LAPORAN TUGAS

PROGRAM STUDI S1 MATEMATIKA

Struktur Data Dan Algoritma

4420102131

Oleh:

1. Ayu Setya Permatasari (23030214008)
2. Ermelya Helmi Anggraeni A (23030214024)
3. Vita Chamelia Sofiatul Ula (23030214055)
4. Nabilah Nesya Adiarni Azzahra (23030214098)
5. Algian Jawwad Faras Rofni (23030214180)

MA - 2023F

Dosen Pengampu:
Riska Wahyu ROMADHONIA, S.Si., M.Sc.

Ko-Asisten:
Farhan Nurrahman

10 Juni 2025

1 Permasalahan

Mahasiswa sering kali dihadapi pada banyak kegiatan dalam waktu yang bersamaan. Selain mengikuti kuliah dan menyelesaikan tugas, mahasiswa juga aktif dalam organisasi, mengikuti perlombaan, hingga menjalani magang. Semua kegiatan ini tertentu membutuhkan pengaturan waktu yang efesien agar tidak saling bertabrakan. Tanpa manajemen waktu yang tepat, mahasiswa kurang efesien dalam mengatur jadwal. Karena kegiatan maupun tugas sering terjadi bersamaan, sehingga butuh kemampuan untuk membagi waktu dan menentukan mana yang harus didahulukan.

Banyaknya mahasiswa yang masih mencatat jadwal atau tugas-tugas mereka secara manual, baik di buku catatan maupun aplikasi yang tidak terorganisir. Cara seperti ini sangat rawan lupa, tidak efesien dan kurang mendukung dalam mengatur prioritas. Hal ini bisa berdampak terjadinya keterlambatan pengumpulan tugas, lupa jadwal penting. Taylor,H.L(1990) Manajemen waktu adalah proses perencanaan dan pengendalian waktu yang digunakan untuk aktivitas tertentu guna meningkatkan efesien dan produktivitas. Untuk mengatasi permaslahan tersebut, kami mengembangkan sebuah aplikasi berbasis Graphical User Interface (GUI) yang diberi nama PriotizeMe.

Aplikasi ini dirancang khusus sebagai alat bantu untuk mencatat agenda menggunakan sistem to-do list yang terorganisir. Aplikasi ini dirancang agar tampilannya mudah dipahami dan digunakan, bahkan oleh orang yang belum terbiasa memakai aplikasi serupa. Aplikasi ini dibuat dengan tampilan GUI yang sederhana dan menarik. Salah satu keunggulan dari aplikasi PrioritizeMe adalah fitur pengelompokkan tugas berdasarkan prioritas dan pengingat otomatis sebelum deadline.

Fitur ini membantu mahasiswa menyelesaikan tugas-tugas penting lebih dulu dan tidak melewatkan deadline. Selain itu PrioritizeMe juga mendukung penggabungan antara kegiatan akademik dan non-akademik dalam satu sistem, sehingga pengguna tidak perlu lagi mencatat jadwal terlalu banyak tempat. Misalnya, mahasiswa bisa mencatat jadwal kuliah, tugas, rapat organisasi. Aplikasi ini juga diharapkan dapat membentuk kebiasaan manajemen waktu yang baik, seperti menyusun rencana harian, mengecek tugas-tugas yang harus dikerjakan.

Jika kebiasaan ini dilakukan terus-menerus, mahasiswa tidak hanya akan terbantu selama kuliah, tetapi juga saat masuk kedunia kerja nanti, dimana keterampilan mengatur waktu menjadi sangat penting. Secara keseluruhan PrioritizeMe hadir sebagai solusi praktis untuk mahasiswa dalam menghadapi kesibukan sehari-hari. Dengan fitur-fitur yang mendukung keteraturan, kedislinan, dan efesien, aplikasi ini diharapakan bisa mengingkatkan produktivitas mahasiswa dan membantu merekan menjalani kehidupan kampus dengan lebih teratur.

