LAPORAN TUGAS BESAR
ANALISIS KOMPLEKSITAS ALGORITMA NON-OVERLAPPING TIME INTERVALS UNTUK PENJADWALAN

Disusun Oleh:
Kelompok 9

Achmad Baihaqie Wibowo	**103012400026**
Rafi Dzaki Azhari	**103012400336**
Jean Yudhistira Diva Waluyo	**103012400113**

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY

2025

DAFTAR ISI

BAB I
PENDAHULUAN

1.1. Latar Belakang

Dalam berbagai sistem penjadwalan, baik pada sistem akademik, sistem manajemen kegiatan, maupun aplikasi pengelolaan waktu, sering ditemukan permasalahan berupa konflik jadwal akibat tumpang tindih interval waktu antar aktivitas. Permasalahan ini muncul ketika suatu aktivitas baru ditambahkan ke dalam jadwal yang sudah ada tanpa mekanisme pengecekan konflik yang sistematis. Akibatnya, dua atau lebih aktivitas dapat terjadwal pada waktu yang sama, sehingga menurunkan validitas dan keandalan sistem penjadwalan tersebut. Secara empiris, pengecekan konflik waktu sering dilakukan secara manual atau dengan logika sederhana yang tidak terstruktur, sehingga kurang efisien ketika jumlah aktivitas meningkat.

Secara teoretis, permasalahan konflik jadwal dapat dimodelkan sebagai permasalahan interval waktu, di mana setiap aktivitas direpresentasikan oleh pasangan waktu mulai (*start time*) dan waktu selesai (*end time*). Dua interval dikatakan saling bertumpang tindih (*overlapping*) apabila interval waktu tersebut beririsan. Dalam ranah algoritma dan struktur data, permasalahan ini termasuk ke dalam kategori interval scheduling problem, yang menuntut solusi algoritmik untuk mendeteksi dan mencegah konflik interval secara sistematis. Algoritma Non-Overlapping Intervals merupakan salah satu pendekatan yang digunakan untuk memeriksa validitas suatu interval terhadap kumpulan interval lain dengan membandingkan batas waktu mulai dan selesai secara terdefinisi.

Urgensi dari permasalahan ini terletak pada kebutuhan akan algoritma yang bersifat umum, efisien, dan mudah diimplementasikan untuk menangani konflik interval waktu. Tanpa algoritma yang jelas dan terstruktur, sistem penjadwalan berisiko menghasilkan jadwal yang tidak valid. Oleh karena itu, diperlukan perancangan dan analisis algoritma Non-Overlapping Intervals yang mampu menyaring aktivitas bebas konflik dari sekumpulan kandidat aktivitas. Selain itu, pembahasan algoritma dalam dua pendekatan, yaitu iteratif dan rekursif, menjadi penting untuk memahami perbedaan karakteristik, efisiensi, serta implikasi implementasi dari masing-masing pendekatan dalam penyelesaian permasalahan konflik interval waktu.

1.2. Batasan Masalah

Agar pembahasan dalam laporan ini tetap terfokus dan terarah, maka diberikan beberapa batasan permasalahan. Analisis yang dilakukan pada algoritma Non-Overlapping Intervals dibatasi hanya pada kompleksitas waktu (time complexity) algoritma. Perhitungan dan pembahasan mengenai kompleksitas ruang (*space complexity*) tidak disertakan dalam laporan ini.

Selain itu, algoritma yang dibahas hanya mencakup dua pendekatan, yaitu pendekatan iteratif dan rekursif, dengan asumsi struktur data yang digunakan berupa array statis. Analisis kompleksitas waktu dilakukan berdasarkan parameter input yang telah ditentukan, tanpa mempertimbangkan optimasi tambahan, variasi struktur data, maupun implementasi spesifik pada perangkat keras atau bahasa pemrograman tertentu.

1.3. Rumusan Masalah

1. Apa parameter *input size* utama dari algoritma *Non-Overlapping Time Intervals for Scheduling*?
2. Apa operasi dasar atau operasi khas dari algoritma *Non-Overlapping Time Intervals for Scheduling*?
3. Kapan algoritma mengalami kondisi kasus terbaik, terburuk, dan rata-rata?
4. Bagaimana perhitungan kompleksitas waktu algoritma *Non-Overlapping Time Intervals for Scheduling* menggunakan pendekatan iteratif pada kondisi kasus terbaik (*best case*), kasus terburuk (*worst case*), dan kasus rata-rata (*average case*)?
5. Bagaimana perhitungan kompleksitas waktu algoritma *Non-Overlapping Time Intervals for Scheduling* menggunakan pendekatan rekursif pada kondisi kasus terbaik (*best case*), kasus terburuk (*worst case*), dan kasus rata-rata (*average case*)?
6. Bagaimana implementasi algoritma *Non-Overlapping Time Intervals for Scheduling* ke dalam bentuk perangkat lunak sederhana berdasarkan pendekatan iteratif dan rekursif?
7. Bagaimana perbandingan kinerja algoritma antara pendekatan iteratif dan rekursif berdasarkan kompleksitas waktu dan *running time* yang dihasilkan?

1.4. Tujuan

Adapun tujuan dari penulisan laporan ini adalah sebagai berikut:

1. Mengidentifikasi parameter *input size* utama yang mempengaruhi performa algoritma *Non-Overlapping Time Intervals for Scheduling.*
2. Menentukan operasi dasar yang menjadi unit perhitungan utama dalam kompleksitas waktu algoritma tersebut.
3. Menganalisis dan memetakan kondisi spesifik yang menyebabkan algoritma berada pada kasus terbaik, kasus terburuk, dan kasus rata-rata.
4. Melakukan perhitungan kompleksitas waktu secara matematis pada pendekatan iteratif untuk ketiga skenario kasus (terbaik, terburuk, dan rata-rata).
5. Melakukan perhitungan kompleksitas waktu secara matematis pada pendekatan rekursif untuk ketiga skenario kasus (terbaik, terburuk, dan rata-rata).
6. Mengimplementasikan algoritma ke dalam perangkat lunak sederhana berbasis web interaktif sederhana untuk menguji validitas kedua pendekatan secara langsung.
7. Menganalisis perbandingan kinerja antara pendekatan iteratif dan rekursif dengan mengomparasi hasil kompleksitas teoritis terhadap running time empiris yang diperoleh dari simulasi.

