Analisis Efisiensi Perbandingan Algoritma Greedy dan Brute Force dalam Penyelesaian Masalah Job Scheduling

TUGAS BESAR STRATEGI ALGORITMA

Disusun oleh :

Adam Zahran (103012380527)

Firdhaus Dwi Sukma (103012380540)

Si Gede Ngurah Chandra Adi Natha (103012380526)

PROGRAM STUDI S1 INFORMATIKA

FAKULTAS INFORMATIKA

UNIVERSITAS TELKOM

BANDUNG

2024

Pendahuluan

1. Latar Belakang

Job scheduling adalah permasalahan penting dalam ilmu komputer dan penelitian operasional, melibatkan penentuan urutan pengerjaan tugas pada sumber daya untuk memaksimalkan atau meminimalkan suatu fungsi objektif, seperti waktu penyelesaian total atau pemanfaatan sumber daya. Masalah ini sering ditemui dalam konteks produksi, pemrosesan data, logistik, dan manajemen proyek, sehingga solusi efisien sangat dibutuhkan.

Dua pendekatan utama untuk menyelesaikan masalah job scheduling adalah algoritma Greedy dan Brute Force. Algoritma Greedy selalu memilih opsi terbaik pada setiap langkah, sederhana dan cepat, namun tidak selalu menghasilkan solusi optimal. Sebaliknya, algoritma Brute Force mencoba semua kemungkinan solusi untuk menemukan yang terbaik, memastikan solusi optimal tapi dengan biaya komputasi sangat tinggi, membuatnya tidak praktis untuk masalah besar.

Penelitian ini menganalisis dan membandingkan efisiensi kedua algoritma dalam menyelesaikan masalah job scheduling. Analisis mencakup evaluasi performa pada berbagai skenario dan ukuran masalah, mempertimbangkan waktu eksekusi, kompleksitas, dan kualitas solusi. Hasil penelitian diharapkan memberikan panduan praktis dan kontribusi teoretis dalam memilih algoritma yang paling sesuai serta mengembangkan metode penyelesaian yang lebih efisien.

2. Pernyataan Masalah

Algoritma yang kami gunakan adalah Brute Force dan Greedy, permasalahan utama yang dihadapi dalam pengerjaan ini adalah:

1. Bagaimana performa algoritma Brute Force dibandingkan dengan algoritma Greedy dalam optimasi penjadwalan pekerjaan dari segi waktu eksekusi dan kualitas solusi?
2. Apa faktor-faktor yang mempengaruhi perbedaan kinerja antara algoritma Brute Force dan Greedy dalam konteks penjadwalan pekerjaan?
3. Apa implikasi praktis dari perbandingan kinerja kedua algoritma tersebut terhadap pengambilan keputusan dalam aplikasi dunia nyata?

Dengan menjawab pertanyaan-pertanyaan ini, laporan ini bertujuan untuk memberikan wawasan mendalam tentang efisiensi dan efektivitas kedua algoritma, serta menawarkan rekomendasi untuk meningkatkan kinerja aplikasi penjadwalan pekerjaan yang memerlukan optimasi yang akurat dan cepat. Selain itu, laporan ini juga akan mengidentifikasi kondisi di mana masing-masing algoritma lebih cocok digunakan, serta memberikan panduan praktis bagi pengambilan keputusan dalam berbagai konteks aplikasi dunia nyata.

3. Pendekatan Teknis

Laporan ini akan mengambil pendekatan terstruktur dengan langkah awal dalam mengumpulkan data dari sumber kamus bahasa Inggris daring. Selanjutnya, akan dilakukan implementasi dua pendekatan algoritma, yakni Brute Force dan Greedy. Evaluasi hasilnya akan mencakup analisis kualitatif dan kuantitatif, yang akan diikuti oleh serangkaian eksperimen dan analisis mendalam untuk menyusun rekomendasi berdasarkan temuan. Tujuan utama laporan

ini adalah memberikan pemahaman yang komprehensif tentang efisiensi dan efektivitas dari kedua pendekatan algoritma dalam menyelesaikan masalah pemecahan kata.