2 *Source Code*

Tuliskan *source code* disini. Berikut adalah contoh *source code*:

```
import tkinter as tk
from tkinter import ttk, messagebox
from datetime import datetime, timedelta
from tkcalendar import DateEntry
from collections import deque
import datetime
import time
from PIL import Image, ImageTk
import heapq
import itertools



tugas_aktif = []
tugas_selesai = []

prioritas_level = {
    "Tinggi": 1,
    "Sedang": 2,
    "Rendah": 3
}

entry_finder = itertools.count()


# Define Task Page dan Buat GUI
class TaskPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.configure(padx=10, pady=10)
        self.create_widgets()

    def create_widgets(self):
        btn_home = tk.Button(self, text="    Beranda",
        bg="#2196F3", fg="white", command=lambda: self.controller.
           show_frame("HomePage"))
        btn_home.pack(anchor='w')

        main_frame = tk.Frame(self)
        main_frame.pack(fill='both', expand=True)


        # membuat frame di sebelah kiri
        frame_kiri = tk.Frame(main_frame, padx=10, pady=10)
        frame_kiri.pack(side='left', fill='y')
```

```
tk.Label(frame_kiri, text="Nama Tugas").pack()
self.entry_nama = tk.Entry(frame_kiri, width=30)
self.entry_nama.pack()

tk.Label(frame_kiri, text="Deskripsi").pack()
self.entry_deskripsi = tk.Entry(frame_kiri, width=30)
self.entry_deskripsi.pack()

tk.Label(frame_kiri, text="Deadline (Tanggal):").pack()
self.entry_deadline = DateEntry(frame_kiri, width=27,
date_pattern='yyyy-mm-dd')
self.entry_deadline.pack()

tk.Label(frame_kiri, text="Jam (24h):").pack()
jam_values = [f"{i:02}" for i in range(24)]
self.jam_combobox = ttk.Combobox(frame_kiri,
values=jam_values, width=5, state="readonly", justify="
   center")
self.jam_combobox.set("00")
self.jam_combobox.pack()

tk.Label(frame_kiri, text="Menit:").pack()
menit_values = [f"{i:02}" for i in range(60)]
self.menit_combobox = ttk.Combobox(frame_kiri,
values=menit_values, width=5, state="readonly", justify="
   center")
self.menit_combobox.set("00")
self.menit_combobox.pack()

tk.Label(frame_kiri, text="Prioritas").pack()
self.combo_prioritas = ttk.Combobox(frame_kiri,
values=["Tinggi", "Sedang", "Rendah"], state="readonly")
self.combo_prioritas.pack()
self.combo_prioritas.set("Sedang")

tk.Label(frame_kiri, text="Kategori").pack()
self.combo_kategori = ttk.Combobox(frame_kiri,
values=["Analisis Real 2", "Metode Numerik", "Pemodelan", "
   PDP",
"Teori Graf", "Aljabar Abstrak 1","Organisasi", "Akademik",
"Pribadi", "Lainnya"], state="readonly")
self.combo_kategori.pack()
self.combo_kategori.set("Lainnya")

btn_tambah = tk.Button(frame_kiri, text="Tambah Tugas",
width=30, bg="#4CAF50", fg="white",
    command=lambda: tambah_tugas(
        self.entry_nama.get(),
        self.entry_deskripsi.get(),
```

```
            f"{self.entry_deadline.get_date().strftime('%Y-%m-%
               d')} {self.jam_combobox.get()}:{self.
               menit_combobox.get()}",
            self.combo_prioritas.get(),
            self.combo_kategori.get(),
            self.tree_tugas,
            self.entry_nama, self.entry_deskripsi, self.
               combo_prioritas, self.entry_deadline, self.
               jam_combobox, self.menit_combobox))
btn_tambah.pack(pady=5)

btn_edit = tk.Button(frame_kiri, text="Edit Tugas", width
   =30, bg="#FFC107", fg="black",
    command=lambda: edit_tugas(self.tree_tugas, self.
       controller.search_var, self.controller.sort_var))
btn_edit.pack(pady=5)

btn_selesai = tk.Button(frame_kiri, text="Tandai Selesai >>
   ", width=30, bg="#2196F3", fg="white",
    command=lambda: tandai_selesai(self.tree_tugas, self.
       tree_selesai,
    self.controller.search_var, self.controller.sort_var))
btn_selesai.pack(pady=5)

overdue_frame = tk.Frame(frame_kiri, padx=10, pady=10, bg="
   #FFCCCC", borderwidth=2, relief="groove")
overdue_frame.pack(pady=(10, 20), fill='x')
overdue_label1 = tk.Label(overdue_frame, text="* Baris yang
    berlabel merah adalah",
font=("Arial", 10), fg="red", bg="#FFCCCC")
overdue_label1.pack(pady=0)
overdue_label2 = tk.Label(overdue_frame, text="tugas yang
   deadline terlambat",
font=("Arial", 10), fg="red", bg="#FFCCCC")
overdue_label2.pack(pady=5)
overdue_label3 = tk.Label(overdue_frame, text="dan belum
   diselesaikan.",
font=("Arial", 10), fg="red", bg="#FFCCCC")
overdue_label3.pack(pady=5)


# membuat frame di sebelah kanan
frame_kanan = tk.Frame(main_frame, padx=10, pady=10)
frame_kanan.pack(side='right', fill='both', expand=True)

filter_frame = tk.Frame(frame_kanan)
filter_frame.pack(fill='x')


# Search tugas
```

```
tk.Label(filter_frame, text="Cari Tugas:").pack(side='left'
   )
self.entry_search = tk.Entry(filter_frame, textvariable=
   self.controller.search_var, width=20)
self.entry_search.pack(side='left', padx=5)
self.entry_search.bind("<KeyRelease>", lambda e: self.
   update_treeviews())

tk.Label(filter_frame, text="Sortir:").pack(side='left')
sort_cb = ttk.Combobox(filter_frame, values=["Nama Tugas",
"Deadline", "Prioritas",], textvariable=self.controller.
   sort_var, state="readonly")
sort_cb.pack(side='left', padx=5)
sort_cb.bind("<<ComboboxSelected>>", lambda e: self.
   update_treeviews())

frame_tree_tugas = tk.Frame(frame_kanan)
frame_tree_tugas.pack(fill='both', expand=True, pady=5)

tk.Label(frame_tree_tugas, text="Daftar Tugas Aktif", font
   =("Arial", 12, "bold")).pack(pady=(0, 5))
self.tree_tugas = ttk.Treeview(frame_tree_tugas,
columns=("Nama Tugas", "Deskripsi", "Deadline", "Prioritas"
   , "Kategori"), show='headings')
for col in self.tree_tugas["columns"]:
    self.tree_tugas.heading(col, text=col)
    self.tree_tugas.column(col, width=200)
self.tree_tugas.pack(side='left', fill='both', expand=True)
self.tree_tugas.bind("<Double-1>", self.
   on_tugas_double_click)

scrollbar_tugas = ttk.Scrollbar(frame_tree_tugas, orient='
   vertical', command=self.tree_tugas.yview)
scrollbar_tugas.pack(side='right', fill='y')
self.tree_tugas.configure(yscrollcommand=scrollbar_tugas.
   set)

tk.Label(frame_kanan, text="Daftar Tugas Selesai", font=("
   Arial", 12, "bold")).pack(pady=(10,0))

frame_tree_selesai = tk.Frame(frame_kanan)
frame_tree_selesai.pack(fill='both', expand=True, pady=5)

self.tree_selesai = ttk.Treeview(frame_tree_selesai,
   columns=("Nama Tugas",
"Deskripsi", "Deadline", "Prioritas", "Kategori"), show='
   headings')
for col in self.tree_selesai["columns"]:
    self.tree_selesai.heading(col, text=col)
    self.tree_selesai.column(col, width=200)