1.5. Manfaat

Penulisan laporan ini diharapkan dapat memberikan manfaat dalam meningkatkan pemahaman teoritis dan praktis mengenai analisis kompleksitas waktu algoritma *Non-Overlapping Time Intervals for Scheduling*, khususnya melalui perbandingan pendekatan iteratif dan rekursif. Selain itu, laporan ini dapat menjadi referensi akademik dalam penerapan konsep analisis algoritma, serta memberikan acuan dalam perancangan dan pengembangan perangkat lunak yang membutuhkan mekanisme penjadwalan bebas konflik waktu secara sistematis dan efisien.

BAB II
LANDASAN TEORI

2.1 Non-Overlapping Time Intervals for Scheduling

Dua interval waktu [s1, e1] dan [s2, e2] dikatakan non-overlapping atau tidak tumpang tindih jika dan hanya jika interval pertama selesai sebelum interval kedua dimulai, atau sebaliknya. Jika syaratnya tidak terpenuhi, maka terjadi tabrakan secara interval waktu, sehingga nantinya akan terdapat aktivitas yang konflik secara waktu pelaksanaan.

Jadwal Wajib atau dalam kajian ini dikenalkan sebagai *term* Mandatory Schedule adalah himpunan aktivitas utama yang bersifat terfiksasi dan tidak dapat diganggu gugat. Jadwal wajib harus bersifat konsisten secara internal, dalam artian di dalam daftar jadwal wajib itu sendiri sudah dipastikan tidak ada aktivitas yang tumpang tindih secara interval waktu.

Jadwal Kandidat atau dalam kajian ini dikenalkan sebagai *term* Candidate Schedule merupakan himpunan aktivitas baru yang bersifat dinamis atau tentatif. Aktivitas-aktivitas ini adalah "pemohon" yang ingin masuk ke dalam jadwal utama. Berbeda dengan jadwal wajib, daftar kandidat mungkin saja memiliki konflik internal di antara sesama kandidat.

Algoritma bertugas dalam hal filterisasi, yakni memeriksa setiap elemen dalam Candidate Schedule dan memvalidasinya terhadap seluruh isi Mandatory Schedule. Jika suatu aktivitas kandidat ditemukan menabrak secara interval waktu dengan minimal salah satu aktivitas dari jadwal wajib, maka kandidat tersebut akan ditolak. Hasil akhirnya adalah himpunan bagian dari Jadwal Kandidat yang seluruh isi aktivitas nya tidak konflik dengan seluruh aktivitas pada jadwal wajib yang sudah ada.

2.2 Pendekatan Iteratif

Pendekatan iteratif merupakan metode penyelesaian masalah algoritmik yang menggunakan struktur perulangan untuk menjalankan serangkaian instruksi secara berulang hingga kondisi tertentu terpenuhi. Dalam pendekatan ini, proses penyelesaian masalah dilakukan secara bertahap dengan memperbarui nilai variabel pada setiap iterasi, tanpa melibatkan pemanggilan fungsi secara berulang seperti pada pendekatan rekursif. Pendekatan iteratif umumnya memanfaatkan struktur kontrol seperti *for*, *while*, atau *do–while*.

Secara konseptual, ide berpikir pada pendekatan iteratif adalah memecah permasalahan menjadi langkah-langkah berurutan yang dieksekusi satu per satu dalam suatu alur yang linier. Setiap iterasi merepresentasikan satu langkah pemrosesan terhadap elemen data tertentu, dan hasil sementara dapat langsung digunakan atau diperbarui pada iterasi berikutnya. Pendekatan ini memungkinkan pengendalian alur program yang lebih eksplisit dan mudah dilacak, karena seluruh proses eksekusi terjadi dalam satu kerangka kontrol yang jelas.

Dalam konteks algoritma, pendekatan iteratif sering dipilih karena memiliki efisiensi penggunaan memori yang lebih baik dibandingkan pendekatan rekursif, sebab tidak memerlukan penyimpanan *call stack* tambahan. Selain itu, pendekatan iteratif cenderung lebih stabil dan mudah dioptimalkan, terutama ketika digunakan untuk memproses data dalam jumlah besar. Oleh karena itu, pendekatan iteratif banyak diterapkan dalam permasalahan yang melibatkan pengolahan array, pencarian, penyaringan data, serta analisis konflik atau perbandingan elemen secara berulang.

2.3. Pendekatan Rekursif

Pendekatan rekursif merupakan metode penyelesaian masalah algoritma yang menggunakan pemanggilan fungsi terhadap dirinya sendiri untuk memecah permasalahan menjadi sub-permasalahan yang lebih kecil. Dalam pendekatan ini, suatu fungsi akan terus memanggil dirinya sendiri hingga mencapai kondisi berhenti yang disebut sebagai basis rekursi. Setelah basis rekursi terpenuhi, proses eksekusi akan kembali secara bertahap ke pemanggilan sebelumnya hingga seluruh proses selesai.

Ide berpikir pada pendekatan rekursif didasarkan pada prinsip *divide and conquer*, yaitu membagi masalah utama menjadi bagian-bagian yang memiliki struktur serupa dengan masalah awal. Setiap pemanggilan rekursif merepresentasikan satu langkah pemrosesan terhadap sebagian data, sementara hasil akhir diperoleh melalui penggabungan atau pengembalian nilai dari pemanggilan fungsi tersebut. Pendekatan ini memungkinkan penyelesaian masalah yang bersifat berulang dan terstruktur secara konseptual menjadi lebih sederhana dan mudah dipahami.

Dalam penerapan algoritma, pendekatan rekursif sering digunakan untuk permasalahan yang memiliki pola pengulangan alami, seperti penelusuran data, pemrosesan struktur hierarkis, dan pemeriksaan berurutan terhadap elemen data. Meskipun pendekatan rekursif dapat meningkatkan kejelasan logika algoritma, pendekatan ini memiliki keterbatasan pada penggunaan memori karena setiap pemanggilan fungsi memerlukan ruang pada *call stack*. Oleh karena itu, pemilihan pendekatan rekursif perlu mempertimbangkan ukuran input dan efisiensi eksekusi agar tidak menimbulkan overhead yang berlebihan.

