

**TUGAS AKHIR**

**OPTIMASI JADWAL PRIBADI BERDASARKAN PRIORITAS DAN  
DURASI MENGGUNAKAN ALGORITMA BACKTRACKING DENGAN  
STRATEGI GREEDY DAN PRUNING**



**Oleh:**

**Muhammad Fadly Ukhrowi Akhfir**

**2125250044**

**Program Studi  
Teknik Informatika  
Fakultas Ilmu Komputer dan Rekayasa  
Universitas Multi Data Palembang  
2025**

## Daftar Isi

1.Latar Belakang.....	2
2.Tujuan.....	4
3.Studi Literatur / Tinjauan Pustaka.....	5
3.1 Penjadwalan ( <i>Scheduling</i> ) .....	5
3.2 Optimasi Penjadwalan.....	5
3.3 Algoritma <i>Backtracking</i> .....	6
3.4 Strategi <i>Greedy</i> .....	6
3.5 Teknik <i>Pruning</i> .....	6
4.Metodologi .....	7
4.1 Data.....	7
4.1.1 Data Real-time (Input Pengguna melalui GUI) .....	7
4.1.2 Data Uji (Testing Data) .....	7
4.2 Algoritma .....	8
4.2.1 Algoritma <i>Backtracking</i> .....	8
4.2.2 Strategi <i>Greedy Sorting</i> .....	9
4.2.3 Teknik <i>Pruning</i> .....	10
4.3 Rancangan Program .....	11
4.3.1 Arsitektur Program.....	11
4.3.2 Flowchart.....	11
4.3.3 Diagram Use Case.....	11
4.3.4 Kode Program.....	12
5.Implementasi .....	18
5.1 Input dan Output Program .....	18
5.2 Tampilan Program (GUI) .....	19
5.3 Penjelasan Cara Penggunaan .....	21
6.Pengujian .....	26
6.1 Pengujian Fungsional .....	26
6.2 Pengujian Performa (Performance Testing) .....	26
7.Analisis .....	28
8.Kesimpulan dan Saran .....	29
8.1 Kesimpulan.....	29
8.2 Saran .....	30

# 1. Latar Belakang

Manajemen waktu yang efektif merupakan aspek krusial dalam kehidupan sehari-hari, terutama bagi individu yang memiliki berbagai aktivitas dengan tingkat prioritas dan durasi yang berbeda-beda. Ketidakmampuan dalam menyusun jadwal yang optimal dapat mengakibatkan penurunan produktivitas dan peningkatan stres. Oleh karena itu, diperlukan sistem penjadwalan yang mampu mengakomodasi preferensi individu berdasarkan prioritas dan durasi aktivitas.

Algoritma *Backtracking* merupakan salah satu metode yang efektif dalam menyelesaikan masalah penjadwalan, terutama yang berkaitan dengan *constraint satisfaction problems*. Dengan menelusuri semua kemungkinan solusi dan melakukan *backtrack* ketika menemui kondisi yang tidak memenuhi syarat, algoritma ini dapat menemukan solusi optimal dalam ruang pencarian yang besar (Winata, 2023).

Namun, penerapan *backtracking* secara murni sering kali menghadapi masalah efisiensi karena eksplorasi ruang solusi yang luas. Untuk mengatasi hal tersebut, strategi *Greedy* dapat diterapkan dengan mengurutkan aktivitas berdasarkan rasio prioritas terhadap durasi, sehingga aktivitas dengan nilai tertinggi diprioritaskan. Strategi ini membantu dalam mempercepat proses pencarian solusi dengan memilih opsi terbaik pada setiap langkah (Winata, 2023).

Selain itu, teknik *Pruning* digunakan untuk mengeliminasi jalur pencarian yang tidak mungkin menghasilkan solusi optimal. Dengan memotong cabang-cabang yang tidak memenuhi syarat sejak awal, algoritma dapat menghemat waktu dan sumber daya komputasi.

Kombinasi dari algoritma *backtracking* dengan strategi *greedy* dan *pruning* telah terbukti efektif dalam berbagai studi. Misalnya, dalam penelitian oleh Winata (2023), pendekatan ini digunakan untuk menyelesaikan masalah penjadwalan kegiatan dengan mempertimbangkan prioritas dan durasi aktivitas. Demikian pula, dalam penelitian oleh

Siregar dan Siregar (2023), algoritma *greedy* diterapkan untuk penjadwalan pekerjaan dalam manajemen waktu dan sumber daya.

Dalam konteks penjadwalan pribadi, pendekatan ini dapat diimplementasikan melalui antarmuka pengguna grafis (GUI) yang memungkinkan individu untuk memasukkan data aktivitas mereka dan menerima jadwal yang dioptimalkan secara *real-time*. Hal ini tidak hanya meningkatkan efisiensi waktu tetapi juga memberikan fleksibilitas bagi pengguna dalam mengatur kegiatan sehari-hari mereka.

Dengan demikian, pengembangan sistem penjadwalan pribadi yang mengintegrasikan algoritma *backtracking* dengan strategi *greedy* dan *pruning* menjadi solusi yang menjanjikan untuk meningkatkan manajemen waktu individu. Proyek ini bertujuan untuk merancang dan mengimplementasikan sistem tersebut, serta mengevaluasi kinerjanya dalam berbagai skenario penggunaan.

## 2. Tujuan

Tujuan dari proyek ini adalah untuk merancang dan mengimplementasikan sistem penjadwalan kegiatan berbasis preferensi pengguna yang mampu menghasilkan jadwal optimal berdasarkan dua parameter utama: prioritas dan durasi aktivitas. Sistem ini akan memanfaatkan algoritma *Backtracking* yang telah di optimasi melalui pendekatan *Greedy Sorting* dan *Pruning* untuk meningkatkan efisiensi pencarian solusi. Secara lebih rinci, tujuan dari proyek ini adalah sebagai berikut:

1. Mengembangkan model penjadwalan pribadi yang dapat mengakomodasi beragam aktivitas berdasarkan masukan pengguna berupa nama aktivitas, durasi, dan tingkat prioritas (skala 1–10).
2. Menerapkan algoritma *Backtracking* sebagai metode utama dalam menyusun kombinasi aktivitas terbaik yang tidak melebihi total waktu yang tersedia dalam satu hari (misalnya 8–12 jam).

3. Mengintegrasikan strategi *Greedy* untuk mengurutkan aktivitas berdasarkan rasio prioritas terhadap durasi (*priority/duration*) guna memaksimalkan efisiensi eksplorasi solusi.
4. Mengimplementasikan teknik *Pruning* untuk mempercepat proses pencarian dengan menghindari eksplorasi solusi yang tidak mungkin memenuhi syarat optimalisasi.
5. Menguji kinerja sistem dalam menghasilkan solusi yang optimal secara prioritas dan efisien secara waktu, serta melakukan evaluasi terhadap hasil penjadwalan dalam berbagai skenario pengguna.

Dengan tercapainya tujuan-tujuan tersebut, proyek ini diharapkan dapat menjadi solusi efektif dalam membantu individu mengatur waktu mereka secara optimal berdasarkan kebutuhan dan preferensi masing-masing.

### **3. Studi Literatur / Tinjauan Pustaka**

#### **3.1 Penjadwalan (*Scheduling*)**

Penjadwalan merupakan proses penentuan urutan dan waktu pelaksanaan suatu rangkaian aktivitas agar dapat memenuhi kriteria tertentu seperti efisiensi waktu atau utilitas sumber daya. Dalam konteks individu, penjadwalan pribadi bertujuan mengalokasikan waktu secara optimal untuk berbagai kegiatan berdasarkan durasi dan tingkat kepentingannya. Menurut Simanjuntak et al. (2025), perancangan sistem manajemen waktu dapat membantu meningkatkan produktivitas serta mengurangi waktu yang terbuang karena aktivitas yang tumpang tindih.

#### **3.2 Optimasi Penjadwalan**

Optimasi dalam penjadwalan bertujuan menemukan kombinasi aktivitas terbaik yang memenuhi batasan waktu serta menghasilkan nilai total kepentingan (prioritas) tertinggi. Metode optimasi seperti *greedy* dan *backtracking* telah banyak digunakan dalam berbagai permasalahan, termasuk penjadwalan kerja, alokasi sumber daya, dan tugas-tugas berbasis

preferensi. Ardani et al. (2024) membahas implementasi algoritma *greedy* dan *dynamic programming* untuk penjadwalan interval dengan model *knapsack*, yang esensial dalam optimasi.