```

```
    self.tree_selesai.pack(side='left', fill='both', expand=
       True)
    self.tree_selesai.bind("<Double-1>", self.
       on_selesai_double_click)

    scrollbar_selesai = ttk.Scrollbar(frame_tree_selesai,
       orient='vertical', command=self.tree_selesai.yview)
    scrollbar_selesai.pack(side='right', fill='y')
    self.tree_selesai.configure(yscrollcommand=
       scrollbar_selesai.set)

    global tree_tugas, tree_selesai
    tree_tugas = self.tree_tugas
    tree_selesai = self.tree_selesai

    update_treeview(self.tree_tugas, self.controller.search_var
       .get(), self.controller.sort_var.get())
    update_treeview(self.tree_selesai, tugas_list=tugas_selesai
       )
    cek_pengingat(self.tree_tugas, self.controller.search_var,
       self.controller.sort_var, self.controller)


def on_tugas_double_click(self, event):
    """Menangani double-klik pada daftar tugas aktif."""
    selected_item_id = self.tree_tugas.selection()
    if selected_item_id:
        item_values = self.tree_tugas.item(selected_item_id[0],
            'values')

        tugas_ditemukan = None
        for prio_num, dline_obj, count, tugas_dict in
           tugas_aktif:
            if tugas_dict["nama"] == item_values[0] and
               tugas_dict["deadline"] == item_values[2]:
                tugas_ditemukan = tugas_dict
                break

        if tugas_ditemukan:
            lihat_detail_tugas(tugas_ditemukan)
        else:
            messagebox.showwarning("Error", "Tugas tidak
               ditemukan dalam daftar aktif.")

def on_selesai_double_click(self, event):
    """Menangani double-klik pada daftar tugas selesai."""
    selected_item_id = self.tree_selesai.selection()
    if selected_item_id:
        item_values = self.tree_selesai.item(selected_item_id
           [0], 'values')
```

```
            tugas_ditemukan = None
            for t in tugas_selesai:
                if t["nama"] == item_values[0] and t["deadline"] ==
                    item_values[2]:
                    tugas_ditemukan = t
                    break

            if tugas_ditemukan:
                lihat_detail_tugas(tugas_ditemukan)
            else:
                messagebox.showwarning("Error", "Tugas tidak
                   ditemukan dalam daftar selesai.")


    def update_treeviews(self):
        """Memperbarui kedua treeview (aktif dan selesai)
           berdasarkan filter pencarian dan opsi sortir."""
        update_treeview(self.tree_tugas, self.controller.search_var
           .get(), self.controller.sort_var.get())
        update_treeview(self.tree_selesai, self.controller.
           search_var.get(), tugas_list=tugas_selesai)



# Fungsi notifikasi berwarna
def tampilkan_notifikasi_berwarna(judul, pesan, warna):
    notif = tk.Toplevel()
    notif.title(judul)
    notif.configure(bg=warna)
    notif.geometry("400x150")
    notif.resizable(False, False)

    label = tk.Label(notif, text=pesan, bg=warna, fg="white",
    font=("Arial", 12, "bold"), wraplength=380, justify="center")
    label.pack(expand=True, padx=10, pady=10)

    btn = tk.Button(notif, text="Tutup", command=notif.destroy, bg=
       "white")
    btn.pack(pady=10)

    notif.grab_set()

# Fungsi masukkan tugas ke tugas aktif secara terurut queue
def masukkan_tugas_terurut(queue, tugas, sort_by="Deadline"):
    temp = deque()
    masuk = False
    while queue:
        tugas_lama = queue.popleft()
        if not masuk:
```

```
            if sort_by == "Deadline" and tugas["deadline"] <
               tugas_lama["deadline"]:
                temp.append(tugas)
                masuk = True
            elif sort_by == "Prioritas" and tugas["prioritas"] <
               tugas_lama["prioritas"]:
                temp.append(tugas)
                masuk = True
        temp.append(tugas_lama)
    if not masuk:
        temp.append(tugas)
    queue.extend(temp)

# Tambahkan fungsi ini di atas fungsi tambah_tugas

def masukkan_tugas_terurut_prioritas(list_tugas, tugas_baru):
    """
    Menyisipkan tugas_baru ke dalam list_tugas (bisa deque atau
       list biasa)
    secara terurut berdasarkan prioritas (Tinggi, Sedang, Rendah).
    Jika prioritas sama, yang lebih baru diletakkan setelah yang
       sudah ada.
    """

    # Pastikan list_tugas adalah deque atau list yang bisa di-
       insert
    is_deque = isinstance(list_tugas, deque)
    temp_list = list(list_tugas) if is_deque else list_tugas #
       Konversi ke list untuk insert

    prioritas_baru = prioritas_level.get(tugas_baru["prioritas"])

    inserted = False
    for i, tugas_lama in enumerate(temp_list):
        prioritas_lama = prioritas_level.get(tugas_lama["prioritas"
           ])

        if prioritas_baru < prioritas_lama: # Prioritas baru lebih
           tinggi (nilai lebih kecil)
            temp_list.insert(i, tugas_baru)
            inserted = True
            break

    if not inserted: # Jika belum disisipkan (paling rendah
       prioritasnya atau list kosong)
        temp_list.append(tugas_baru)

    # Kosongkan deque asli dan isi ulang dari temp_list yang sudah
       terurut
    if is_deque:
```

```
            list_tugas.clear()
            list_tugas.extend(temp_list)
        else: # Jika list_tugas adalah list biasa (seperti
           tugas_selesai)
            list_tugas[:] = temp_list # Update list in-place

def tambah_tugas(nama, deskripsi, deadline, prioritas, kategori,
    tree, entry_nama, entry_deskripsi, combo_prioritas,
       entry_deadline, jam_combobox, menit_combobox):
    if not nama or not deadline or not prioritas or not kategori:
        messagebox.showwarning("Peringatan", "Nama, deadline,
           prioritas, dan kategori harus diisi.")
        return

    deskripsi_input = deskripsi
    if len(deskripsi_input) > 500: # Contoh batasan 500 karakter
        messagebox.showwarning("Peringatan", "Deskripsi terlalu
           panjang (maks. 500 karakter).")
        return

    try:
        deadline_dt = datetime.datetime.strptime(deadline, '%Y-%m-%
           d %H:%M')
    except ValueError:
        messagebox.showwarning("Peringatan", "Format deadline tidak
            valid. Gunakan YYYY-MM-DD HH:MM.")
        return

    tugas = {
        "nama": nama,
        "deskripsi": deskripsi_input,
        "deadline": deadline,  # string representation
        "deadline_obj": deadline_dt, # <--- Tambahkan objek
           datetime untuk perbandingan