BAB III
PERANCANGAN ALGORITMA

3.1. Pendekatan Iteratif

Fungsi Utama

Algorithm 2 Get Conflict Free Activities (Iterative)

```
function GET_CONFLICT_FREE_ACTIVITIES_ITERATIVE
  (
    Candidates : array[1..maxC] of Activity,
    Mandatory : array[1..maxM] of Activity,
    nC, nM : integer
  ) → array of Activity
  Kamus
    current : Activity
    startTime, endTime, nR, i : integer
    Result : array[1..maxC] of Activity
  Algoritma
    Result ← emptyActivityArray
    nR ← 0
    i ← 1
    while i ≤ nC do
      current ← Candidates[i]
      startTime ← current.start.hour × 60 + current.start.minute
      endTime ← current.end.hour × 60 + current.end.minute
      if IS_TIME_CONFLICT_FREE_ITERATIVE(Mandatory, nM, current, startTime, endTime) then
        nR ← nR + 1
        Result[nR] ← current
      end if
      i ← i + 1
    end while
    return Result
end function
```

Fungsi Helper

Algorithm 1 Check Time Conflict (Iterative)

```
function IS_TIME_CONFLICT_FREE_ITERATIVE
  (
    Mandatory : array[1..maxM] of Activity,
    nM : integer,
    current : Activity,
    startTime : integer,
    endTime : integer
  ) → boolean
  Kamus
    i, mandatoryStart, mandatoryEnd : integer
  Algoritma
    i ← 1
    while i ≤ nM do
      mandatoryStart ← Mandatory[i].start.hour × 60 + Mandatory[i].start.minute
      mandatoryEnd ← Mandatory[i].end.hour × 60 + Mandatory[i].end.minute
      if current.day = Mandatory[i].day then
        if (startTime < mandatoryEnd and endTime > mandatoryStart) then
          return false
        end if
      end if
      i ← i + 1
    end while
    return true
end function
```

Fungsi `GET_CONFLICT_FREE_ACTIVITIES_ITERATIVE` melakukan iterasi terhadap seluruh aktivitas dari Jadwal Kandidat sebanyak `nC` kali. Pada setiap iterasi, fungsi `IS_TIME_CONFLICT_FREE_ITERATIVE` dipanggil untuk memeriksa apakah aktivitas dari Jadwal Kandidat tersebut memiliki konflik waktu dengan seluruh aktivitas dari Jadwal Wajib sebanyak `nM` kali. Jika aktivitas dari Jadwal Kandidat tersebut tidak memiliki konflik waktu secara hari dan interval waktu dengan seluruh aktivitas mana pun pada Jadwal Wajib, maka aktivitas dari Jadwal Kandidat tersebut ditambahkan ke dalam struktur data hasil, dalam hal ini array `Result`. Proses iterasi berlanjut hingga seluruh aktivitas dari Jadwa Kandidat selesai diperiksa.

3.2. Pendekatan Rekursif

Fungsi Utama

Algorithm 4 Get Conflict Free Activities (Recursive)

```
function GET_CONFLICT_FREE_ACTIVITIES_RECURSIVE
  (
    Candidates, Result : array[1..maxC] of Activity,
    Mandatory : array[1..maxM] of Activity,
    i, nC, nR, nM : integer
  ) → array of Activity
  Kamus
    current : Activity
    startTime, endTime : integer
  Algoritma
    if i = 1 then
      Result ← emptyActivityArray
    end if
    if i > nC then
      return Result
    end if
    current ← Candidates[i]
    startTime ← current.start.hour × 60 + current.start.minute
    endTime ← current.end.hour × 60 + current.end.minute
    if IS_TIME_CONFLICT_FREE_RECURSIVE(Mandatory, nM, current, startTime, endTime, 1) then
      nR ← nR + 1
      Result[nR] ← current
    end if
    return GET_CONFLICT_FREE_ACTIVITIES_RECURSIVE
      (
        Candidates,
        Result,
        Mandatory,
        i + 1,
        nC,
        nR,
        nM
      )
end function
```

Fungsi Helper

Algorithm 3 Check Time Conflict (Recursive)

```
function IS_TIME_CONFLICT_FREE_RECURSIVE
  (
    Mandatory : array[1..maxM] of Activity,
    nM : integer,
    current : Activity,
    startTime : integer,
    endTime : integer,
    i : integer
  ) → boolean
  Kamus
    mandatoryStart, mandatoryEnd : integer
  Algoritma
    if i > nM then
      return true
    end if
    mandatoryStart ← Mandatory[i].start.hour × 60 + Mandatory[i].start.minute
    mandatoryEnd ← Mandatory[i].end.hour × 60 + Mandatory[i].end.minute
    if current.day = Mandatory[i].day then
      if (startTime < mandatoryEnd and endTime > mandatoryStart) then
        return false
      end if
    end if
    return IS_TIME_CONFLICT_FREE_RECURSIVE(Mandatory, nM, current, startTime, endTime, i + 1)
end function
```

Berbeda dengan pendekatan iteratif, pendekatan rekursif menyelesaikan persoalan pemeriksaan konflik waktu dengan cara mendekomposisi daftar kandidat melalui pemanggilan fungsi yang berulang terhadap dirinya sendiri. Fungsi `GET_CONFLICT_FREE_ACTIVITIES_RECURSIVE` menelusuri daftar kandidat sebanyak `nC` kali, di mana setiap pemanggilan fungsi mewakili pemeriksaan satu indeks aktivitas kandidat tertentu. Dalam setiap tingkatan rekursi, fungsi helper digunakan untuk melakukan verifikasi terhadap `nM` aktivitas dalam Jadwal Wajib guna mengidentifikasi adanya tumpang tindih waktu. Inti dari perancangan ini terletak pada mekanisme akumulasi hasil, di mana algoritma akan memutuskan untuk menyertakan aktivitas kandidat ke dalam array hasil hanya jika pemeriksaan terhadap jadwal wajib memberikan nilai valid, kemudian meneruskan sisa daftar kandidat ke tingkat rekursi berikutnya. Proses ini akan terus menyelam ke dalam lapisan rekursi yang lebih dalam hingga mencapai *base case*, yaitu ketika seluruh kandidat telah selesai diperiksa atau indeks telah melampaui batas daftar. Setelah mencapai titik tersebut, fungsi akan melakukan proses pengembalian nilai sambil menyusun kembali aktivitas-aktivitas yang tidak memiliki konflik ke dalam struktur data hasil, dalam hal ini array `Result` secara utuh.

BAB IV
ANALISIS DAN PEMBAHASAN

4.1. Identifikasi Parameter Input Size Algoritma *Non-Overlapping Time Intervals for Scheduling*

Dalam analisis efisiensi algoritma ini, didefinisikan dan ditetapkan dua parameter *input size*:

- `nC` sebagai jumlah aktivitas kandidat atau banyaknya elemen pada array `Candidates`.
- `nM` sebagai jumlah aktivitas pada jadwal wajib atau banyaknya elemen pada array `Mandatory`.