4. **Hasil dan Pembahasan**

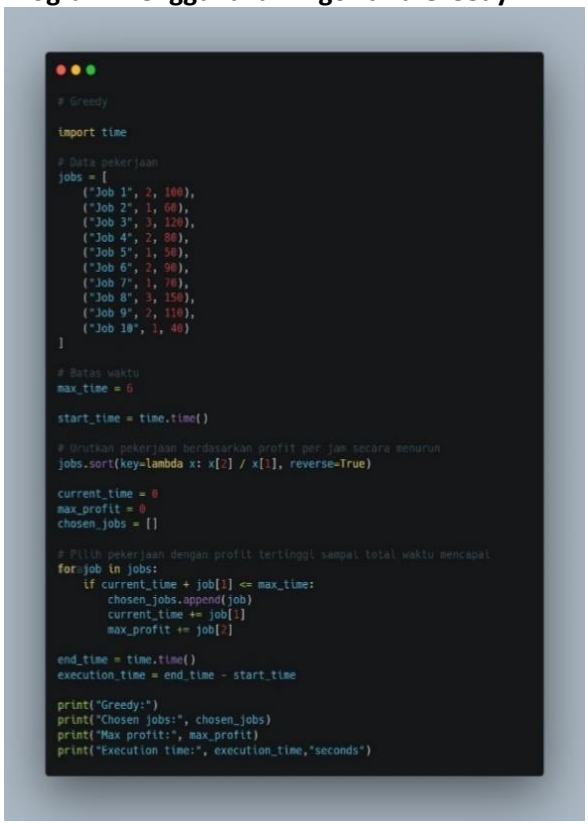
Setelah evaluasi dilakukan, algoritma Brute Force terbukti menjadi pilihan yang sangat efisien dalam hal waktu eksekusi. Brute Force melakukan pencarian secara eksplisit melalui semua kemungkinan solusi tanpa mempertimbangkan strategi optimasi yang canggih. Namun, kekurangannya terletak pada kompleksitasnya yang tinggi karena harus memeriksa setiap kemungkinan solusi secara menyeluruh. Meskipun demikian, Brute Force memastikan bahwa tidak ada solusi yang terlewatkan dalam prosesnya. Di sisi lain, algoritma Greedy menawarkan pendekatan yang berbeda. Greedy cenderung memilih langkah terbaik pada setiap langkahnya, tanpa mempertimbangkan implikasi jangka panjangnya. Meskipun Greedy dapat memberikan solusi yang cepat, tidak dapat menjamin solusi optimal secara global karena keputusan yang dibuat pada setiap langkahnya hanya berdasarkan pada informasi lokal. Dengan mempertimbangkan karakteristik spesifik dari masalah yang dihadapi, pemilihan antara algoritma Brute Force dan Greedy dapat disesuaikan untuk mencapai solusi yang optimal dalam pemrosesan kata. Berikut adalah contoh program dan hasilnya untuk kedua algoritma.

No Pekerjaan	Durasi Pekerjaan (jam)	Profit Pekerjaan
1	2	100
2	1	60
3	3	120
4	2	80
5	1	50
6	2	90
7	1	70
8	3	150
9	2	110
10	1	40

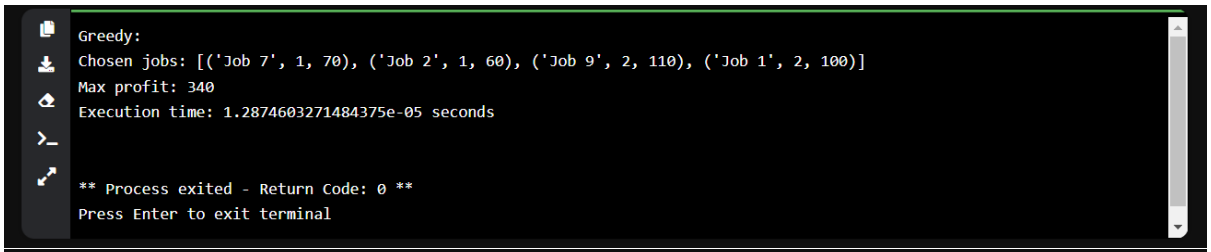
Program menggunakan Algoritma Brute Force

```python
# Brute Force

from itertools import combinations
import time

# Data pekerjaan
jobs = [
    ("Job 1", 2, 100),
    ("Job 2", 1, 60),
    ("Job 3", 3, 120),
    ("Job 4", 2, 80),
    ("Job 5", 1, 50),
    ("Job 6", 2, 90),
    ("Job 7", 1, 70),
    ("Job 8", 3, 150),
    ("Job 9", 2, 110),
    ("Job 10", 1, 40)
]

# Batas waktu
max_time = 6

start_time = time.time()

max_profit = 0
best_combination = None

# Generate all combinations of jobs
for r in range(len(jobs) + 1):
    for combination in combinations(jobs, r):
        total_time = sum(job[1] for job in combination)
        total_profit = sum(job[2] for job in combination)
        if total_time <= max_time and total_profit >
max_profit: max_profit = total_profit
            best_combination = combination

end_time = time.time()
execution_time = end_time - start_time

print("Brute Force:")
print("Best combination:", best_combination)
print("Max profit:", max_profit)
print("Execution time:", execution_time, "seconds")
```