        "prioritas": prioritas,
        "kategori": kategori,
        "diingatkan": False,
        "terlambat": False
    }

    now = datetime.datetime.now()
    if deadline_dt < now:
        tugas["terlambat"] = True
        # Tetap gunakan masukkan_tugas_terurut_prioritas untuk
           tugas_selesai jika Anda ingin urut
        masukkan_tugas_terurut_prioritas(tugas_selesai, tugas)
        update_treeview(tree_selesai, tugas_list=tugas_selesai)
        messagebox.showinfo("Sukses", "Tugas berhasil ditambahkan
           dan sudah melewati deadline, langsung dipindahkan ke
           daftar selesai!")
```

```
    else:
        # PUSH TUGAS KE HEAP BERDASARKAN PRIORITAS DAN DEADLINE
        # Struktur tuple: (prioritas_numerik, deadline_obj,
           counter_unik, tugas_dict)
        prioritas_num = prioritas_level.get(prioritas)
        count = next(entry_finder) # Dapatkan nilai counter unik

        # Simpan tuple ini ke dalam heap
        heapq.heappush(tugas_aktif, (prioritas_num, deadline_dt,
           count, tugas)) # <--- PUSH KE HEAP

        update_treeview(tree) # Update tree_tugas
        messagebox.showinfo("Sukses", "Tugas berhasil ditambahkan!"
           )

    # Reset input fields
    entry_nama.delete(0, tk.END)
    entry_deskripsi.delete(0, tk.END)
    combo_prioritas.set("Sedang")
    entry_deadline.set_date(datetime.datetime.now())
    jam_combobox.set("00")
    menit_combobox.set("00")


def edit_tugas(tree, search_var, sort_var):
    selected_item_id = tree.selection() # Perbaikan: Gunakan
       selected_item_id
    if not selected_item_id:
        messagebox.showwarning("Peringatan", "Pilih tugas yang
           ingin diedit.")
        return

    # Dapatkan tugas asli dari tugas_aktif berdasarkan item yang
       dipilih di Treeview
    # Karena tugas_aktif sekarang adalah heap dari tuple, kita
       perlu mencari dictionary tugasnya
    item_values = tree.item(selected_item_id[0], 'values')
    tugas_lama_untuk_edit = None
    for prio, dline_obj, count, t in tugas_aktif: # Iterate over
       the heap tuples
        if t["nama"] == item_values[0] and t["deadline"] ==
           item_values[2]:
            tugas_lama_untuk_edit = t # Dapatkan dictionary tugas
               aslinya
            break

    if not tugas_lama_untuk_edit:
        messagebox.showwarning("Error", "Tugas tidak ditemukan
           untuk diedit.")
        return
```

```
tugas_sementara = dict(tugas_lama_untuk_edit) # Buat salinan
   dictionary

def simpan_perubahan():
    tanggal = entry_deadline.get_date()
    jam = combo_jam.get()
    menit = combo_menit.get()

    try:
        waktu_str = f"{jam}:{menit}"
        deadline_str = f"{tanggal.strftime('%Y-%m-%d')} {
           waktu_str}"
        new_deadline_dt = datetime.datetime.strptime(
           deadline_str, '%Y-%m-%d %H:%M')
    except Exception as e:
        messagebox.showerror("Error", f"Format waktu tidak
           valid: {e}")
        return

    new_nama = entry_nama.get()
    new_deskripsi = entry_deskripsi.get()
    new_prioritas = combo_prioritas.get()
    new_kategori = combo_kategori.get()

    # Validasi nama dan prioritas baru
    if not new_nama or not new_prioritas or not new_kategori:
        messagebox.showwarning("Peringatan", "Nama, prioritas,
           dan kategori harus diisi.")
        return

    # Buat tugas baru dengan data yang diupdate
    tugas_baru_setelah_edit = {
        "nama": new_nama,
        "deskripsi": new_deskripsi,
        "deadline": deadline_str,
        "deadline_obj": new_deadline_dt, # Pastikan ini
           diupdate
        "prioritas": new_prioritas,
        "kategori": new_kategori,
        "diingatkan": False, # Reset flag pengingat setelah
           edit
        "terlambat": False # Reset terlambat status
    }

    temp_heap_list = []
    found_and_removed = False
    for item_tuple in tugas_aktif:
        if item_tuple[3] is not tugas_lama_untuk_edit: # Jika
           bukan tugas yang sedang diedit
```

```
                temp_heap_list.append(item_tuple)
            else:
                found_and_removed = True

        if found_and_removed:
            tugas_aktif[:] = [] # Kosongkan heap asli
            heapq.heapify(temp_heap_list) # Bangun ulang heap
            tugas_aktif.extend(temp_heap_list) # Isi kembali
               tugas_aktif

        # Tambahkan tugas yang diedit kembali ke heap (jika tidak
           terlambat)
        now_check = datetime.datetime.now()
        if new_deadline_dt < now_check:
            tugas_baru_setelah_edit["terlambat"] = True
            masukkan_tugas_terurut_prioritas(tugas_selesai,
               tugas_baru_setelah_edit)
            messagebox.showinfo("Sukses", "Tugas berhasil diedit
               dan sudah melewati deadline, langsung dipindahkan ke
                daftar selesai!")
        else:
            prioritas_num = prioritas_level.get(new_prioritas)
            count = next(entry_finder)
            heapq.heappush(tugas_aktif, (prioritas_num,
               new_deadline_dt, count, tugas_baru_setelah_edit))
            messagebox.showinfo("Sukses", "Tugas berhasil diedit!")

        update_treeview(tree, search_var.get(), sort_var.get())
        update_treeview(tree_selesai, tugas_list=tugas_selesai) #
           Pastikan juga tugas_selesai terupdate
        popup.destroy()

    popup = tk.Toplevel()
    popup.title("Edit Tugas")
    popup.geometry("400x500")


    tk.Label(popup, text="Nama:").grid(row=0, column=0, sticky='w')
    entry_nama = tk.Entry(popup)
    entry_nama.insert(0, tugas_lama_untuk_edit["nama"])
    entry_nama.grid(row=0, column=1)

    tk.Label(popup, text="Deskripsi:").grid(row=1, column=0, sticky
       ='w')
    entry_deskripsi = tk.Entry(popup)
    entry_deskripsi.insert(0, tugas_lama_untuk_edit["deskripsi"])
    entry_deskripsi.grid(row=1, column=1)

    tk.Label(popup, text="Deadline (Tanggal):").grid(row=2, column
       =0, sticky='w')
```

```
entry_deadline = DateEntry(popup, date_pattern='yyyy-mm-dd')
try:
    dt = datetime.datetime.strptime(tugas_lama_untuk_edit["
       deadline"], '%Y-%m-%d %H:%M')
    entry_deadline.set_date(dt)
except Exception:
    pass
entry_deadline.grid(row=2, column=1)

tk.Label(popup, text="Jam:").grid(row=3, column=0, sticky='w')
jam_values = [f"{i:02}" for i in range(24)]
combo_jam = ttk.Combobox(popup, values=jam_values, width=5,
   state="readonly", justify="center")
try:
    dt = datetime.datetime.strptime(tugas_lama_untuk_edit["