4.2. Identifikasi Operasi Dasar Algoritma *Non-Overlapping Time Intervals for Scheduling*

Operasi dasar dalam algoritma ini adalah operasi perbandingan yang berada di dalam struktur pengulangan terdalam. Secara spesifik, operasi tersebut adalah pengecekan kondisi bentrok waktu:

```
(startTime < mandatoryEnd AND endTime > mandatoryStart)
```

4.3 Identifikasi Kondisi Kasus Terbaik, Terburuk, dan Rata-rata dari Algortima *Non-Overlapping Time Intervals for Scheduling*

4.3.1. Best Case

Terjadi ketika tiap aktivitas dari Jadwal Kandidat langsung mengalami konflik pada pemeriksaan pertama di fungsi helper. Dengan kata lain, perbandingan `(current.day == Mandatory[i].day)` dan perbandingan `(startTime < mandatoryEnd AND endTime > mandatoryStart)` selalu bernilai `true` pada pemeriksaan pertama di fungsi `helper` untuk tiap iterasi aktivitas kandidatnya.

4.3.2. Worst Case

Terjadi ketika tidak ada konflik waktu untuk seluruh aktivitas kandidat terhadap seluruh aktivitas jadwal wajib yang sudah ada. Dengan kata lain, untuk setiap aktivitas kandidat dilakukan pemeriksaan terhadap seluruh `nM` aktivitas. Atau mungkin bentrok ditemukan pada elemen terakhir jadwal wajib.

4.3.3. Average Case

Terjadi ketika sebagian aktivitas kandidat mengalami konflik dan sebagian tidak mengalami konflik. Dengan kata lain perbandingan `(startTime < mandatoryEnd AND endTime > mandatoryStart)` tidak selalu bernilai `true` atau `false`.

4.4. Analisis Efisiensi Algoritma *Non-Overlapping Time Intervals for Scheduling* Versi Iteratif

4.4.1. Best Case

$$
\begin{aligned}
T_{min}(nC, nM) &= \sum_{i=1}^{nC} 1 \\
T_{min}(nC, nM) &= (nC - 1 + 1) \times 1 \\
T_{min}(nC, nM) &= nC \\
T_{min}(nC, nM) &\in \Theta(nC) \\
\therefore T_{min}(nC, nM) &\text{ berkelas linear}
\end{aligned}
$$

4.4.2. Worst Case

$$
\begin{aligned}
T_{max}(nC, nM) &= \sum_{i=1}^{nC} \sum_{j=1}^{nM} 1 \\
T_{max}(nC, nM) &= \sum_{i=1}^{nC} (nM - 1 + 1) \times 1 \\
T_{max}(nC, nM) &= \sum_{i=1}^{nC} nM \\
T_{max}(nC, nM) &= (nC - 1 + 1) \times nM \\
T_{max}(nC, nM) &= nC \times nM \\
T_{max}(nC, nM) &\in \Theta(nC \times nM) \\
\therefore T_{max}(nC, nM) &\text{ berkelas kuadratik}
\end{aligned}
$$

4.4.3. Average Case

$$T_{avg}(nC, nM) = \frac{\sum_{i=1}^{T_{max}(nC,nM)} i}{T_{max}(nC, nM)}$$

$$T_{avg}(nC, nM) = \frac{1 + 2 + 3 + \cdots + T_{max}(nC, nM)}{T_{max}(nC, nM)}$$

$$T_{avg}(nC, nM) = \frac{\frac{T_{max}(nC,nM) \times [T_{max}(nC,nM)+1]}{2}}{T_{max}(nC, nM)}$$

$$T_{avg}(nC, nM) = \frac{T_{max}(nC, nM) + 1}{2}$$

$$T_{avg}(nC, nM) = \frac{nC \times nM + 1}{2} \in \Theta(nC \times nM)$$

$$\therefore T_{avg}(nC, nM) \text{ berkelas kuadratik}$$

Catatan: Kondisi terburuk dan rata-rata berkelas **Θ**(*nC***nM*), secara klasifikasi kelas kompleksitas waktu formal dapat digolongkan sebagai kuadratik jika dan hanya jika nC dan nM keduanya tumbuh secara proporsional. Jika *nM* konstan dengan *nC* bergerak maka kompleksitas tersebut tumbuh secara linear terhadap pertumbuhan *nC* dan Jika *nC* konstan maka kompleksitas tersebut tumbuh secara linear terhadap pertumbuhan *nM*, fenomena seperti ini dalam dunia matematika khususnya kajian kalkulus multivariat dapat digolongkan fungsi bilinear. Pada kelas algoritma kali ini yang sedang dikaji terlihat bahwa *input size* ada dua, hal ini tetap valid dibuktikan dengan kajian yang sudah umum di ranah analisis algoritma terkait *time complexity* untuk algoritma DFS dan BFS.

4.5. Analisis Efisiensi Algoritma *Non-Overlapping Time Intervals for Scheduling* Versi Rekursif

4.5.1. Best Case

Relasi Rekurensi :

$$T_{\min}(nC, nM) = \begin{cases} 1, & nC = 1 \\ T_{\min}(nC-1, nM) + 1, & nC > 1 \end{cases}$$

Perhitungan (Metode Substitusi):

$$T_{\min}(nC, nM) = T_{\min}(nC-1, nM) + 1$$
$$T_{\min}(nC, nM) = T_{\min}(nC-2, nM) + 2$$
$$T_{\min}(nC, nM) = T_{\min}(nC-3, nM) + 3$$
$$\vdots$$
$$T_{\min}(nC, nM) = T_{\min}(nC-k, nM) + k$$

Akan berhenti saat $nC - k = 1$

$$\Leftrightarrow k = nC - 1$$

sehingga :

$$T_{\min}(nC, nM) = T_{\min}(1, nM) + nC - 1$$
$$T_{\min}(nC, nM) = 1 + nC - 1$$
$$T_{\min}(nC, nM) = nC \in \Theta(nC)$$

4.5.2. Worst Case

Relasi Rekurensi :

$$T_{\max}(nC, nM) = \begin{cases} nM, & nC = 1 \\ T_{\max}(nC-1, nM) + nM, & nC > 1 \end{cases}$$

