('Rod Length')
    plt.ylabel('Time Complexity (seconds)')
    plt.title('Time Complexity of Rod Cutting Algorithms')
    plt.legend()
    plt.show()


# Generate random rod lengths and prices
lengths, prices = generate_random_rods(10)


# Print the generated rod lengths and prices
print("Rod Lengths:", lengths)
print("Prices:", prices)


# Calculate and print the maximum obtainable price using dynamic programming
max_price = cut_rod(lengths, prices, len(lengths))
print("Maximum Obtainable Price (Dynamic Programming):", max_price)


# Calculate and print the maximum obtainable price using brute force
max_price_brute_force = brute_force_cut_rod(lengths, prices, len(lengths))
print("Maximum Obtainable Price (Brute Force):", max_price_brute_force)
```

```
# Plot the time complexity graph

plot_time_complexity(lengths, prices)
```