       deadline"], '%Y-%m-%d %H:%M')
    combo_jam.set(f"{dt.hour:02d}")
except Exception:
    combo_jam.set("00")
combo_jam.grid(row=3, column=1, sticky='w')

tk.Label(popup, text="Menit:").grid(row=4, column=0, sticky='w'
   )
menit_values = [f"{i:02}" for i in range(60)]
combo_menit = ttk.Combobox(popup, values=menit_values, width=5,
    state="readonly", justify="center")
try:
    dt = datetime.datetime.strptime(tugas_lama_untuk_edit["
       deadline"], '%Y-%m-%d %H:%M')
    combo_menit.set(f"{dt.minute:02d}")
except Exception:
    combo_menit.set("00")
combo_menit.grid(row=4, column=1, sticky='w')

tk.Label(popup, text="Prioritas:").grid(row=5, column=0, sticky
   ='w')
combo_prioritas = ttk.Combobox(popup, values=["Tinggi", "Sedang
   ", "Rendah"], state="readonly")
combo_prioritas.set(tugas_lama_untuk_edit["prioritas"])
combo_prioritas.grid(row=5, column=1)

tk.Label(popup, text="Kategori:").grid(row=6, column=0, sticky=
   'w')
combo_kategori = ttk.Combobox(popup, values=["Analisis Real 2",
    "Metode Numerik",
"Pemodelan", "PDP", "Teori Graf", "Aljabar Abstrak 1","
   Organisasi", "Akademik",
"Pribadi", "Lainnya"], state="readonly")
combo_kategori.set(tugas_lama_untuk_edit["kategori"])
combo_kategori.grid(row=6, column=1)
```

```
    btn_simpan = tk.Button(popup, text="Simpan", command=
       simpan_perubahan)
    btn_simpan.grid(row=7, column=0, columnspan=2, pady=10)


def tandai_selesai(tree_tugas, tree_selesai, search_var, sort_var):
    if not tugas_aktif:
        messagebox.showinfo("Informasi", "Tidak ada tugas aktif
           untuk diselesaikan.")
        return

    priority_num, deadline_dt, count, tugas_yang_akan_diselesaikan
       = heapq.heappop(tugas_aktif)

    masukkan_tugas_terurut_prioritas(tugas_selesai,
       tugas_yang_akan_diselesaikan)

    # Perbarui tampilan GUI
    update_treeview(tree_tugas, search_var.get(), sort_var.get())
    update_treeview(tree_selesai, tugas_list=tugas_selesai)

    messagebox.showinfo("Sukses", f"Tugas '{
       tugas_yang_akan_diselesaikan['nama']}' (Prioritas: {
       tugas_yang_akan_diselesaikan['prioritas']}) berhasil
       ditandai selesai!")


def update_treeview(tree, search_filter="", sort_by="", tugas_list=
   None):
    for item in tree.get_children():
        tree.delete(item)

    display_list = []
    if tugas_list is None:
        # Jika ini tugas_aktif, kita perlu ekstrak tugas dari tuple
            heap
        for item_tuple in tugas_aktif:
            display_list.append(item_tuple[3]) # Ambil dictionary
               tugasnya saja

    else:
        # Jika ini tugas_selesai (sudah diurutkan oleh
           masukkan_tugas_terurut_prioritas)
        display_list = list(tugas_list) # Buat salinan untuk
           filtering/sorting

    search_filter = search_filter.strip().lower()

    # FILTER: Nama, Kategori, dan Deadline
```

```
filtered_tasks = [
    t for t in display_list
    if (search_filter in t["nama"].lower()
        or search_filter in t["kategori"].lower()
        or search_filter in t["deadline"].lower())
]

# SORTING: Terapkan Insertion Sort berdasarkan pilihan user
if sort_by == "Deadline":
    insertion_sort(filtered_tasks, key_func=lambda x: datetime.
       datetime.strptime(x["deadline"], '%Y-%m-%d %H:%M'))
elif sort_by == "Nama Tugas":
    insertion_sort(filtered_tasks, key_func=lambda x: x["nama"
       ].lower())
elif sort_by == "Prioritas":
    # Untuk prioritas, kita urutkan secara ascending
       berdasarkan nilai numerik prioritas (1=Tinggi, 2=Sedang,
        3=Rendah)
    insertion_sort(filtered_tasks, key_func=lambda x:
       prioritas_level.get(x["prioritas"], 99))

MAX_DESKRIPSI_LENGTH = 70

for t in filtered_tasks:
    display_deskripsi = t["deskripsi"]
    if len(display_deskripsi) > MAX_DESKRIPSI_LENGTH:
        display_deskripsi = display_deskripsi[:
           MAX_DESKRIPSI_LENGTH] + "..."

    tags = ()
    if tree == tree_selesai:
        if t.get("terlambat", False):
            tags = ('overdue',)
    else:
        now = datetime.datetime.now()
        deadline_compare = t.get("deadline_obj")
        if not deadline_compare:
            try:
                deadline_compare = datetime.datetime.strptime(t
                   ["deadline"], '%Y-%m-%d %H:%M')
                t["deadline_obj"] = deadline_compare
            except ValueError:
                deadline_compare = None

        if deadline_compare and deadline_compare < now:
            tags = ('overdue',)
        elif t["prioritas"] == "Tinggi":
            tags = ('high_priority',)
        elif t["prioritas"] == "Sedang":
            tags = ('medium_priority',)
```

```
            elif t["prioritas"] == "Rendah":
                tags = ('low_priority',)

        tree.insert('', 'end', values=(t["nama"], display_deskripsi
           , t["deadline"],
        t["prioritas"], t["kategori"]), tags=tags)

    tree.tag_configure('high_priority')
    tree.tag_configure('medium_priority')
    tree.tag_configure('low_priority')
    tree.tag_configure('overdue', background='#F44336', foreground=
       'white')


def cek_pengingat (tree, search_var, sort_var, root):
    now = datetime.datetime.now()
    changed = False

    temp_aktif = []

    for priority_num, deadline_dt, count, tugas in list(tugas_aktif
       ):
        if deadline_dt < now:
            tugas["terlambat"] = True
            masukkan_tugas_terurut_prioritas(tugas_selesai, tugas)
            changed = True
        else:
            temp_aktif.append((priority_num, deadline_dt, count,
               tugas))
            selisih_hari = (deadline_dt.date() - now.date()).days
            if (
                not tugas.get("diingatkan")
                and selisih_hari == 1
            ):
                tugas["diingatkan"] = True # Set flag agar tidak
                   muncul lagi

                # Tentukan warna notifikasi berdasarkan prioritas
                warna = "#7337FFB2" # Default warna biru untuk H-1
                if tugas["prioritas"] == "Tinggi":
                    warna = "#FF0000" # Merah untuk prioritas
                       Tinggi
                elif tugas["prioritas"] == "Sedang":
                    warna = "#FFA500" # Oranye untuk prioritas
                       Sedang
                elif tugas["prioritas"] == "Rendah":
                    warna = "#17CC17" # Hijau untuk prioritas
                       Rendah

                # Tampilkan notifikasi
```