Perhitungan :

$$T_{\max}(nC, nM) = T_{\max}(nC-1, nM) + nM$$
$$T_{\max}(nC, nM) = T_{\max}(nC-2, nM) + 2 \times nM$$
$$T_{\max}(nC, nM) = T_{\max}(nC-3, nM) + 3 \times nM$$
$$\vdots$$
$$T_{\max}(nC, nM) = T_{\max}(nC-k, nM) + k \times nM$$

Akan berhenti saat $nC - k = 1$

$$\Leftrightarrow k = nC - 1$$

sehingga :

$$T_{\max}(nC, nM) = T_{\max}(1, nM) + (nC-1) \times nM$$
$$T_{\max}(nC, nM) = nM + nC \times nM - nM$$
$$T_{\max}(nC, nM) = nC \times nM \in \Theta(nC \times nM)$$

4.5.3. Average Case

$$T_{avg}(nC, nM) = \frac{\sum_{i=1}^{T_{max}(nC,nM)} i}{T_{max}(nC, nM)}$$

$$T_{avg}(nC, nM) = \frac{1 + 2 + 3 + \cdots + T_{max}(nC, nM)}{T_{max}(nC, nM)}$$

$$T_{avg}(nC, nM) = \frac{\frac{T_{max}(nC,nM) \times [T_{max}(nC,nM)+1]}{2}}{T_{max}(nC, nM)}$$

$$T_{avg}(nC, nM) = \frac{T_{max}(nC, nM) + 1}{2}$$

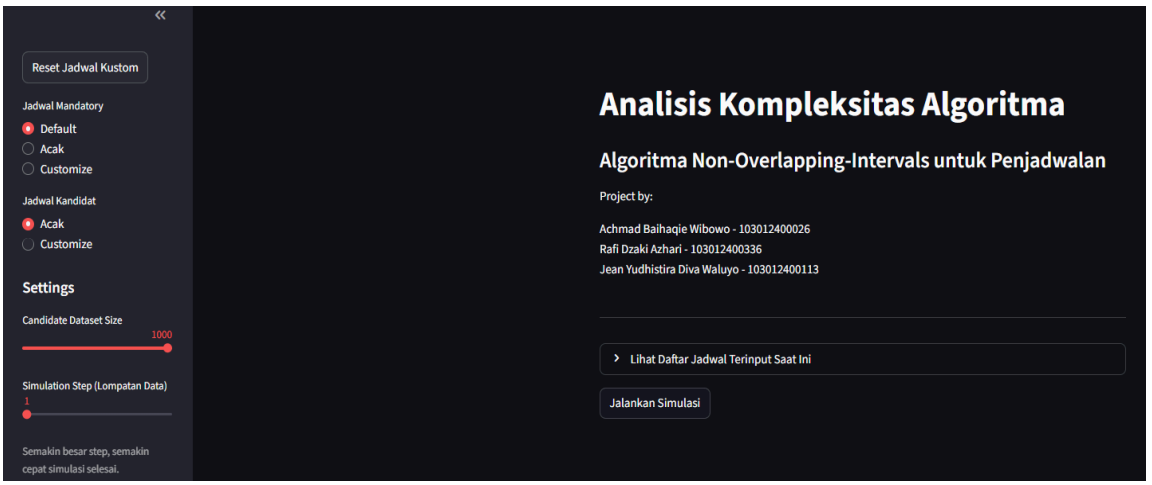
$$T_{avg}(nC, nM) = \frac{nC \times nM + 1}{2} \in \Theta(nC \times nM)$$

$$\therefore T_{avg}(nC, nM) \text{ berkelas kuadratik}$$

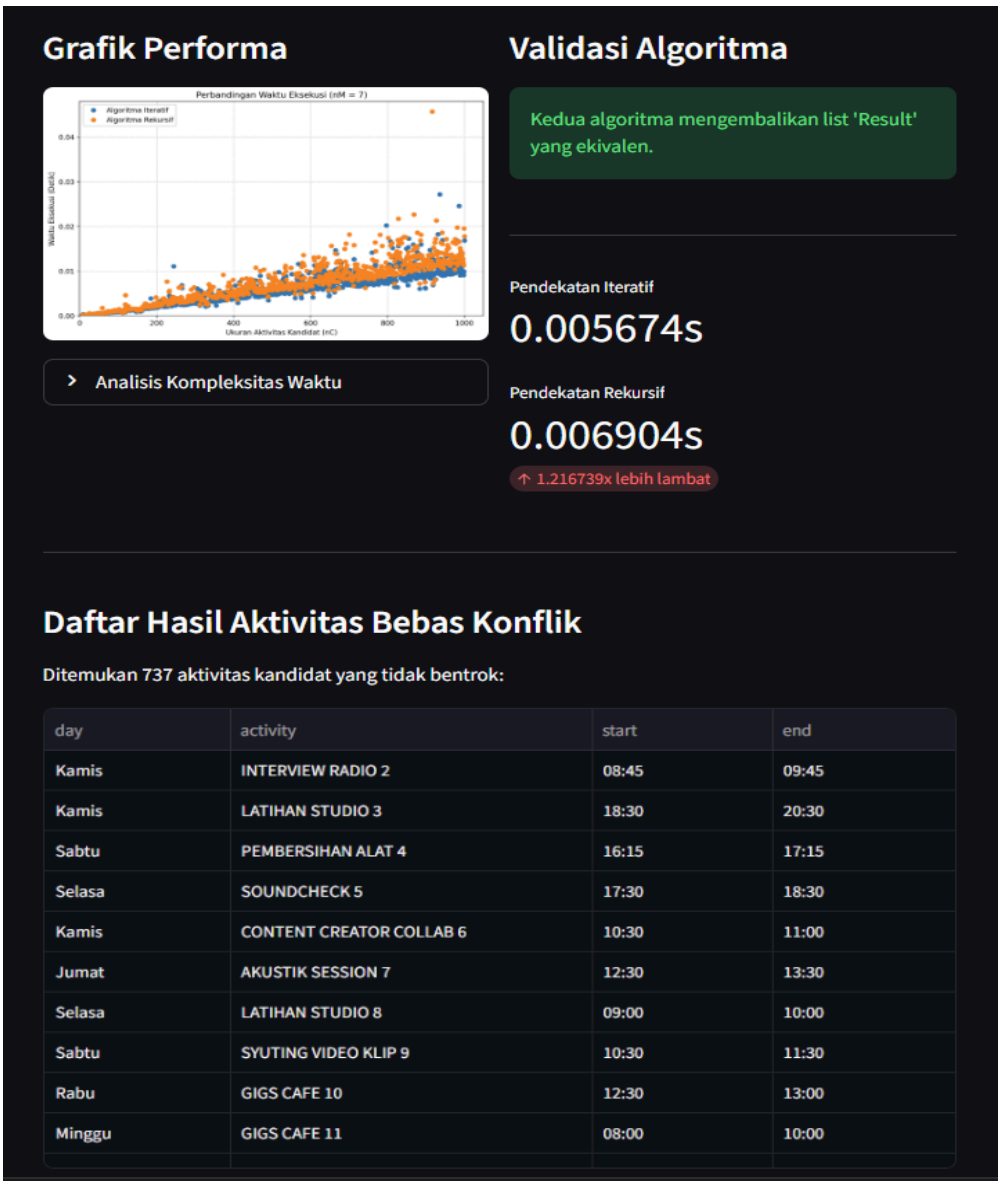
4.6. Implementasi Perangkat Lunak Sederhana Menggunakan Prinsip Dasar *Non-Overlapping Time Intervals for Scheduling* untuk Kebutuhan Penjadwalan