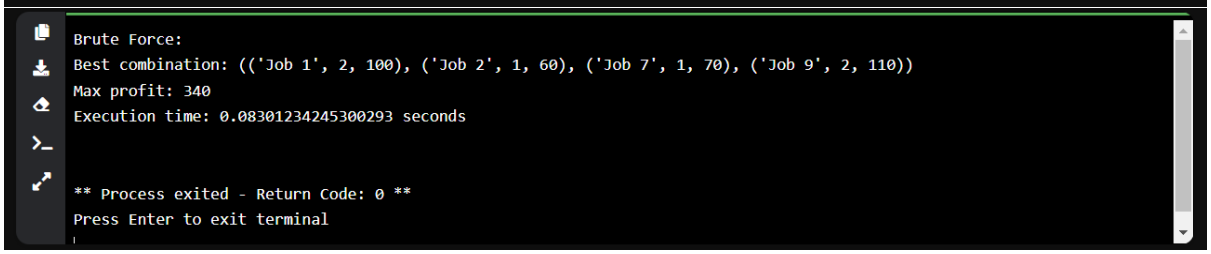
Program menggunakan Algoritma Greedy



```python
# Greedy

import time

# Data pekerjaan
jobs = [
    ("Job 1", 2, 100),
    ("Job 2", 1, 60),
    ("Job 3", 3, 120),
    ("Job 4", 2, 80),
    ("Job 5", 1, 50),
    ("Job 6", 2, 90),
    ("Job 7", 1, 70),
    ("Job 8", 3, 150),
    ("Job 9", 2, 110),
    ("Job 10", 1, 40)
]

# Batas waktu
max_time = 6

start_time = time.time()

# Urutkan pekerjaan berdasarkan profit per jam secara menurun
jobs.sort(key=lambda x: x[2] / x[1], reverse=True)

current_time = 0
max_profit = 0
chosen_jobs = []

# Pilih pekerjaan dengan profit tertinggi sampai total waktu mencapai
for job in jobs:
    if current_time + job[1] <= max_time:
        chosen_jobs.append(job)
        current_time += job[1]
        max_profit += job[2]

end_time = time.time()
execution_time = end_time - start_time

print("Greedy:")
print("Chosen jobs:", chosen_jobs)
print("Max profit:", max_profit)
print("Execution time:", execution_time,"seconds")
```



```
Greedy:
Chosen jobs: [('Job 7', 1, 70), ('Job 2', 1, 60), ('Job 9', 2, 110), ('Job 1', 2, 100)]
Max profit: 340
Execution time: 1.2874603271484375e-05 seconds


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```



```
Brute Force:
Best combination: (('Job 1', 2, 100), ('Job 2', 1, 60), ('Job 7', 1, 70), ('Job 9', 2, 110))
Max profit: 340
Execution time: 0.08301234245300293 seconds


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

5. **Eksperimen**

   Dalam kasus yang dianalisis, kami bekerja dengan data yang di generate secara acak dan berformat Excel, kemudian dikonversi menjadi format CSV untuk memudahkan pemrosesan. Data tersebut terdiri dari 3 kolom utama, yaitu:

- Job Name (Nama Pekerjaan)
- Durasi Pekerjaan (Jam)
- Profit

Data yang kami gunakan untuk analisis terdapat 3 data yang sama yaitu :

- Data Pekerjaan dengan isi total 15 data
- Data Pekerjaan dengan isi total 20 data
- Data Pekerjaan dengan isi total 25 data

Data ini dihasilkan secara acak sehingga tidak memerlukan pemrosesan awal, pemfilteran, atau perlakuan khusus lainnya sebelum digunakan dalam analisis. Dari data – data tersebut kami akan melakukan analisis terhadap algoritma brute force dan greedy untuk melihat perbandingannya dalam hal eksekusi waktu dan ketepatan dalam hasil yang didapatkan.