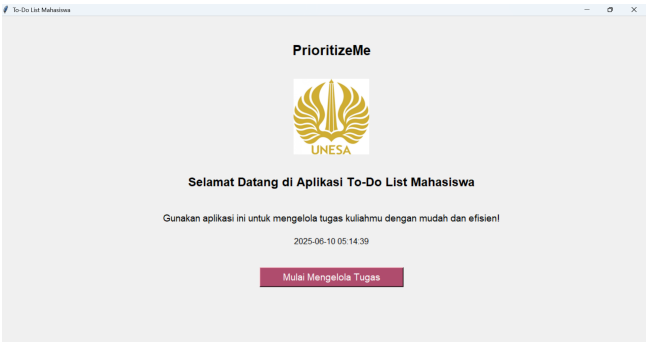
```
                tampilkan_notifikasi_berwarna(
                    "Pengingat Tugas BESOK!",
                    f"Tugas '{tugas['nama']}' ({tugas['prioritas
                        ']}) akan jatuh tempo BESOK pada {
                        deadline_dt.strftime('%Y-%m-%d %H:%M')}!",
                    warna
                )

    tugas_aktif.clear()
    for item in temp_aktif:
        heapq.heappush(tugas_aktif, item)
    if changed:
        update_treeview(tree, search_var.get(), sort_var.get())
        update_treeview(tree_selesai, tugas_list=tugas_selesai)

    root.after(3000, lambda: cek_pengingat(tree, search_var,
       sort_var, root))


def lihat_detail_tugas(tugas):
    """Menampilkan detail lengkap tugas dalam jendela terpisah."""
    detail_window = tk.Toplevel()
    detail_window.title(f"Detail Tugas: {tugas['nama']}")
    detail_window.geometry("300x400")
    detail_window.resizable(False, False)

    detail_frame = tk.Frame(detail_window, padx=10, pady=10)
    detail_frame.pack(fill='both', expand=True)

    tk.Label(detail_frame, text="Nama Tugas:", font=("Arial", 10,
    "bold")).grid(row=0, column=0, sticky='nw', pady=2)
    tk.Label(detail_frame, text=tugas["nama"], font=("Arial",
    10)).grid(row=0, column=1, sticky='nw', pady=2)


    tk.Label(detail_frame, text="Deskripsi:", font=("Arial",
    10, "bold")).grid(row=1, column=0, sticky='nw', pady=2)
    deskripsi_text_area = tk.Text(detail_frame, wrap='word', height
       =8, width=40, font=("Arial", 10),
    bg=detail_frame.cget('bg'), relief='flat') # Sesuaikan warna
       latar
    deskripsi_text_area.insert(tk.END, tugas["deskripsi"])
    deskripsi_text_area.config(state='disabled') # Nonaktifkan agar
        tidak bisa diedit
    deskripsi_text_area.grid(row=1, column=1, sticky='nsew', pady
       =2)

    scrollbar_desc = ttk.Scrollbar(detail_frame, orient='vertical',
        command=deskripsi_text_area.yview)
    scrollbar_desc.grid(row=1, column=2, sticky='ns', pady=2)
```

```
    deskripsi_text_area.config(yscrollcommand=scrollbar_desc.set)

    tk.Label(detail_frame, text="Deadline:", font=("Arial", 10,
    "bold")).grid(row=2, column=0, sticky='nw', pady=2)
    tk.Label(detail_frame, text=tugas["deadline"], font=("Arial",
    10)).grid(row=2, column=1, sticky='nw', pady=2)

    # Prioritas
    tk.Label(detail_frame, text="Prioritas:", font=("Arial",
    10, "bold")).grid(row=3, column=0, sticky='nw', pady=2)
    tk.Label(detail_frame, text=tugas["prioritas"],
    font=("Arial", 10)).grid(row=3, column=1, sticky='nw', pady=2)

    # Kategori
    tk.Label(detail_frame, text="Kategori:", font=("Arial", 10,
    "bold")).grid(row=4, column=0, sticky='nw', pady=2)
    tk.Label(detail_frame, text=tugas["kategori"],
    font=("Arial", 10)).grid(row=4, column=1, sticky='nw', pady=2)

    # Konfigurasi agar Text widget bisa membesar
    detail_frame.grid_rowconfigure(1, weight=1)
    detail_frame.grid_columnconfigure(1, weight=1)
    # Tombol tutup
    btn_tutup = tk.Button(detail_frame, text="Tutup",
    command=detail_window.destroy, bg="#AF4C6D", fg="white")
    btn_tutup.grid(row=5, column=0, columnspan=3, pady=10)

    detail_window.grab_set()
    detail_window.transient(detail_window.master)

# Define Home Page frame
class HomePage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.configure(padx=20, pady=20)

        label = tk.Label(self, text="PrioritizeMe", font=("Arial",
           20, "bold"))
        label.pack(pady=30)


        # Load and display logo image
        try:
            original_image = Image.open("C:/Users/Nabilah/OneDrive/
               Gambar/Logo_Universitas Negeri Surabaya.jpg")
            resized_image = original_image.resize((150, 150))  #
               Sesuaikan ukuran gambar
            self.logo = ImageTk.PhotoImage(resized_image)
            logo_label = tk.Label(self, image=self.logo)
```