### 3.3 Algoritma *Backtracking*

*Backtracking* adalah metode eksplorasi solusi secara rekursif yang menguji semua kemungkinan solusi dengan cara menyusun solusi sebagian dan membatalkan (*backtrack*) jika tidak memenuhi kriteria. Metode ini cocok digunakan dalam kasus di mana kombinasi dari pilihan-pilihan harus diuji satu per satu, seperti dalam penjadwalan atau pemilihan sub set. Menurut Noovriyanto (2023), *backtracking* memberikan hasil optimal meskipun memiliki kompleksitas yang tinggi, sehingga perlu dioptimalkan.

### 3.4 Strategi *Greedy*

*Greedy* adalah strategi penyelesaian masalah dengan memilih solusi terbaik secara lokal pada setiap langkahnya, dengan harapan menghasilkan solusi global yang baik. Dalam konteks penjadwalan, pendekatan ini berguna untuk menyusun urutan aktivitas berdasarkan nilai rasio (prioritas/durasi) secara menurun, agar aktivitas dengan nilai efisien tinggi diproses lebih awal. Ardani et al. (2024) juga menekankan bahwa algoritma *greedy* dapat digunakan untuk penjadwalan interval dengan model *knapsack*.

### 3.5 Teknik *Pruning*

*Pruning* digunakan untuk memangkas jalur eksplorasi yang tidak perlu dalam algoritma pencarian solusi seperti *backtracking*. Dengan teknik ini, sistem akan menghentikan pencarian jika waktu tersisa sudah tidak memungkinkan, atau ketika nilai optimal tidak bisa dicapai lagi dari cabang tersebut. Strategi ini penting agar sistem dapat bekerja lebih cepat dan efisien. Menurut Trisanto (2019), optimasi sistem informasi penjadwalan kuliah dapat dilakukan dengan mengombinasikan *heuristic search* dan teknik *smart backtracking* serta *look ahead*.

## 4. Metodologi

### 4.1 Data

Data yang digunakan dalam penelitian ini terbagi menjadi dua kategori utama, yaitu data masukan secara real-time melalui antarmuka aplikasi (GUI) dan data uji (testing data) yang digunakan untuk pengujian kinerja algoritma.

#### 4.1.1 Data Real-time (Input Pengguna melalui GUI)

Data utama dalam sistem ini berasal dari input pengguna yang dimasukkan secara langsung melalui antarmuka aplikasi berbasis GUI. Pengguna dapat menambahkan satu atau beberapa aktivitas yang akan dijadwalkan, dengan mengisi tiga parameter penting, yaitu:

1. Nama Aktivitas: deskripsi singkat mengenai aktivitas yang akan dilakukan.
2. Durasi Aktivitas: estimasi waktu yang diperlukan untuk menyelesaikan aktivitas, dinyatakan dalam satuan menit.
3. Prioritas Aktivitas: tingkat kepentingan aktivitas yang dinyatakan dalam skala numerik 1 sampai 10, di mana nilai 10 menunjukkan prioritas tertinggi.

Data ini bersifat dinamis dan fleksibel, sesuai preferensi dan kebutuhan setiap pengguna. Setelah semua data dimasukkan, sistem akan secara otomatis menjalankan proses optimasi untuk menghasilkan jadwal yang efisien berdasarkan algoritma yang telah dirancang.

#### 4.1.2 Data Uji (Testing Data)

Selain data masukan dari pengguna, penelitian ini juga menggunakan data uji (testing data) untuk mengevaluasi kinerja dan akurasi algoritma *Backtracking* yang diterapkan. Data uji ini bersifat sintetis dan dibuat secara otomatis menggunakan skrip program *Python*. Tujuan penggunaan data uji adalah sebagai berikut:

1. Menguji performa algoritma pada berbagai ukuran dataset.
2. Mengevaluasi waktu komputasi dalam kondisi beban tinggi (misalnya ratusan hingga ribuan aktivitas).

3. Menganalisis stabilitas dan konsistensi hasil jadwal dengan skenario prioritas dan durasi yang bervariasi.

Format penyimpanan data uji menggunakan JSON (*JavaScript Object Notation*) karena mudah dibaca dan diolah oleh sistem *Python*. Data uji ini akan disimpan dalam berkas *data\_uji.json* dan dibaca secara otomatis saat proses pengujian dimulai.

## 4.2 Algoritma

Penelitian ini menggunakan algoritma *Backtracking* sebagai pendekatan utama untuk menyelesaikan permasalahan penjadwalan pribadi berdasarkan durasi dan prioritas. Untuk meningkatkan efisiensi proses pencarian solusi, algoritma ini dikombinasikan dengan dua strategi optimasi, yaitu *Greedy Sorting* dan *Pruning*.

### 4.2.1 Algoritma *Backtracking*

*Backtracking* merupakan metode *rekursif* yang digunakan untuk mengeksplorasi semua kemungkinan solusi dari suatu permasalahan kombinatorial. Pada sistem ini, *backtracking* akan mencoba menyusun berbagai kombinasi aktivitas yang memungkinkan, lalu memilih kombinasi dengan skor prioritas tertinggi yang tidak melebihi batas total waktu harian pengguna. Proses *backtracking* dilakukan dengan cara:

Menyusun aktivitas satu per satu dalam kombinasi sementara.

1. Menghitung akumulasi durasi dan skor prioritas dari kombinasi tersebut.
2. Melakukan (mundur) jika durasi melebihi waktu yang tersedia.
3. Menyimpan solusi terbaik sejauh ini berdasarkan skor tertinggi.



```

def backtrack(index, waktu_tersisa, jadwal_sementara, skor_sementara, skor_sisa):
    ...
    for i in range(index, len(aktivitas_urut)):
        akt = aktivitas_urut[i]

        if akt.durasi > waktu_tersisa:
            continue

        jadwal_sementara.append(akt)
        backtrack(
            i + 1,
            waktu_tersisa - akt.durasi,
            jadwal_sementara,
            skor_sementara + akt.prioritas,
            skor_sisa - akt.prioritas
        )
        jadwal_sementara.pop()

```

Gambar 4.1 Implementasi Algoritma *Backtracking*

#### 4.2.2 Strategi *Greedy Sorting*

Untuk mengurangi eksplorasi kombinasi yang tidak efisien, aktivitas terlebih dahulu diurutkan menggunakan strategi *Greedy*, yaitu berdasarkan rasio prioritas/durasi. Strategi ini mengutamakan aktivitas yang memberikan nilai prioritas tinggi dengan waktu yang relatif pendek.

```

def jadwal_optimal(daftar_aktivitas, total_waktu):
    aktivitas_urut = sorted(
        daftar_aktivitas,
        key=lambda x: x.prioritas / x.durasi,
        reverse=True
    )

```

Gambar 4.2 Strategi *Greedy Sorting*

Dengan mengurutkan aktivitas terlebih dahulu secara menurun berdasarkan rasio efisiensi tersebut, algoritma memiliki kemungkinan lebih tinggi untuk menemukan solusi optimal di awal proses, sehingga mengurangi beban komputasi secara keseluruhan.

### 4.2.3 Teknik Pruning

*Pruning* diterapkan dalam sistem ini untuk mempercepat proses pencarian solusi optimal dengan menghentikan eksplorasi jalur yang tidak menjanjikan. Berdasarkan implementasi kode, terdapat dua strategi utama yang digunakan:

1. **Pruning berdasarkan estimasi skor maksimum** : Jika skor sementara ditambah sisa skor yang mungkin sudah tidak mampu melebihi skor terbaik yang telah ditemukan ( $\text{skor\_sementara} + \text{skor\_sis} \leq \text{hasil\_terbaik}[\text{'skor'}]$ ), maka jalur tersebut langsung dipangkas karena tidak potensial.
2. **Pruning berdasarkan ambang kepuasan** : Jika skor terbaik saat ini telah mencapai atau melampaui 95% dari skor total teoritis ( $\text{hasil\_terbaik}[\text{'skor'}] \geq \text{ambang\_kepuasan}$ ), maka proses pencarian dihentikan karena dianggap sudah cukup baik.