Untuk melakukan eksperimen tersebut kami menetapkan untuk mencari :

- Total Maximum Profit (Profit tertinggi) dari pekerjaan yang dipilih dari masing – masing algoritma
- Durasi Maximum dari pekerjaan yang dipilih yaitu dengan total 10 jam

5.1 Metode Greedy

A. Uji dengan Data Pekerjaan dengan isi total 15 data

Job Name	Duration	Profit
Job_14	1	950
Job_9	1	949
Job_2	1	935
Job_11	1	877
Job_13	1	855
Job_6	1	700
Job_10	2	945
Job_4	2	823
Total	10	7034
Execution Time (s)		0.00154

B. Uji dengan Data Pekerjaan dengan isi total 20 data

Job Name	Duration	Profit
Job_14	1	950
Job_9	1	949
Job_2	1	935
Job_11	1	877
Job_13	1	855
Job_6	1	700
Job_20	1	643
Job_10	2	945
Total	9	6854
Execution Time (s)		0.002054

C. Uji dengan Data Pekerjaan dengan isi total 25 data

Job Name	Duration	Profit
Job_14	1	950
Job_9	1	949
Job_2	1	935
Job_11	1	877
Job_13	1	855
Job_6	1	700
Job_20	1	643
Job_10	2	945
Total	9	6854
Execution Time (s)		0.002152

5.2 Metode Brute Force

A. Uji dengan Data Pekerjaan denga isi total 15 data

Job Name	Duration	Profit
Job_2	1	935
Job_4	2	823
Job_6	1	700
Job_9	1	949
Job_10	2	945
Job_11	1	877
Job_13	1	855
Job_14	1	950
Total	10	7034
Execution Time (s)		0.025225

B. Uji dengan Data Pekerjaan denga isi total 20 data

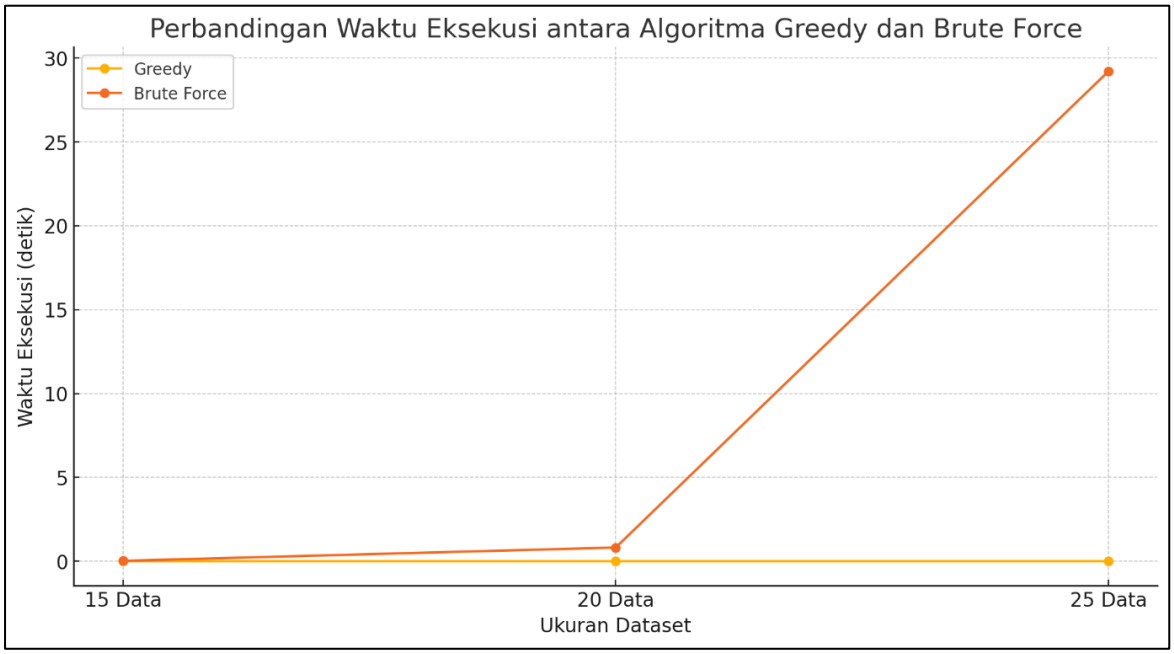
Job Name	Duration	Profit
Job_2	1	935
Job_6	1	700
Job_9	1	949
Job_10	2	945
Job_11	1	877
Job_13	1	855
Job_14	1	950
Job_18	1	365
Job_20	1	643
Total	10	7219
Execution Time (s)		0.818104