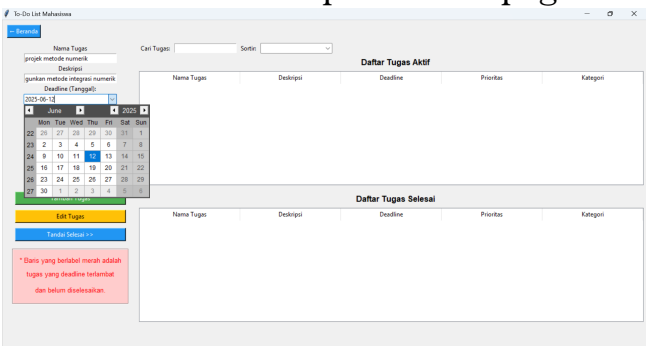
```
            logo_label.pack(pady=5)
        except Exception as e:
            print("Gagal Memuat logo:", e)

        # Welcome message
        label = tk.Label(self, text="Selamat Datang di Aplikasi To-
           Do List Mahasiswa",
        font=("Arial", 18, "bold"))
        label.pack(pady=30)

        info = tk.Label(self, text="Gunakan aplikasi ini untuk
           mengelola tugas kuliahmu dengan mudah dan efisien!",
        font=("Arial", 14))
        info.pack(pady=10)

        # Current date and time
        self.time_label = tk.Label(self, font=("Arial", 12))
        self.time_label.pack(pady=10)
        self.update_time()  # Call the method to update time

        btn_start = tk.Button(self, text="Mulai Mengelola Tugas",
        font=("Arial", 14), bg="#AF4C6D", fg="white", width=25,
        command=lambda: controller.show_frame("TaskPage"))
        btn_start.pack(pady=30)

    def update_time(self):
        current_time = time.strftime("%Y-%m-%d %H:%M:%S")
        self.time_label.config(text=current_time)
        self.time_label.after(1000, self.update_time)


def insertion_sort(data_list, key_func, reverse=False):

    n = len(data_list)
    for i in range(1, n):
        current_value = data_list[i]
        current_key = key_func(current_value)
        position = i

        while position > 0:
            previous_value = data_list[position - 1]
            previous_key = key_func(previous_value)

            if reverse: # Descending
                if current_key > previous_key:
                    data_list[position] = previous_value
                    position -= 1
                else:
                    break
            else: # Ascending
```

```
                if current_key < previous_key:
                    data_list[position] = previous_value
                    position -= 1
                else:
                    break
        data_list[position] = current_value

class MainApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("To-Do List Mahasiswa")
        self.geometry("1000x600")

        container = tk.Frame(self)
        container.pack(side='top', fill='both', expand=True)

        self.frames = {}

        self.search_var = tk.StringVar()
        self.sort_var = tk.StringVar()

        for F in (HomePage, TaskPage):
            page_name = F.__name__
            frame = F(parent=container, controller=self)
            self.frames[page_name] = frame
            frame.grid(row=0, column=0, sticky='nsew')

        self.show_frame("HomePage")

    def show_frame(self, page_name):
        frame = self.frames[page_name]
        frame.tkraise()

if __name__ == "__main__":
    app = MainApp()
    app.mainloop()
```