Perangkat lunak pada penelitian ini dikembangkan menggunakan bahasa pemrograman Python sebagai bahasa utama dalam implementasi algoritma. Untuk pengembangan antarmuka pengguna berbasis web, digunakan framework Streamlit, yang memungkinkan pembuatan aplikasi web interaktif secara cepat dan sederhana tanpa memerlukan pengelolaan *frontend* secara manual. Selain itu, beberapa pustaka pendukung digunakan, antara lain Pandas untuk pengolahan dan penyajian data dalam bentuk tabel, Matplotlib untuk visualisasi grafik kinerja algoritma, serta modul time untuk pengukuran waktu eksekusi algoritma. Pengaturan batas rekursi dilakukan menggunakan modul sys untuk menjaga stabilitas eksekusi pada ukuran data yang besar.

Sebagai contoh implementasi, data *default* yang digunakan adalah penjadwalan band untuk sebuah konser, setiap band memiliki jadwal tampil dengan waktu tertentu. Ketika sebuah band ingin menambahkan jadwal yang baru, sistem harus memastikan bahwa jadwal itu tidak bertabrakan dengan jadwal lain di hari yang sama.



Antarmuka pengguna terdiri atas *sidebar* sebagai pengatur parameter simulasi dan halaman utama sebagai media visualisasi hasil. Sidebar menyediakan pengaturan jenis data jadwal (*default*, acak, atau kustom), ukuran dataset kandidat, serta langkah simulasi. Halaman utama menampilkan tabel jadwal, grafik perbandingan waktu eksekusi, hasil validasi algoritma, dan daftar aktivitas kandidat yang bebas konflik. Struktur UI ini dirancang agar alur eksperimen mudah dipahami.



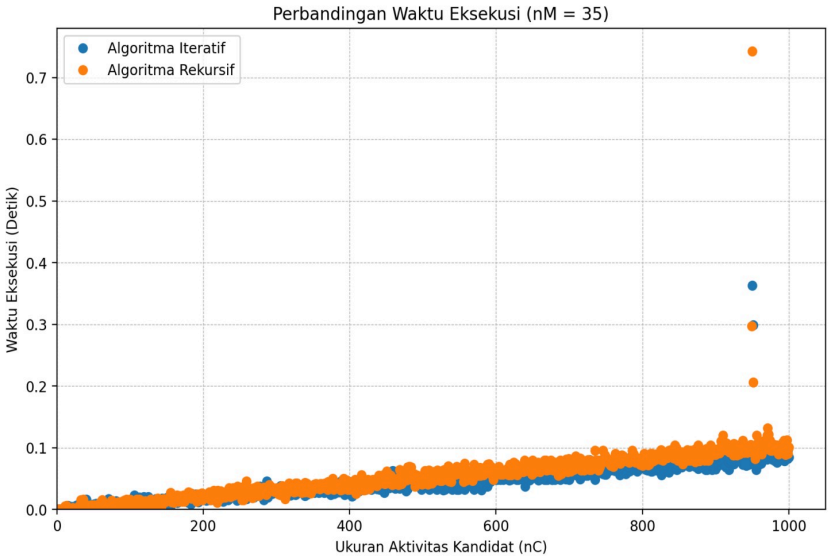
Hasil simulasi menunjukkan bahwa kedua pendekatan menghasilkan daftar aktivitas bebas konflik yang ekivalen, sehingga membuktikan kebenaran implementasi algoritma. Namun, pendekatan iteratif memiliki waktu eksekusi yang lebih cepat dibandingkan pendekatan rekursif, terutama pada ukuran input yang besar. Perbedaan performa ini disebabkan oleh *overhead* pemanggilan fungsi berulang pada pendekatan rekursif.

Secara keseluruhan, implementasi ini membuktikan bahwa algoritma *Non-Overlapping Time Intervals for Scheduling* dapat diterapkan secara efektif dalam perangkat lunak berbasis web.

4.7. Perbandingan Kinerja Algoritma antara Pendekatan Iteratif dan Rekursif Berdasarkan Kompleksitas Waktu Teoritis dan Running Time yang Dihasilkan

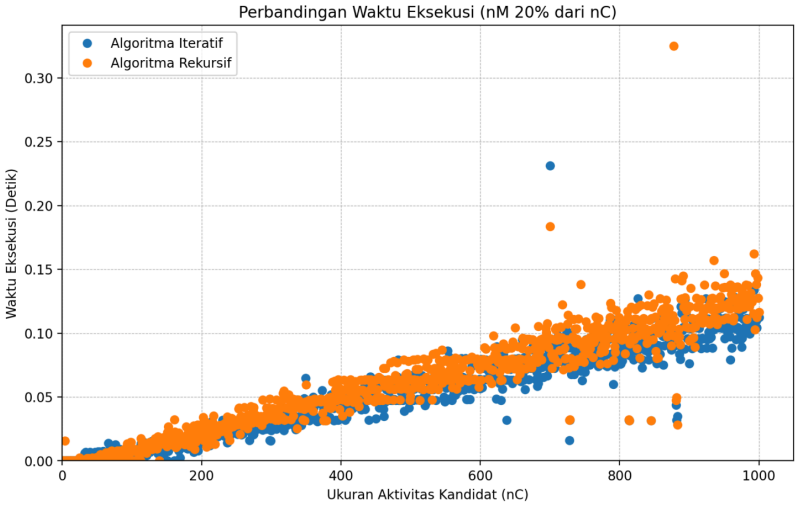
4.7.1. Analisis Berdasarkan *Running Time* (Simulasi Empiris)