Kedua pendekatan ini menjadikan algoritma lebih efisien dalam mengeksplorasi solusi tanpa mengorbankan kualitas hasil secara signifikan.

```
if skor_sementara + skor_sisa <= hasil_terbaik['skor']:
    return

if hasil_terbaik['skor'] >= ambang_kepuasan:
    return
```

Gambar 4.3 Implementasi Teknik *Pruning*

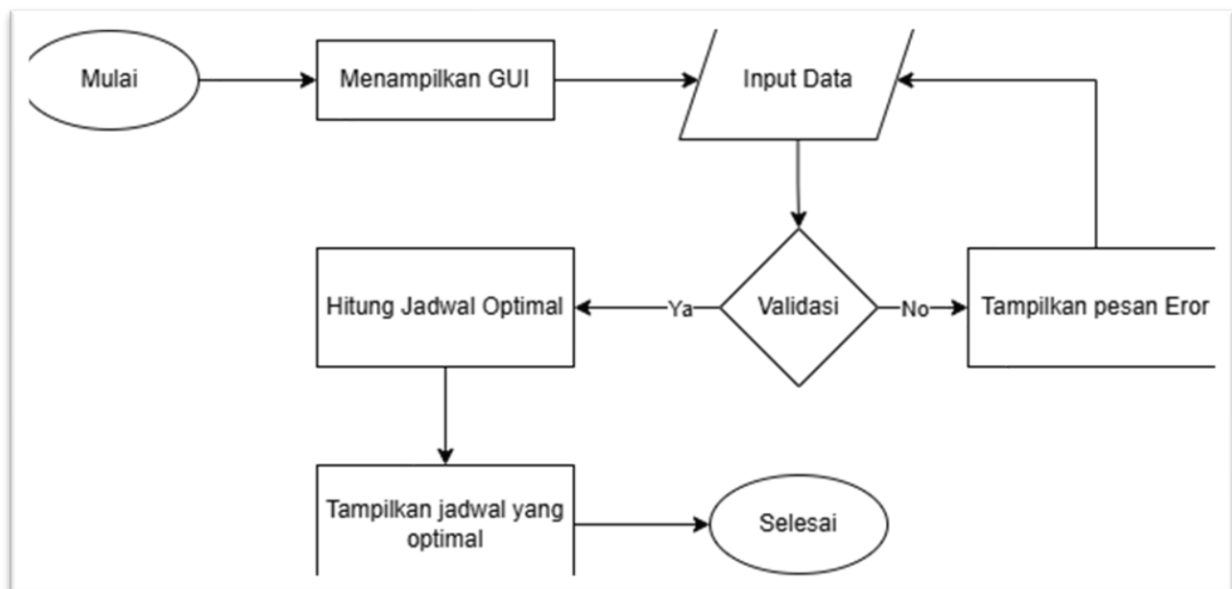
## 4.3 Rancangan Program

### 4.3.1 Arsitektur Program

Arsitektur program pada aplikasi Penjadwalan Aktivitas Harian ini dirancang menggunakan pendekatan modular dan terstruktur agar mudah dikembangkan, diuji, serta dipelihara. Program terdiri dari dua komponen utama, yaitu antarmuka pengguna (GUI) dan logika penjadwalan yang diimplementasikan menggunakan algoritma *backtracking* yang dikombinasikan dengan strategi *greedy* dan *pruning*.

### 4.3.2 Flowchart

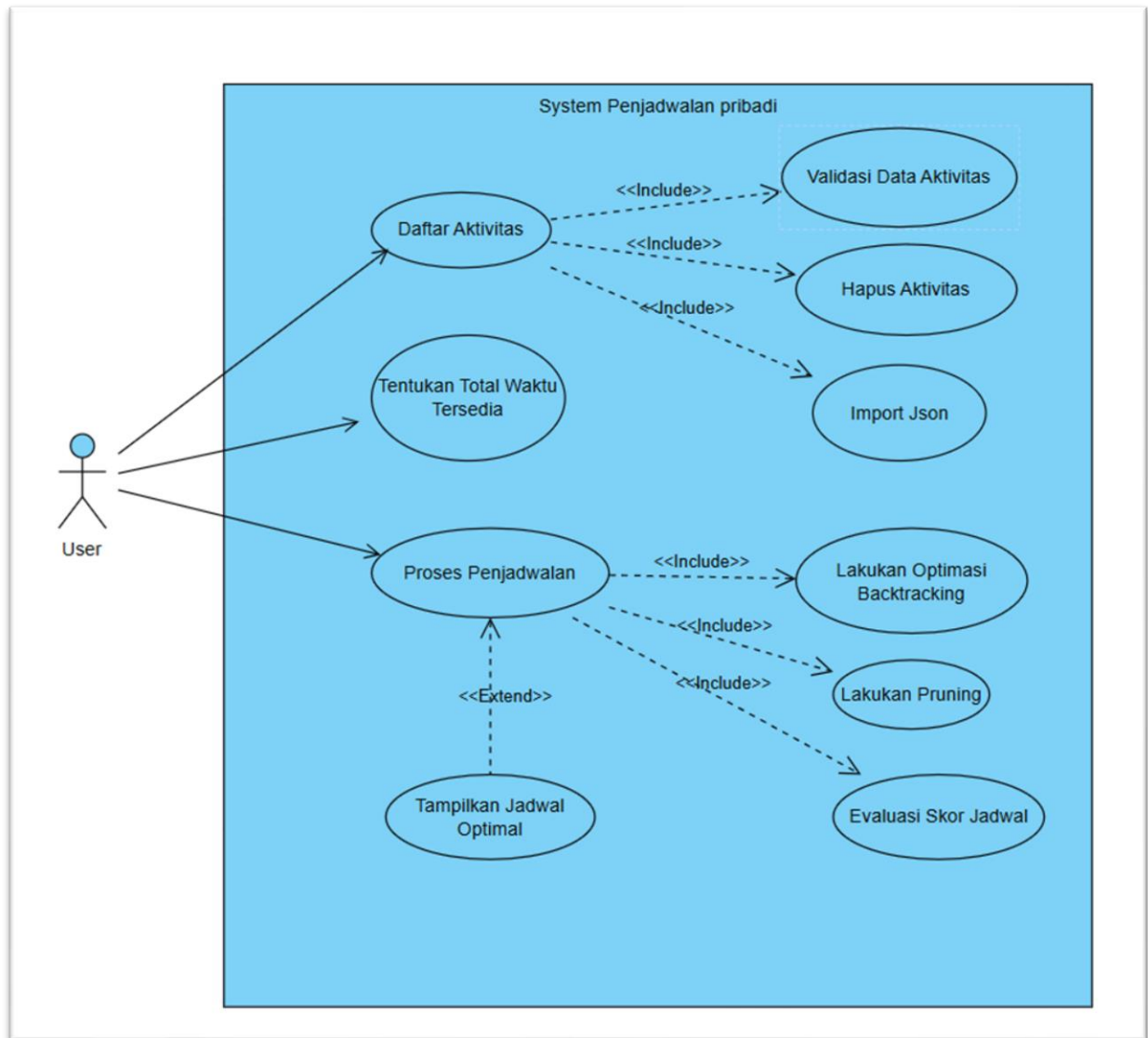
Flowchart berikut menggambarkan alur sistem dari awal pengguna membuka aplikasi hingga mendapatkan hasil penjadwalan:



Gambar 4.4 *Flowchart* perancangan program

### 4.3.3 Diagram Use Case

Diagram berikut menggambarkan interaksi antara pengguna (aktor) dengan sistem dalam bentuk fungsionalitas-fungsionalitas utama yang tersedia:



Gambar 4.5 Use Case Diagram perancangan program

#### 4.3.4 Kode Program

Kode pada Gambar 4.6 berikut menampilkan implementasi dari kombinasi algoritma yang digunakan untuk menyusun jadwal aktivitas pribadi secara optimal. Algoritma ini menggabungkan pendekatan *backtracking* dengan strategi *greedy* dan *pruning* untuk mengeksplorasi kemungkinan jadwal secara efisien. Setiap aktivitas akan dipilih berdasarkan rasio prioritas terhadap durasi, dan hanya kombinasi terbaik yang disimpan sebagai solusi. Proses pencarian solusi jadwal disusun dengan memilih kombinasi aktivitas yang memberikan total skor prioritas tertinggi dalam batas waktu yang tersedia.

```

class Aktivitas:
    def __init__(self, nama, durasi, prioritas):
        self.nama = nama
        self.durasi = durasi
        self.prioritas = prioritas

    def __repr__(self):
        return f"{self.nama} ({self.durasi}m, P{self.prioritas})"

def jadwal_optimal(daftar_aktivitas, total_waktu):
    aktivitas_urut = sorted(
        daftar_aktivitas,
        key=lambda x: x.prioritas / x.durasi,
        reverse=True
    )

    hasil_terbaik = {
        'jadwal': [],
        'skor': 0
    }

    max_skor_teoretis = sum(a.prioritas for a in aktivitas_urut)
    ambang_kepuasan = int(max_skor_teoretis * 0.95)

    def backtrack(index, waktu_tersisa, jadwal_sementara, skor_sementara, skor_sisa):
        nonlocal hasil_terbaik
        if skor_sementara + skor_sisa <= hasil_terbaik['skor']:
            return

        if hasil_terbaik['skor'] >= ambang_kepuasan:
            return

        if skor_sementara > hasil_terbaik['skor']:
            hasil_terbaik['jadwal'] = jadwal_sementara[:]
            hasil_terbaik['skor'] = skor_sementara

        for i in range(index, len(aktivitas_urut)):
            akt = aktivitas_urut[i]

            if akt.durasi > waktu_tersisa:
                continue

            jadwal_sementara.append(akt)
            backtrack(
                i + 1,
                waktu_tersisa - akt.durasi,
                jadwal_sementara,
                skor_sementara + akt.prioritas,
                skor_sisa - akt.prioritas
            )
            jadwal_sementara.pop()

    total_prioritas = sum(a.prioritas for a in aktivitas_urut)
    backtrack(0, total_waktu, [], 0, total_prioritas)

    return hasil_terbaik

```

Gambar 4.6 Kode Implementasi Algoritma pada jadwal.py

Kode pada Gambar 4.7 menampilkan antarmuka awal aplikasi penjadwalan aktivitas harian, yang menunjukkan berbagai fitur seperti *input* aktivitas, impor data dari *file JSON*, dan

pemrosesan jadwal optimal. Antarmuka grafis dikembangkan menggunakan pustaka *PyQt5* untuk memberikan tampilan yang interaktif dan mudah digunakan.

```
class App(QWidget):
    def __init__(self):
        super().__init__()

        self.setWindowTitle('Penjadwalan Aktivitas Harian')
        self.setGeometry(100, 100, 600, 400)

        layout = QVBoxLayout()

        self.input_nama = QLineEdit(self)
        self.input_nama.setPlaceholderText("Nama Aktivitas")
        layout.addWidget(self.input_nama)

        self.input_durasi = QLineEdit(self)
        self.input_durasi.setPlaceholderText("Durasi (menit)")
        layout.addWidget(self.input_durasi)

        self.input_prioritas = QLineEdit(self)
        self.input_prioritas.setPlaceholderText("Prioritas (1-10)")
        layout.addWidget(self.input_prioritas)

        self.tombol_tambah = QPushButton("Tambah Aktivitas", self)
        self.tombol_tambah.clicked.connect(self.tambah_aktivitas)
        layout.addWidget(self.tombol_tambah)

        self.tombol_import = QPushButton("Import Aktivitas", self)
        self.tombol_import.clicked.connect(self.import_dari_json)
        layout.addWidget(self.tombol_import)

        self.tombol_clear = QPushButton("Clear Aktivitas", self)
        self.tombol_clear.clicked.connect(self.clear_aktivitas)
        layout.addWidget(self.tombol_clear)

        self.table_aktivitas = QTableWidgetItem(self)
        self.table_aktivitas.setColumnCount(3)
        self.table_aktivitas.setHorizontalHeaderLabels(['Nama', 'Durasi', 'Prioritas'])
        layout.addWidget(self.table_aktivitas)

        self.input_waktu = QLineEdit(self)
        self.input_waktu.setPlaceholderText("Total Waktu (menit)")
        layout.addWidget(self.input_waktu)

        self.tombol_proses = QPushButton("Proses Jadwal Optimal", self)
        self.tombol_proses.clicked.connect(self.proses_jadwal)
        layout.addWidget(self.tombol_proses)

        self.table_jadwal = QTableWidgetItem(self)
        self.table_jadwal.setColumnCount(3)
        self.table_jadwal.setHorizontalHeaderLabels(['Nama', 'Durasi', 'Prioritas'])
        layout.addWidget(self.table_jadwal)

        self.label_skor = QLabel("Total Skor: 0", self)
        layout.addWidget(self.label_skor)

        self.timeout_timer = QTimer()
        self.timeout_timer.setSingleShot(True)
        self.timeout_timer.timeout.connect(self.handle_timeout)
        self.timeout_duration = 3600000

        self.setLayout(layout)
        self.daftar_aktivitas = []
```

Gambar 4.7 Gambar Kode Program Tampilan Awal

Potongan kode pada Gambar 4.8 berfungsi untuk menampilkan hasil dari proses penjadwalan optimal. Komponen `table_jadwal` digunakan untuk menampilkan daftar aktivitas yang terpilih, sedangkan `label_skor` menampilkan total skor dari aktivitas-aktivitas tersebut

beserta waktu eksekusi algoritma. Bagian ini membantu pengguna dalam mengevaluasi efisiensi dan kualitas jadwal yang dihasilkan oleh sistem.

```
self.table_jadwal = QTableWidgetItem(self)
self.table_jadwal.setColumnCount(3)
self.table_jadwal.setHorizontalHeaderLabels(['Nama', 'Durasi', 'Prioritas'])
layout.addWidget(self.table_jadwal)

self.label_skor = QLabel("Total Skor: 0", self)
layout.addWidget(self.label_skor)

self.setLayout(layout)
self.daftar_aktivitas = []
```

Gambar 4.8 Kode Program Visualisasi Hasil

Gambar 4.9 menunjukkan proses pemrosesan data input dari pengguna ketika tombol "Tambah Aktivitas" ditekan. Fungsi tambah\_aktivitas melakukan validasi terhadap input, memastikan bahwa nama aktivitas tidak kosong, durasi merupakan angka positif, dan prioritas berada dalam rentang 1 hingga 10. Jika seluruh input valid, aktivitas akan disimpan ke dalam daftar internal dan ditampilkan pada tabel aktivitas.

```

def show_error(self, pesan):
    QMessageBox.warning(self, "Input Tidak Valid", pesan)

def tambah_aktivitas(self):
    nama = self.input_nama.text().strip()
    if not nama:
        self.show_error("Nama aktivitas tidak boleh kosong.")
        return

    try:
        durasi = int(self.input_durasi.text())
        prioritas = int(self.input_prioritas.text())
        if durasi <= 0:
            raise ValueError("Durasi harus lebih dari 0.")
        if not (1 <= prioritas <= 10):
            raise ValueError("Prioritas harus antara 1 dan 10.")
    except ValueError:
        self.show_error("Durasi dan Prioritas harus berupa angka valid (durasi > 0, prioritas >= 1 dan <= 10).")
        return

    aktivitas = Aktivitas(nama, durasi, prioritas)
    self.daftar_aktivitas.append(aktivitas)

    row_position = self.table_aktivitas.rowCount()
    self.table_aktivitas.insertRow(row_position)
    self.table_aktivitas.setItem(row_position, 0, QTableWidgetItem(nama))
    self.table_aktivitas.setItem(row_position, 1, QTableWidgetItem(str(durasi)))
    self.table_aktivitas.setItem(row_position, 2, QTableWidgetItem(str(prioritas)))

    self.input_nama.clear()
    self.input_durasi.clear()
    self.input_prioritas.clear()