C. Uji dengan Data Pekerjaan denga isi total 25 data

Job Name	Duration	Profit
Job_2	1	935
Job_6	1	700
Job_9	1	949
Job_10	2	945
Job_11	1	877
Job_13	1	855
Job_14	1	950
Job_18	1	365
Job_20	1	643
Total	10	7219
Execution Time (s)		29.20425

5.3 Hasil Perbandingan Grafik Algoritma Greedy dan Brute Force

Dataset Size	Algorithm	Total Profit	Execution Time (s)
15 Data	Greedy	7034	0.00154
20 Data	Greedy	6854	0.002054
25 Data	Greedy	6854	0.002152
15 Data	Brute Force	7034	0.025225
20 Data	Brute Force	7034	0.818104
25 Data	Brute Force	7034	29.20425



Berdasarkan eksperimen yang dilakukan menggunakan metode Greedy dan Brute Force dengan variasi kapasitas data, terdapat beberapa temuan menarik.

Untuk metode Greedy, waktu eksekusi meningkat secara konsisten dari kapasitas 15 data hingga 25 data. Ini mengindikasikan bahwa Greedy memiliki konsistensi dalam penanganan peningkatan beban data. Di sisi lain, metode Brute Force menunjukkan peningkatan waktu eksekusi yang lebih signifikan dibandingkan dengan Greedy. Waktu eksekusi metode Brute Force meningkat sangat tinggi seiring dengan bertambahnya kapasitas data dari 15 hingga 25 data. Ini menunjukkan bahwa Brute Force memiliki keterbatasan dalam penanganan dataset yang besar, dan waktu eksekusinya jauh lebih tinggi dibandingkan dengan Greedy, khususnya pada kapasitas data yang lebih besar.

Namun, metode Brute Force menghasilkan solusi yang lebih optimal dibandingkan dengan Greedy. Ini dapat dilihat dari total keuntungan yang diperoleh dari masing-masing metode. Meskipun waktu eksekusi Brute Force lebih tinggi, hasil yang diperoleh lebih optimal, menunjukkan bahwa Brute Force lebih efektif dalam menemukan solusi terbaik.

Dalam eksperimen ini, Greedy menunjukkan waktu eksekusi yang lebih rendah pada kapasitas data 15 , data 20 dan data 25, Greedy tetap lebih cepat dibandingkan dengan Brute Force yang menunjukkan bahwa metode Greedy lebih efisien dalam menangani dataset yang lebih besar jika kecepatan menjadi pertimbangan utama. Metode Greedy mungkin lebih cocok dalam menangani dataset yang besar jika kecepatan eksekusi menjadi pertimbangan utama.

6. Kesimpulan

Berdasarkan analisis performa dari kedua metode ini, Greedy tampak lebih efisien untuk dataset besar dibandingkan dengan Brute Force dari segi waktu eksekusi. Greedy lebih cocok untuk aplikasi yang membutuhkan konsistensi dan prediktabilitas waktu eksekusi dalam skala data yang lebih terbatas. Di sisi lain, Brute Force menghasilkan solusi yang lebih optimal namun membutuhkan waktu eksekusi yang sangat tinggi. Oleh karena itu, pemilihan metode harus mempertimbangkan antara kebutuhan akan waktu eksekusi yang cepat atau hasil yang lebih optimal.

- **Analisis Kompleksitas Waktu dari Algoritma Greedy dan Brute Force :**

1. **Greedy** Kompleksitas waktu untuk keseluruhan kode Greedy adalah $\boldsymbol{O(n \log n)}$, karena operasi pengurutan menjadi faktor paling krusial dalam menentukan efisiensi keseluruhan. Proses ini memiliki kompleksitas waktu $\boldsymbol{O(n \log n)}$, yang menunjukkan bahwa waktu yang diperlukan untuk menjalankan kode meningkat secara logaritmik seiring dengan peningkatan data.

2. **Brute Force** Kompleksitas waktu untuk metode Brute Force secara umum adalah $\boldsymbol{O(n^2)}$, karena metode ini melibatkan pemeriksaan semua kemungkinan kombinasi, yang sangat mempengaruhi kinerja saat dataset semakin besar. Ini sesuai dengan karakteristik umum algoritma Brute Force yang tidak efisien dalam menangani set data besar karena membutuhkan waktu eksekusi yang sangat tinggi.