4.7.1.1. Skenario *nM* konstan



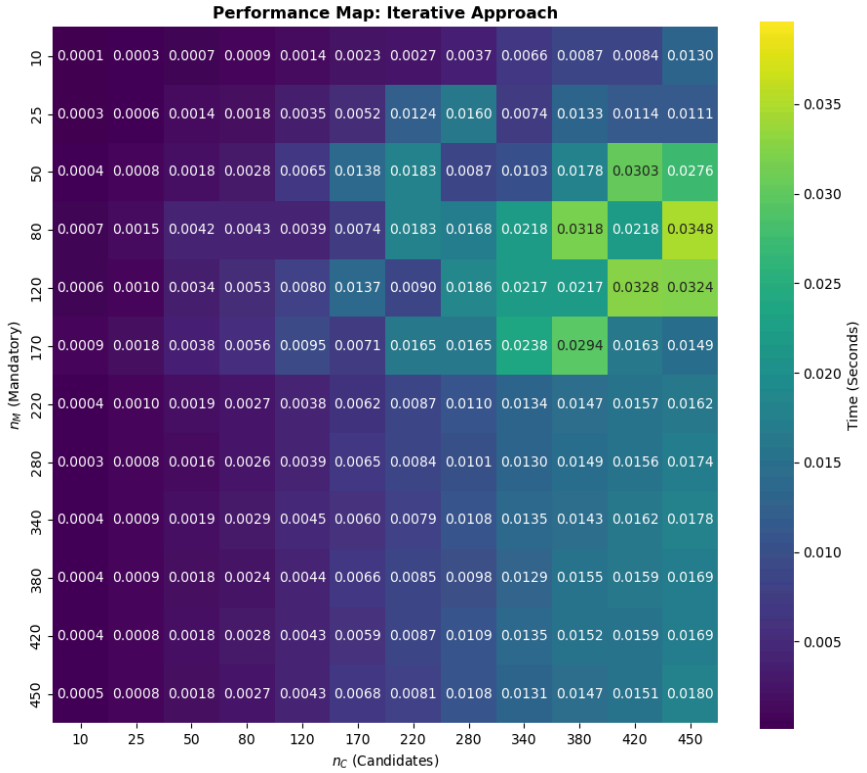
Grafik di atas menunjukkan bahwa baik algoritma iteratif maupun rekursif memiliki tren pertumbuhan waktu yang linear. Secara matematis, hal ini terjadi karena fungsi waktu eksekusi T(nC, nM) = 35 * nC. Karena 35 adalah konstanta, variabel yang mempengaruhi waktu hanyalah nC. Meskipun tren keduanya linear, pendekatan iteratif menunjukkan dominasi performa yang jelas dengan posisi titik data yang secara konsisten berada di bawah titik data rekursif. Hal ini membuktikan bahwa dalam menangani beban kerja yang bersifat repetitif pada satu variabel, mekanisme perulangan pada iteratif jauh lebih efisien dalam mengeksekusi instruksi dibandingkan rekursi.

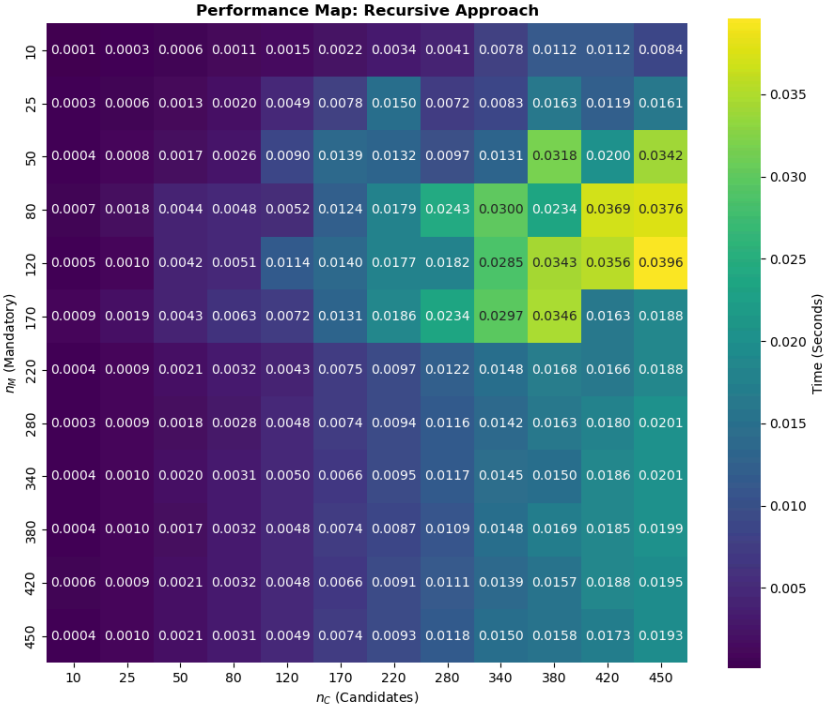
4.7.1.2. Skenario *nM* dinamis (nM = 20% * nC)



Skenario ini merupakan pengujian yang lebih berat karena kedua parameter *input size* tumbuh bersamaan secara proporsional. Berbeda dengan skenario pertama, grafik waktu eksekusi menunjukkan kecenderungan kurva yang lebih curam dibandingkan skenario pertama. Hal ini terjadi karena setiap penambahan satu unit *nC* juga diikuti penambahan unit pada *nM*, yang di mana *nM* selalu 20% dari *nC*. Penambahan beban pada kedua dimensi input inilah yang menyebabkan visualisasi grafik membentuk pola pertumbuhan curam dan bahkan bisa terlihat kuadratik jika pengaturan *nM* dan *nC* lebih dioptimalkan, sesuai analisis kompleksitas secara teoritis yang dilakukan di bagian atas. Dalam kondisi beban kerja yang meningkat secara kuadratik ini, pendekatan iteratif menunjukkan stabilitas performa yang jauh lebih tinggi. Pada grafik rekursif, terlihat adanya fluktuasi atau lonjakan waktu eksekusi yang lebih ekstrem seiring bertambahnya data. Lonjakan ini mengonfirmasi adanya beban *overhead* manajemen memori pada *call stack* milik Python yang semakin berat saat menangani kedalaman rekursi yang besar. Sebaliknya, pendekatan iteratif tetap konsisten pada jalurnya, membuktikan bahwa efisiensi penggunaan memori pada metode perulangan memberikan keunggulan performa yang nyata dalam menangani kompleksitas data yang tinggi.