```

Gambar 4.9 Kode Program Fungsi Validasi & Penambahan Aktivitas

Gambar 4.10 menunjukkan proses fitur untuk mengimpor data aktivitas dari file berformat JSON. File yang valid akan dibaca dan aktivitasnya ditampilkan dalam tabel input. Ini mempermudah pengguna yang ingin langsung menguji jadwal dari data yang sudah tersedia tanpa harus mengetik manual.



```

def import_dari_json(self):
    file_path, _ = QFileDialog.getOpenFileName(self, "Pilih File JSON", "", "JSON Files (*.json)")
    if not file_path:
        return

    try:
        with open(file_path, 'r') as file:
            data = json.load(file)
            for item in data:
                nama = item.get("nama", "").strip()
                durasi = item.get("durasi")
                prioritas = item.get("prioritas")

                if not nama or not isinstance(durasi, int) or not isinstance(prioritas, int):
                    continue
                if durasi <= 0 or not (1 <= prioritas <= 10):
                    continue

                aktivitas = Aktivitas(nama, durasi, prioritas)
                self.daftar_aktivitas.append(aktivitas)

                row_position = self.table_aktivitas.rowCount()
                self.table_aktivitas.insertRow(row_position)
                self.table_aktivitas.setItem(row_position, 0, QTableWidgetItem(nama))
                self.table_aktivitas.setItem(row_position, 1, QTableWidgetItem(str(durasi)))
                self.table_aktivitas.setItem(row_position, 2, QTableWidgetItem(str(prioritas)))
    except Exception as e:
        self.show_error(f"Gagal memuat file JSON:\n{e}")

```

Gambar 4.10 Kode Program Fungsi *Import JSON*

Gambar 4.11 menunjukkan implementasi logika penanganan *timeout* ketika proses pencarian solusi melebihi batas waktu yang ditentukan (maksimal 1 jam). Fungsi *handle\_timeout* akan dijalankan secara otomatis jika batas waktu terlampaui. Pada kondisi ini, sistem akan menghentikan proses algoritma yang masih berjalan dengan memanggil *self.worker.stop()*, lalu menampilkan status "Gagal: Waktu eksekusi melebihi batas (*Timeout*)" di antarmuka.

```
def handle_timeout(self):
    if self.worker.isRunning():
        self.worker.stop()
        self.label_skor.setText("Gagal: Waktu eksekusi melebihi batas (Timeout)")
        QMessageBox.critical(self, "Timeout", "Proses terlalu lama dan dihentikan.")
        self.tombol_proses.setEnabled(True)
```

Gambar 4.11 Kode Penanganan *Timeout* Jika Eksekusi Melebihi Batas Waktu

Gambar 4.12 menunjukkan proses implementasikan menggunakan fungsi `jadwal_optimal()` yang diimpor dari file `jadwal.py`. Program akan menghitung aktivitas-aktivitas terbaik berdasarkan prioritas dan durasi yang sesuai dengan total waktu yang diberikan, lalu menampilkan hasil dan metrik performa eksekusi.

```
def proses_jadwal(self):
    try:
        total_waktu = int(self.input_waktu.text())
        if total_waktu <= 0:
            raise ValueError
    except ValueError:
        self.show_error("Total waktu harus angka lebih dari 0.")
        return

    if not self.daftar_aktivitas:
        self.show_error("Tidak ada aktivitas yang ditambahkan.")
        return

    start_time = time.perf_counter()
    hasil = jadwal_optimal(self.daftar_aktivitas, total_waktu)
    end_time = time.perf_counter()

    elapsed_time_ms = (end_time - start_time) * 1000

    self.table_jadwal.setRowCount(0)
    for aktivitas in hasil['jadwal']:
        row_position = self.table_jadwal.rowCount()
        self.table_jadwal.insertRow(row_position)
        self.table_jadwal.setItem(row_position, 0, QTableWidgetItem(aktivitas.nama))
        self.table_jadwal.setItem(row_position, 1, QTableWidgetItem(str(aktivitas.durasi)))
        self.table_jadwal.setItem(row_position, 2, QTableWidgetItem(str(aktivitas.prioritas)))

    self.label_skor.setText(f"Total Skor: {hasil['skor']} | Waktu Eksekusi: {elapsed_time_ms:.2f} ms")
```

Gambar 4.12 Kode Program Fungsi Proses Jadwal

## 5. Implementasi

### 5.1 Input dan Output Program

Pada aplikasi penjadwalan aktivitas harian pribadi ini, pengguna memberikan input berupa data aktivitas yang ingin dijadwalkan. Setiap aktivitas terdiri dari tiga komponen utama,

yaitu nama aktivitas, durasi pelaksanaan, dan tingkat prioritas. Nama aktivitas diisi dalam bentuk teks yang menggambarkan kegiatan tertentu, misalnya “Belajar”, “Olahraga”, atau “Membaca Buku”. Durasi diinput dalam satuan menit dan harus berupa bilangan bulat positif, sedangkan prioritas diberikan dalam skala 1 hingga 10, di mana angka yang lebih besar menunjukkan tingkat kepentingan yang lebih tinggi. Selain itu, pengguna juga perlu memasukkan total waktu harian yang tersedia untuk penjadwalan, yang juga dimasukkan dalam satuan menit. Data aktivitas ini dapat ditambahkan secara manual melalui form yang tersedia pada tampilan antarmuka atau diimpor dari file eksternal dengan format JSON.

Output dari program ini ditampilkan dalam bentuk visual melalui antarmuka pengguna berbasis GUI (*Graphical User Interface*). Setelah pengguna menekan tombol “Proses Jadwal Optimal”, sistem akan menjalankan algoritma pencarian solusi optimal berbasis *backtracking* yang disempurnakan dengan strategi *greedy* dan *pruning*. Hasil dari proses ini adalah daftar aktivitas terpilih yang dianggap paling optimal untuk dimasukkan dalam jadwal, dengan mempertimbangkan batas waktu yang tersedia dan nilai prioritas masing-masing aktivitas. Aktivitas-aktivitas yang terpilih akan ditampilkan dalam bentuk tabel, lengkap dengan nama, durasi, dan prioritasnya. Selain itu, aplikasi juga akan menampilkan total skor yang merupakan jumlah nilai prioritas dari semua aktivitas yang berhasil dijadwalkan, serta waktu eksekusi algoritma dalam satuan milidetik. Informasi ini memberikan gambaran kepada pengguna mengenai efisiensi hasil penjadwalan yang telah dihasilkan oleh sistem.

## 5.2 Tampilan Program (GUI)

Antarmuka pengguna (GUI) pada aplikasi ini dirancang menggunakan library PyQt5, yang memungkinkan pengguna berinteraksi dengan sistem secara intuitif dan efisien. Gambar 5.1 menampilkan awal program yang terdiri dari beberapa komponen utama yang tersusun secara vertikal, yaitu kolom input nama aktivitas, durasi, dan prioritas, tombol untuk menambahkan aktivitas, serta tabel yang menampilkan daftar aktivitas yang telah dimasukkan.

Nama Aktivitas  
 Durasi (menit)  
 Prioritas (1-10)

Tambah Aktivitas  
 Import Aktivitas  
 Clear Aktivitas

Nama	Durasi	Prioritas
------	--------	-----------

Total Waktu (menit)

Proses Jadwal Optimal

Nama	Durasi	Prioritas
------	--------	-----------

Total Skor: 0

Gambar 5.1 Tampilan Form Input Aktivitas dan Tabel Daftar Aktivitas

Selain itu, terdapat juga tombol *Import* Aktivitas yang memungkinkan pengguna memuat data dari *file* JSON eksternal berformat sesuai struktur yang ditentukan. Aktivitas yang berhasil diimpor akan langsung ditampilkan pada tabel aktivitas di bawahnya, seperti ditampilkan pada Gambar 5.2. Tabel ini memperlihatkan setiap aktivitas yang telah dimasukkan lengkap dengan informasi nama, durasi, dan prioritas yang bersangkutan.

Nama Aktivitas  
 Durasi (menit)  
 Prioritas (1-10)

Tambah Aktivitas  
 Import Aktivitas  
 Clear Aktivitas

	Nama	Durasi	Prioritas
1	Olahraga Pagi	30	8
2	Belajar Algoritma	90	10
3	Makan Siang	45	7
4	Istirahat	60	6
5	Ngoding Proyek	120	9
6	Perang Saudara	500	9

Total Waktu (menit)

Proses Jadwal Optimal

Nama	Durasi	Prioritas
------	--------	-----------

Total Skor: 0

Gambar 5.2 Tabel Aktivitas yang Sudah Diinput

Selanjutnya, di bawah tabel aktivitas, pengguna dapat menentukan Total Waktu (dalam satuan menit) yang tersedia untuk menjadwalkan aktivitas-aktivitas tersebut. Setelah waktu ditentukan, pengguna dapat menekan tombol Proses Jadwal Optimal untuk menjalankan algoritma *backtracking* yang dikombinasikan dengan strategi *greedy* dan *pruning* guna memilih kombinasi aktivitas terbaik. Proses ini ditampilkan pada Gambar 5.3.