LAMPIRAN

1. Code Program Metode Greedy

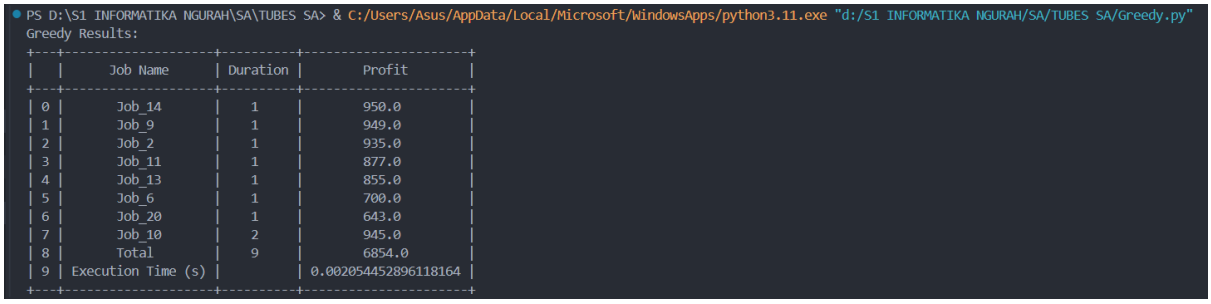
```python
# Greedy

import pandas as pd
import time
from tabulate import tabulate

# Load data pekerjaan dari file CSV
jobs = pd.read_csv('Jobs_Data_25.csv')

def greedy_max_profit(jobs, max_duration):
    start_time = time.time()  # Mulai pengukuran waktu

    # Hitung rasio profit terhadap durasi
    jobs['Profit_to_Duration_Ratio'] = jobs['Profit'] / jobs['Duration']

    # Urutkan pekerjaan berdasarkan rasio dari tinggi ke rendah
    jobs_sorted = jobs.sort_values(by='Profit_to_Duration_Ratio', ascending=False)

    total_duration = 0
    total_profit = 0
    best_combination = []

    # Pilih pekerjaan berdasarkan urutan rasio profit terhadap durasi
    for job in jobs_sorted.itertuples(index=False):
        if total_duration + job.Duration <= max_duration:
            total_duration += job.Duration
            total_profit += job.Profit
            best_combination.append(job)
        else:
            break

    end_time = time.time()  # Akhiri pengukuran waktu
    execution_time = end_time - start_time  # Hitung waktu eksekusi

    return best_combination, total_profit, execution_time

# Eksekusi fungsi dengan durasi maksimum 10 jam
best_combination, best_profit, execution_time = greedy_max_profit(jobs, 10)

# Buat DataFrame untuk kombinasi terbaik
output_data = []
for job in best_combination:
    output_data.append([job[0], job[1], job[2]])

output_df = pd.DataFrame(output_data, columns=['Job Name', 'Duration', 'Profit'])

# Tambahkan total profit dan waktu eksekusi ke DataFrame
output_df.loc[len(output_df)] = ['Total', sum(job.Duration for job in best_combination),
output_df.loc[len(output_df)] = ['Execution Time (s)', '', execution_time]

# Cetak hasil menggunakan tabulate untuk output yang lebih rapi
print("Greedy Results:")
print(tabulate(output_df, headers='keys', tablefmt='pretty'))
```

2. Hasil Eksekusi Program Metode Greedy

- Uji dengan 15 data

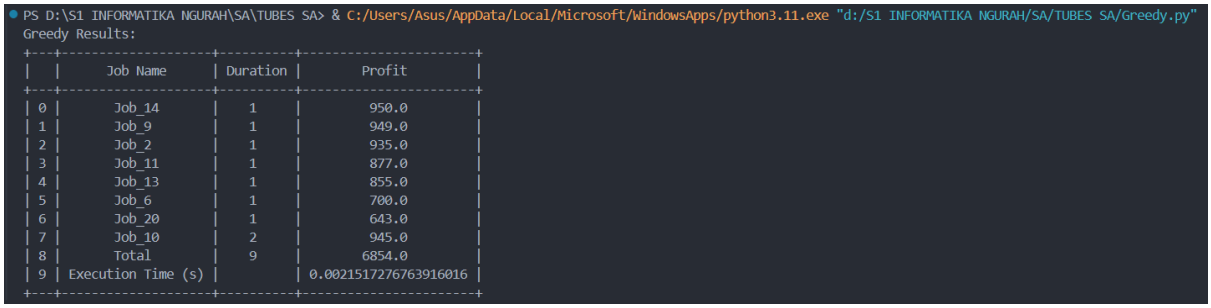
```
PROBLEMS  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE

PS D:\S1 INFORMATIKA NGURAH\SA\TUBES SA> & C:/Users/Asus/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/S1 INFORMATIKA NGURAH/SA/TUBES SA/Greedy.py"
Greedy Results:
+---+--------------------+----------+----------------------+
|   |      Job Name      | Duration |        Profit        |
+---+--------------------+----------+----------------------+
| 0 |       Job_14       |    1     |        950.0         |
| 1 |       Job_9        |    1     |        949.0         |
| 2 |       Job_2        |    1     |        935.0         |
| 3 |       Job_11       |    1     |        877.0         |
| 4 |       Job_13       |    1     |        855.0         |
| 5 |       Job_6        |    1     |        700.0         |
| 6 |       Job_10       |    2     |        945.0         |
| 7 |       Job_4        |    2     |        823.0         |
| 8 |       Total        |    10    |        7034.0        |
| 9 | Execution Time (s) |          | 0.0015397071838378906 |
+---+--------------------+----------+----------------------+
```