4.7.1.3. Skenario Perspektif 3D dalam Format 2D Visualisasi Heatmap





```python
# raw analytic by us :)

# ratio > 1 == recursive is slower
ratio_matrix = results_recur / results_iter
diff_matrix = results_recur - results_iter


total_scenarios = results_iter.size
recursive_slower_count = np.sum(results_recur > results_iter)
avg_ratio = np.mean(ratio_matrix)
max_ratio = np.nanmax(ratio_matrix)
min_ratio = np.nanmin(ratio_matrix)

# output insights
print("="*66)
print("HASIL ANALISIS KOMPARATIF: ITERATIF VS REKURSIF (Based on Heatmap)")
print("="*66)

print(f"Total Skenario Pengujian: {total_scenarios} Skenario atau Sel")
print(f"Jumlah Sel Rekursif > Iteratif      : {recursive_slower_count} dari {total_scenarios} sel")
print(f"Persentase Dominasi Efisiensi      : {(recursive_slower_count/total_scenarios)*100:.1f}%")
print("-" * 66)
print(f"Rata-rata Faktor Overhead Rekursif : {avg_ratio:.2f}x lebih lambat")
print(f"Rentang Overhead                   : {min_ratio:.2f}x s.d. {max_ratio:.2f}x")
print("="*66)

if avg_ratio > 1:
    print(f"Pendekatan rekursif secara konsisten menunjukkan waktu eksekusi yang lebih tinggi.")
```

```
==================================================================
HASIL ANALISIS KOMPARATIF: ITERATIF VS REKURSIF (Based on Heatmap)
==================================================================
Total Skenario Pengujian: 144 Skenario atau Sel
Jumlah Sel Rekursif > Iteratif      : 125 dari 144 sel
Persentase Dominasi Efisiensi      : 86.8%
------------------------------------------------------------------
Rata-rata Faktor Overhead Rekursif : 1.14x lebih lambat
Rentang Overhead                   : 0.45x s.d. 1.97x
==================================================================
Pendekatan rekursif secara konsisten menunjukkan waktu eksekusi yang lebih tinggi.
```

4.7.2. Perbandingan Efisiensi

Meskipun secara perhitungan teoritis kedua pendekatan berada pada kelas yang sama untuk tiap kondisinya. Hasil simulasi empiris menunjukkan perbedaan performa yang konsisten, yakni algoritma rekursif terlihat lebih lambat. Hal ini merepresentasikan *overhead* pemanggilan fungsi secara berulang yang membebani memori sistem. Sedangkan, algoritma iteratif secara konsisten berada di bawah garis rekursif, menandakan waktu eksekusi yang lebih cepat. Hal ini disebabkan karena iterasi bekerja dalam satu *stack frame* yang efisien di Python.

BAB V

PENUTUP

5.1. Kesimpulan

Berdasarkan seluruh rangkaian analisis kompleksitas dan pengujian performa yang telah dilakukan, dapat disimpulkan bahwa algoritma *Non-Overlapping Time Intervals for Scheduling* memiliki ketergantungan utama pada dua parameter *input size*, yaitu jumlah Jadwal Kandidat *nC* dan jumlah Jadwal Wajib *nM*. Operasi dasar yang menjadi unit perhitungan utama dalam sistem ini adalah operasi perbandingan waktu yang memvalidasi ada atau tidaknya tumpang tindih antara jadwal baru dengan jadwal yang sudah terfiksasi. Secara teoritis, baik pendekatan iteratif maupun rekursif berada pada kelas kompleksitas yang sama, yakni $\Theta(nC)$ untuk kasus terbaik dan $\Theta(nC*nM)$ untuk kasus terburuk serta rata-rata. Hal ini menunjukkan bahwa peningkatan beban kerja akan berbanding lurus dengan hasil perkalian dari kedua dimensi input tersebut.

Hasil pengujian empiris melalui perangkat lunak simulasi memvalidasi teori tersebut, di mana grafik waktu eksekusi menunjukkan tren linear yang sangat baik ketika salah satu variabel bersifat konstan dan berubah menjadi tren yang kuadratik (jika pengaturan *input size* lebih dioptimalkan) ketika kedua variabel tumbuh secara proporsional. Namun, dalam perbandingan kinerja secara langsung, ditemukan bahwa pendekatan iteratif memiliki efisiensi yang lebih tinggi dibandingkan pendekatan rekursif. Keunggulan iteratif terlihat dari waktu eksekusi yang lebih cepat dan grafik yang lebih stabil, sementara pendekatan rekursif cenderung lebih lambat akibat adanya beban tambahan pada manajemen memori *stack frame* . Selain itu, perspektif melalui visualisasi *heatmap* menegaskan, untuk kebutuhan sistem penjadwalan dengan volume area data yang besar, penggunaan pendekatan iteratif cenderung lebih baik secara konsisten dibandingkan dengan pendekatan rekursif..

REFERENSI

GeekforGeeks. (2025, August 7). *Non-Overlapping Intervals*. GeeksforGeeks. Retrieved December 29, 2025, from https://www.geeksforgeeks.org/dsa/minimum-removals-required-to-make-ranges-non-overlapping/

Mathplotlib Development Team. (2025). *Matplotlib documentation — Matplotlib 3.10.8 documentation*. Matplotlib. Retrieved December 29, 2025, from https://matplotlib.org/stable/index.html

Pandas Development Team. (2025, September 29). *pandas documentation — pandas 2.3.3 documentation*. Pandas. Retrieved December 29, 2025, from https://pandas.pydata.org/docs/

Python Software Foundation. (2025). *Python 3 documentation*. 3.14.2 Documentation. Retrieved December 29, 2025, from https://docs.python.org/3/

Streamlit. (2025). Streamlit documentation. Retrieved December 29, 2025, from https://docs.streamlit.io/

LAMPIRAN

- Repositori Github: https://github.com/wibotron130823/conflict-free-scheduler-app-and-analysis
- Tautan Slides: https://docs.google.com/presentation/d/1AXrGWAm0wjUT1PJWkrTpkg5hk1XjBgUfEz5hgNAh52g/edit?usp=sharing