160

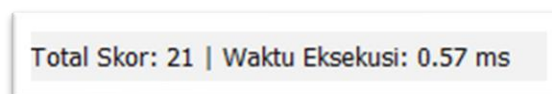
Proses Jadwal Optimal

	Nama	Durasi	Prioritas
1	Olahraga Pagi	30	8
2	Makan Siang	45	7
3	Istirahat	60	6

Total Skor: 21 | Waktu Eksekusi: 0.57 ms

Gambar 5.3 Tombol Proses dan Tabel Hasil Penjadwalan

Hasil dari proses penjadwalan akan langsung muncul dalam bentuk tabel kedua di bagian bawah antarmuka, menampilkan daftar aktivitas yang terpilih beserta durasi dan prioritasnya. Di bagian akhir, aplikasi juga menampilkan total skor dari aktivitas-aktivitas terjadwal serta waktu eksekusi algoritma dalam milidetik. Informasi ini penting untuk mengevaluasi performa dari solusi yang dihasilkan. Tampilan hasil ini diperlihatkan dengan jelas pada Gambar 5.4.



Total Skor: 21 | Waktu Eksekusi: 0.57 ms

Gambar 5.4 Label Total Skor dan Waktu Eksekusi

### 5.3 Penjelasan Cara Penggunaan

Aplikasi Penjadwalan Aktivitas Harian ini dirancang untuk membantu pengguna dalam menyusun jadwal aktivitas yang optimal berdasarkan durasi dan tingkat prioritas. Berikut ini adalah langkah-langkah penggunaan aplikasi yang dapat diikuti oleh pengguna:

## Langkah-langkah Penggunaan:

### 1. Persiapan Awal

A. Pastikan *Python* telah terinstal di komputer pengguna (disarankan menggunakan *Python* versi 3.8 atau lebih baru).

B. Unduh seluruh *file* program pada link berikut

(<https://github.com/MuhammadFadly090/Jadwal-harian->), termasuk:

- `main.py` (*file* utama untuk antarmuka pengguna)
- `jadwal.py` (berisi logika algoritma penjadwalan)
- Berkas pendukung lainnya seperti file JSON

C. Aktivasi *Virtual Environment*

- Untuk **Windows**: `venv\Scripts\activate`
- Untuk **macOS/Linux**: `source venv/bin/activate`

D. Menjalankan Aplikasi

Setelah *environment* aktif, jalankan aplikasi dengan perintah:

- `python main.py`

### 2. Menambahkan Aktivitas Secara Manual

Pengguna mengisi nama aktivitas, durasi (dalam satuan menit), dan tingkat prioritas (rentang nilai 1 hingga 10) pada kolom input yang tersedia. Setelah data terisi dengan benar, pengguna menekan tombol “Tambah Aktivitas”. Sistem akan memvalidasi input, kemudian menampilkan aktivitas yang valid pada tabel daftar aktivitas.

Olahraga

90

2

Tambah Aktivitas

Import Aktivitas

Clear Aktivitas

Nama	Durasi	Prioritas
------	--------	-----------

Gambar 5.5 Tampilan penambahan aktivitas manual oleh pengguna.

Nama Aktivitas

Durasi (menit)

Prioritas (1-10)

Tambah Aktivitas

Import Aktivitas

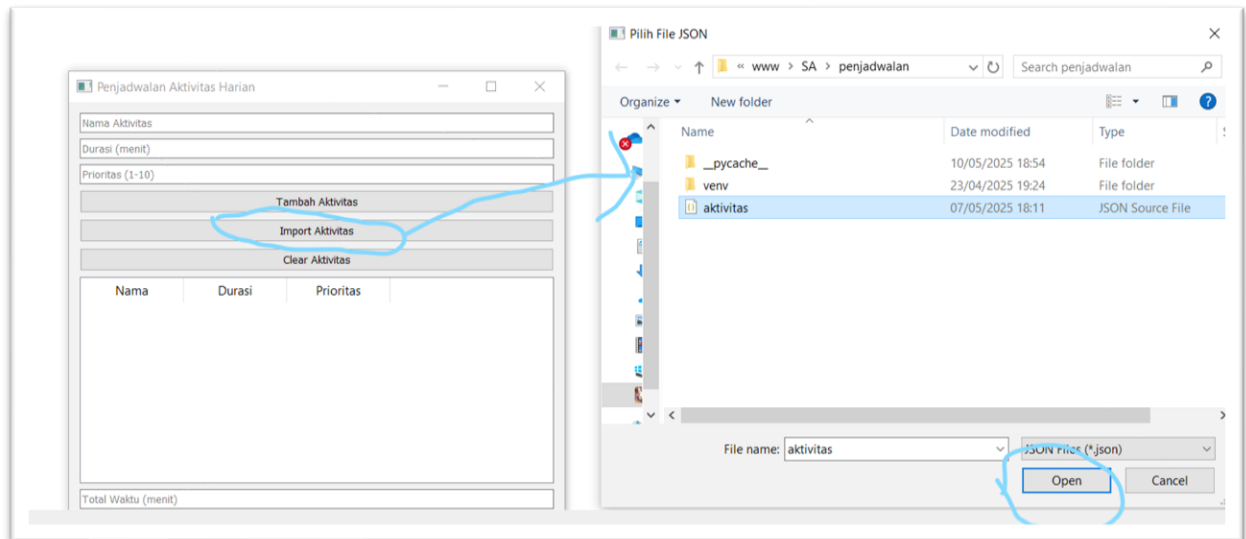
Clear Aktivitas

	Nama	Durasi	Prioritas
1	Olahraga	90	2

Gambar 5.6 Tampilan hasil penambahan aktivitas manual oleh pengguna.

### 3. Mengimpor Aktivitas dari File JSON

Selain input manual, aplikasi menyediakan fitur untuk mengimpor daftar aktivitas melalui *file* berekstensi .json. Pengguna menekan tombol “*Import Aktivitas*”, kemudian memilih file yang sesuai. Aplikasi akan membaca dan menampilkan data yang valid ke dalam tabel aktivitas.



Gambar 5.7 Tampilan proses *import* aktivitas dari *file JSON*.

Nama Aktivitas				
Durasi (menit)				
Prioritas (1-10)				
Tambah Aktivitas				
Import Aktivitas				
Clear Aktivitas				
	Nama	Durasi	Prioritas	
1	Olahraga Pagi	30	8	
2	Belajar Algoritma	90	10	
3	Makan Siang	45	7	
4	Istirahat	60	6	
5	Ngoding Proyek	120	9	
6	Perang Saudara	500	9	

Gambar 5.8 Tampilan hasil proses *import* aktivitas dari *file JSON*.

#### 4. Mengisi Total Waktu yang Tersedia

Pengguna harus menginputkan total waktu harian yang tersedia dalam satuan menit pada kolom “Total Waktu (menit)”. Nilai ini akan menjadi batas maksimal waktu yang dapat digunakan untuk menjadwalkan aktivitas.

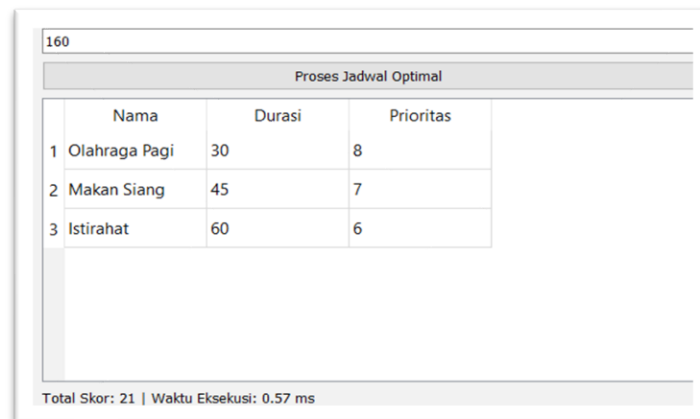
160
Proses Jadwal Optimal

Gambar 5.9 Input total waktu harian



## 5. Memproses Jadwal Optimal

Setelah daftar aktivitas lengkap dan total waktu telah diisi, pengguna dapat menekan tombol “Proses Jadwal Optimal”. Aplikasi akan menjalankan algoritma kombinasi *backtracking*, *greedy*, dan *pruning* untuk mencari kombinasi aktivitas terbaik yang dapat dimasukkan dalam batas waktu tersebut.