- Uji dengan 20 data



```
PS D:\S1 INFORMATIKA NGURAH\SA\TUBES SA> & C:/Users/Asus/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/S1 INFORMATIKA NGURAH/SA/TUBES SA/Greedy.py"
Greedy Results:
+---+--------------------+----------+----------------------+
|   |      Job Name      | Duration |        Profit        |
+---+--------------------+----------+----------------------+
| 0 |       Job_14       |    1     |        950.0         |
| 1 |       Job_9        |    1     |        949.0         |
| 2 |       Job_2        |    1     |        935.0         |
| 3 |       Job_11       |    1     |        877.0         |
| 4 |       Job_13       |    1     |        855.0         |
| 5 |       Job_6        |    1     |        700.0         |
| 6 |       Job_20       |    1     |        643.0         |
| 7 |       Job_10       |    2     |        945.0         |
| 8 |       Total        |    9     |        6854.0        |
| 9 | Execution Time (s) |          | 0.002054452896118164 |
+---+--------------------+----------+----------------------+
```

- Uji dengan 25 data



```
PS D:\S1 INFORMATIKA NGURAH\SA\TUBES SA> & C:/Users/Asus/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/S1 INFORMATIKA NGURAH/SA/TUBES SA/Greedy.py"
Greedy Results:
+---+--------------------+----------+----------------------+
|   |      Job Name      | Duration |        Profit        |
+---+--------------------+----------+----------------------+
| 0 |       Job_14       |    1     |        950.0         |
| 1 |       Job_9        |    1     |        949.0         |
| 2 |       Job_2        |    1     |        935.0         |
| 3 |       Job_11       |    1     |        877.0         |
| 4 |       Job_13       |    1     |        855.0         |
| 5 |       Job_6        |    1     |        700.0         |
| 6 |       Job_20       |    1     |        643.0         |
| 7 |       Job_10       |    2     |        945.0         |
| 8 |       Total        |    9     |        6854.0        |
| 9 | Execution Time (s) |          | 0.0021517276763916016 |
+---+--------------------+----------+----------------------+
```

3. Code Program Metode Brute Force

```
# Brute Force

const pluckDeep = key => obj => key.split('.').reduce((accum, key) => accum[key], obj)
import itertools
import pandas as pd
import time
from tabulate import tabulate

# Load data pekerjaan dari file CSV
jobs = pd.read_csv('Jobs_Data_15.csv')

def brute_force_max_profit(jobs, max_duration):
    start_time = time.time()  # Mulai pengukuran waktu
    best_profit = 0  # Inisialisasi profit terbaik
    best_combination = None  # Inisialisasi kombinasi terbaik

    # Iterasi melalui semua kombinasi pekerjaan untuk setiap ukuran subset
    for r in range(1, len(jobs) + 1):
        for combination in itertools.combinations(jobs.itertuples(index=False), r):
            total_duration = sum(job.Duration for job in combination)  # Hitung total durasi
            if total_duration <= max_duration:  # Periksa apakah total durasi sesuai
                total_profit = sum(job.Profit for job in combination)  # Hitung total profit
                if total_profit > best_profit:  # Jika profit lebih besar dari yang sebelumnya
                    best_profit = total_profit  # Perbarui profit terbaik
                    best_combination = combination  # Perbarui kombinasi terbaik

    end_time = time.time()  # Akhiri pengukuran waktu
    execution_time = end_time - start_time  # Hitung waktu eksekusi

    return best_combination, best_profit, execution_time

# Eksekusi fungsi dengan durasi maksimum 10 jam
best_combination, best_profit, execution_time = brute_force_max_profit(jobs, 10)

# Buat DataFrame untuk kombinasi terbaik
output_data = []
for job in best_combination:
    output_data.append([job[0], job[1], job[2]])

output_df = pd.DataFrame(output_data, columns=['Job Name', 'Duration', 'Profit'])

# Tambahkan total profit dan waktu eksekusi ke DataFrame
output_df.loc[len(output_df)] = ['Total', sum(job.Duration for job in best_combination),
best_profit]
output_df.loc[len(output_df)] = ['Execution Time (s)', '', execution_time]

# Cetak hasil menggunakan tabulate untuk output yang lebih rapi
print("Brute Force Results:")
print(tabulate(output_df, headers='keys', tablefmt='pretty'))

const compose = (...fns) => res => fns.reduce((accum, next) => next(accum), res)

const unfold = (f, seed) => {
  const go = (f, seed, acc) => {
    const res = f(seed)
    return res ? go(f, res[1], acc.concat([res[0]])) : acc
  }
  return go(f, seed, [])
}
```