The screenshot shows a window titled "Proses Jadwal Optimal" with a table containing three rows of activity data. The table has columns for "Nama", "Durasi", and "Prioritas". Below the table, a status bar displays "Total Skor: 21 | Waktu Eksekusi: 0.57 ms".

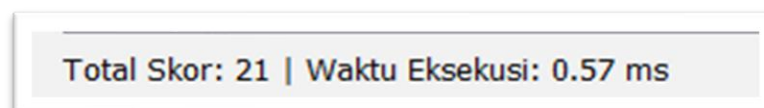
	Nama	Durasi	Prioritas
1	Olahraga Pagi	30	8
2	Makan Siang	45	7
3	Istirahat	60	6

Total Skor: 21 | Waktu Eksekusi: 0.57 ms

Gambar 5.10 Hasil pemrosesan jadwal

## 6. Melihat dan Mengevaluasi Hasil Jadwal

Hasil penjadwalan akan ditampilkan pada tabel “Hasil Jadwal”, yang memuat nama, durasi, dan prioritas dari setiap aktivitas terpilih. Selain itu, ditampilkan juga Total Skor, yaitu jumlah kumulatif dari nilai prioritas aktivitas terjadwal, serta Waktu Eksekusi algoritma dalam satuan milidetik. Hal ini bertujuan untuk memberikan gambaran efisiensi dan kualitas hasil penjadwalan.



The screenshot shows a single line of text: "Total Skor: 21 | Waktu Eksekusi: 0.57 ms".

Total Skor: 21   Waktu Eksekusi: 0.57 ms
--

Gambar 5.11 Tampilan hasil skor total dan waktu eksekusi

## 6. Pengujian

Pengujian dilakukan untuk memastikan bahwa aplikasi *Penjadwalan Aktivitas Harian* berjalan sesuai dengan fungsi yang dirancang serta memiliki performa yang baik dalam mengeksekusi proses penjadwalan. Pengujian dibagi menjadi dua bagian utama, yaitu:

- a) Pengujian Fungsional (*Functional Testing*)
- b) Pengujian Performa (*Performance Testing*)

### 6.1 Pengujian Fungsional

Pengujian ini bertujuan untuk memastikan bahwa setiap fitur dalam aplikasi bekerja sebagaimana mestinya. Berikut adalah skenario pengujian fungsional yang dilakukan:

Tabel 6.1 Pengujian Fungsional

No	Skenario Pengujian	Langkah Pengujian	Hasil yang Diharapkan	Status
1	Tambah aktivitas valid	Isi nama, durasi positif, prioritas 1–10 → klik tombol "Tambah Aktivitas"	Aktivitas masuk ke tabel	Berhasil
2	Tambah aktivitas tidak valid (nama kosong)	Kosongkan nama → klik tombol "Tambah Aktivitas"	Muncul peringatan validasi input	Berhasil
3	Tambah aktivitas dengan durasi/prioritas salah	Isi durasi negatif atau prioritas di luar 1–10 → klik tombol "Tambah Aktivitas"	Input ditolak dan pesan kesalahan ditampilkan	Berhasil
4	Import file JSON valid	Pilih file JSON berisi daftar aktivitas valid	Semua aktivitas ditampilkan di tabel	Berhasil
5	Import file JSON tidak valid	Pilih file JSON dengan format rusak atau salah	Aktivitas tidak ditambahkan, muncul pesan <i>error</i>	Berhasil
6	Proses jadwal optimal dengan input valid	Isi total waktu > 0 → klik "Proses Jadwal Optimal"	Jadwal optimal dan skor ditampilkan	Berhasil
7	Proses jadwal optimal tanpa aktivitas	Klik "Proses Jadwal Optimal" tanpa ada aktivitas	Muncul pesan peringatan	Berhasil
8	Clear semua input dan tabel	Klik tombol "Clear"	Semua input dan tabel dikosongkan	Berhasil

### 6.2 Pengujian Performa (Performance Testing)

Pengujian performa bertujuan untuk mengevaluasi efisiensi dan kestabilan aplikasi saat memproses berbagai jumlah data aktivitas, terutama ketika menggunakan algoritma

*backtracking* dengan strategi *greedy* dan *pruning*. Fokus utama pengujian ini adalah waktu eksekusi dan daya tahan aplikasi saat beban tinggi (*ekstrem*).

a) Tujuan Pengujian

- Mengukur waktu eksekusi proses penjadwalan aktivitas berdasarkan jumlah data yang diproses.
- Menilai stabilitas aplikasi pada kondisi penggunaan normal hingga ekstrem.

b) Informasi Perangkat Uji

Pengujian dilakukan menggunakan perangkat dengan spesifikasi sebagai berikut:

- **Prosesor:** Intel® Core™ i3-1115G4 (11th Gen) @ 3.00GHz
- **RAM:** 8 GB
- **Sistem Operasi:** Windows 10
- **Arsitektur Sistem:** 64-bit (x64-based)

c) Pengujian Performa

Keterangan :

- **Jumlah Aktivitas :** Menyatakan jumlah total aktivitas yang diuji dalam satu percobaan. Setiap aktivitas memiliki atribut durasi dan prioritas yang berbeda-beda
- **Total Waktu (menit) :** Batas waktu maksimal yang tersedia untuk menjadwalkan aktivitas. Algoritma akan memilih kombinasi aktivitas agar tidak melebihi waktu ini.
- **Range Durasi Aktivitas (menit) :** Rentang durasi tiap aktivitas yang dibangkitkan secara acak. Misalnya 10 – 50 berarti setiap aktivitas memiliki durasi antara 10 hingga 50 menit.
- **Waktu Eksekusi (ms) :** Waktu yang dibutuhkan algoritma untuk menemukan solusi terbaik, diukur dalam milidetik (ms). Ini mencerminkan efisiensi algoritma.
- **Status :** Menyatakan apakah algoritma berhasil menemukan solusi optimal sesuai batasan waktu. Jika algoritma berhasil dalam waktu maksimal 1 jam, maka statusnya "Berhasil". Jika tidak ada hasil dalam waktu 1 jam atau terjadi *timeout*/kesalahan, maka statusnya "Gagal".

Tabel 6.2 Hasil Pengujian Performa

No.	Jumlah Aktivitas	Total Waktu (menit)	Range Durasi Aktivitas (menit)	Waktu Eksekusi (ms)	Status
1	5	100	15 – 40	0.54 ms	Berhasil
2	10	180	10 – 50	0.51 ms	Berhasil
3	20	180	10 – 50	62.09 ms	Berhasil
4	30	180	10 – 50	1036.28 ms	Berhasil
5	40	180	10 – 50	10544.58 ms	Berhasil

No.	Jumlah Aktivitas	Total Waktu (menit)	Range Durasi Aktivitas (menit)	Waktu Eksekusi (ms)	Status
6	50	180	10 – 50	54058.03 ms	Berhasil
7	60	180	10 – 50	186639.83ms	Berhasil
8	70	180	10 – 50	674898.ms	Berhasil
9	80	180	10 – 50	1608552.59 ms	Berhasil
10	10	250	10 – 50	1.10 ms	Berhasil
11	30	300	10 – 50	55293.83ms	Berhasil
12	40	300	10 – 50	1854823.31 ms	Berhasil
13	50	300	10 – 50	>1 Jam	Gagal
14	80	300	10 – 50	>1 Jam	Gagal

## 7. Analisis

Metode yang digunakan dalam proyek ini adalah algoritma *backtracking* yang dipadukan dengan strategi *greedy* dan teknik *pruning* untuk menghasilkan penjadwalan aktivitas harian yang optimal berdasarkan durasi dan prioritas. Algoritma ini melakukan eksplorasi ruang solusi secara sistematis untuk menemukan kombinasi aktivitas dengan total prioritas tertinggi tanpa melebihi batas waktu yang tersedia.