4. Hasil Eksekusi Program Metode Brute Force

- Uji dengan 15 data

```
PS D:\S1 INFORMATIKA NGURAH\SA\TUBES SA> & C:/Users/Asus/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/S1 INFORMATIKA NGURAH/SA/TUBES SA/BruteForce.py"

Brute Force Results:
+---+--------------------+----------+----------------------+
|   |      Job Name      | Duration |        Profit        |
+---+--------------------+----------+----------------------+
| 0 |       Job_2        |    1     |        935.0         |
| 1 |       Job_4        |    2     |        823.0         |
| 2 |       Job_6        |    1     |        700.0         |
| 3 |       Job_9        |    1     |        949.0         |
| 4 |       Job_10       |    2     |        945.0         |
| 5 |       Job_11       |    1     |        877.0         |
| 6 |       Job_13       |    1     |        855.0         |
| 7 |       Job_14       |    1     |        950.0         |
| 8 |       Total        |    10    |        7034.0        |
| 9 | Execution Time (s) |          | 0.025224924087524414 |
+---+--------------------+----------+----------------------+
```

- Uji dengan 20 data

```
PS D:\S1 INFORMATIKA NGURAH\SA\TUBES SA> & C:/Users/Asus/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/S1 INFORMATIKA NGURAH/SA/TUBES SA/BruteForce.py"

Brute Force Results:
+----+--------------------+----------+--------------------+
|    |      Job Name      | Duration |       Profit       |
+----+--------------------+----------+--------------------+
| 0  |       Job_2        |    1     |       935.0        |
| 1  |       Job_6        |    1     |       700.0        |
| 2  |       Job_9        |    1     |       949.0        |
| 3  |       Job_10       |    2     |       945.0        |
| 4  |       Job_11       |    1     |       877.0        |
| 5  |       Job_13       |    1     |       855.0        |
| 6  |       Job_14       |    1     |       950.0        |
| 7  |       Job_18       |    1     |       365.0        |
| 8  |       Job_20       |    1     |       643.0        |
| 9  |       Total        |    10    |       7219.0       |
| 10 | Execution Time (s) |          | 0.8181042671203613 |
+----+--------------------+----------+--------------------+
```

- Uji dengan 25 data

```
PS D:\S1 INFORMATIKA NGURAH\SA\TUBES SA> & C:/Users/Asus/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/S1 INFORMATIKA NGURAH/SA/TUBES SA/BruteForce.py"

Brute Force Results:
+----+--------------------+----------+-------------------+
|    |      Job Name      | Duration |      Profit       |
+----+--------------------+----------+-------------------+
| 0  |       Job_2        |    1     |       935.0       |
| 1  |       Job_6        |    1     |       700.0       |
| 2  |       Job_9        |    1     |       949.0       |
| 3  |       Job_10       |    2     |       945.0       |
| 4  |       Job_11       |    1     |       877.0       |
| 5  |       Job_13       |    1     |       855.0       |
| 6  |       Job_14       |    1     |       950.0       |
| 7  |       Job_18       |    1     |       365.0       |
| 8  |       Job_20       |    1     |       643.0       |
| 9  |       Total        |    10    |      7219.0       |
| 10 | Execution Time (s) |          | 29.20425033569336 |
+----+--------------------+----------+-------------------+
```