Berdasarkan hasil uji coba yang ditampilkan pada Tabel 6.2, dapat disimpulkan bahwa performa algoritma sangat dipengaruhi oleh jumlah aktivitas yang diproses. Untuk jumlah aktivitas yang relatif kecil (misalnya  $\leq 30$ ), waktu eksekusi masih tergolong sangat cepat dan efisien. Namun, ketika jumlah aktivitas meningkat menjadi  $\geq 40$ , terjadi peningkatan waktu eksekusi yang signifikan. Sebagai contoh, pada pengujian dengan 80 aktivitas dan total waktu 180 menit, waktu eksekusi mencapai lebih dari 1,6 juta milidetik (sekitar 26 menit). Selain itu, untuk jumlah aktivitas 50 dan 80 dengan total waktu 300 menit, algoritma tidak dapat menyelesaikan proses dalam batas waktu 1 jam dan dinyatakan gagal.

Hal ini mencerminkan kompleksitas algoritma *backtracking* yang secara teoritis memiliki sifat eksponensial, yaitu  $O(2^n)$  dalam kasus terburuk, karena eksplorasi seluruh kombinasi *subset* aktivitas.

Meskipun demikian, penerapan strategi *pruning* terbukti memberikan kontribusi besar dalam efisiensi pencarian solusi. Dua pendekatan utama yang digunakan adalah:

- Menghentikan eksplorasi lebih awal apabila skor sementara ditambah skor sisa tidak akan mampu melampaui skor terbaik yang telah ditemukan.
- Mengakhiri proses lebih dini jika skor sementara telah mencapai ambang kepuasan, yaitu minimal 95% dari total skor maksimum yang mungkin dicapai.

Strategi ini sangat membantu dalam memperkecil ruang pencarian yang perlu dieksplorasi, terutama saat jumlah aktivitas masih moderat. Namun pada skala besar (aktivitas  $\geq 50$ ), ruang solusi tetap terlalu luas untuk diselesaikan dengan cepat tanpa bantuan pendekatan tambahan seperti paralelisme.

## 8. Kesimpulan dan Saran

### 8.1 Kesimpulan

Proyek ini berhasil merancang dan mengimplementasikan sistem penjadwalan aktivitas harian berbasis algoritma *backtracking* yang dioptimalkan dengan strategi *greedy* dan *pruning*. Sistem mampu menghasilkan kombinasi aktivitas terbaik berdasarkan prioritas dan durasi yang sesuai dengan batas waktu yang tersedia.

Dari pengujian yang dilakukan pada Tabel 6.2, diperoleh kesimpulan sebagai berikut:

- Algoritma mampu memberikan solusi optimal untuk kasus-kasus dengan jumlah aktivitas kecil hingga sedang ( $\leq 30$ ) dengan waktu eksekusi yang sangat cepat.
- Seiring bertambahnya jumlah aktivitas, performa menurun drastis akibat sifat eksponensial dari algoritma *backtracking*.
- Penerapan strategi *pruning* secara signifikan mengurangi jumlah cabang yang dieksplorasi, terutama pada skenario sedang, namun tetap tidak cukup efisien untuk skala besar.

- Kombinasi *greedy* (prioritas/durasi) dan *cut-off* skor 95% mampu menjaga kualitas hasil tanpa harus menjelajah seluruh solusi secara utuh.

Dengan demikian, solusi ini efektif untuk penjadwalan pribadi berskala terbatas, terutama saat kualitas jadwal lebih diutamakan dibanding waktu komputasi.

## 8.2 Saran

Untuk pengembangan lebih lanjut, ada beberapa rekomendasi yang dapat dilakukan agar sistem menjadi lebih optimal, antara lain:

- Optimalkan pemrosesan paralel (*multiprocessing*)

Mengingat waktu eksekusi bisa sangat lama saat jumlah aktivitas bertambah, ke depannya algoritma bisa dikembangkan lebih lanjut dengan memanfaatkan *multiprocessing* yang lebih efisien. Hal ini bertujuan agar pembagian pencarian solusi bisa dikerjakan secara paralel oleh banyak core CPU.

- Tambahkan metode pencatatan hasil sementara (*memoization*)

Supaya proses *backtracking* tidak mengulang-ulang perhitungan yang sama, bisa dicoba juga untuk menyimpan hasil sementara dalam *cache* atau menggunakan pendekatan *dynamic programming*. Ini akan menghemat waktu komputasi.

- Buat batasan pemangkasan (*pruning*) yang lebih ketat

Strategi *pruning* sudah cukup membantu, tapi mungkin masih bisa ditingkatkan lagi dengan menerapkan metode yang lebih agresif atau memanfaatkan prediksi skor maksimum dari sisa aktivitas.

Dengan menerapkan beberapa saran di atas, diharapkan sistem bisa menjadi lebih cepat, efisien, dan tetap memberikan hasil yang mendekati optimal bahkan untuk skenario yang kompleks.

## DAFTAR PUSTAKA

- Siregar, A., & Siregar, M. (2023). Penjadwalan Pekerjaan pada Manajemen Waktu dan Sumber Daya Menggunakan Algoritma Greedy. *Jurnal Sains dan Teknologi*, 2(1), 1–10. <https://jurnal.komputasi.org/index.php/jst/article/download/28/22/58>
- Winata, C. D. (2023). Penerapan Algoritma Greedy dan Backtracking dalam Menjadwalkan Kegiatan. Makalah IF2211 Strategi Algoritma, Institut Teknologi Bandung. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Makalah/Makalah-Stima-2023-%28113%29.pdf>
- Situmorang, J., & Sitorus, D. R. (2013). Penerapan Algoritma Greedy dalam Permainan Halma. Skripsi, Universitas Kristen Duta Wacana. <https://katalog.ukdw.ac.id/view/divisions/tek%3D5Finformatika/2013.type.html>
- Yulianto, D. (2017). Penjadwalan Matakuliah Menggunakan Algoritma Greedy: Studi Kasus Penjadwalan Semester Ganjil 2017-2018 Informatika ITENAS. *Jurnal Ilmiah*, 1(1), 1–10. <https://download.garuda.kemdikbud.go.id/article.php?article=1905348&title=Penjadwalan+Matakuliah+Menggunakan+Algoritma+Greedy+Studi+Kasus+Penjadwalan+Semester+Ganjil+2017-2018+Informatika+Itenas&Val=12612>
- Rahmawati, D. (2022). Implementasi Algoritma Greedy Kombinasi dengan Perulangan untuk Penjadwalan Praktikum. *Jurnal Sains dan Teknologi*, 3(2), 15–25. <https://jurnal.ilmubersama.com/index.php/sudo/article/download/8/16/130>
- Ardiansyah, R. (2021). Penentuan Aktivitas Mahasiswa Menggunakan Algoritma Greedy. Makalah IF2211 Strategi Algoritma, Institut Teknologi Bandung. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Makalah/Makalah-IF2211-Stima-2022-K1%20%286%29.pdf>
- Simanjuntak, J., Sihombing, A., Tambunan, E., Manurung, Z., & Simanjuntak, J. (2025). Perancangan Aplikasi Manajemen Waktu untuk Meningkatkan Produktivitas Kerja. *Jurnal Rekayasa Manajemen dan Informatika Komputer (REMIK)*, 2(1), 45–52.
- Ardani, A., Rachmadi, C. O., Sari, A. P., Raditya, N. G., Mutiara, S. L., & Yusuf, M. (2024). Implementasi Algoritma Greedy dan Dynamic Programming untuk Masalah Penjadwalan Interval dengan Model Knapsack. *Jurnal FORMAT: Jurnal Ilmiah Komputer dan Informatika*, 15(1), 23–30.
- Noovriyanto, Z. (2023). Penerapan Algoritma Backtracking Berbasis Blind Search untuk Menentukan Penjadwalan Mengajar. *Seminar Nasional Aplikasi Teknologi Informasi (SNATI)*, 20(1), 13–18.
- Trisanto, D. (2019). Optimasi Sistem Informasi Penjadwalan Kuliah Berbasis Heuristic Search yang Dikombinasikan dengan Teknik Smart Backtracking dan Look Ahead. *Jurnal Teknologi dan Manajemen*, 5(2), 67–74.