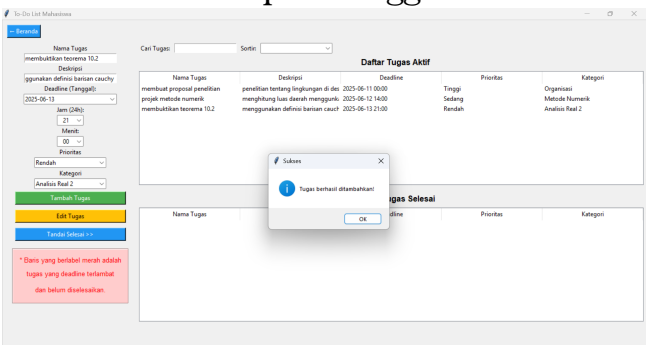
3 Tangkapan Layar Hasil *Running Source Code*



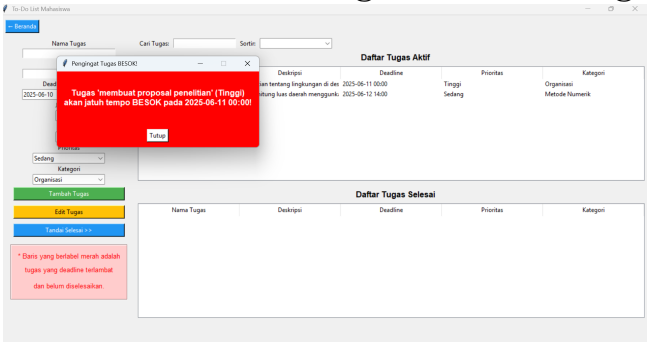
Gambar 1: Tampilan Homepage



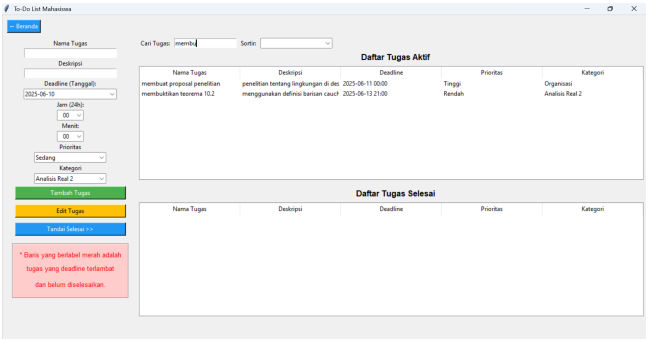
Gambar 2: Input Tanggal Deadline



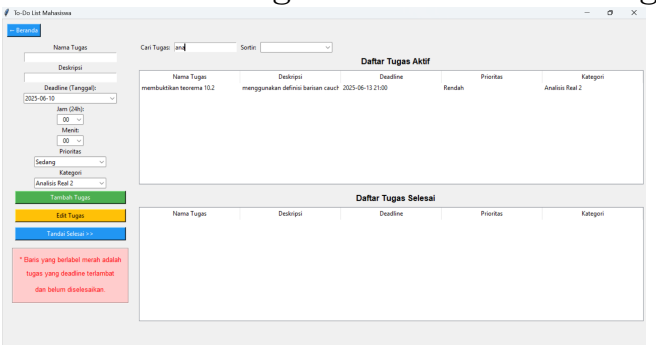
Gambar 3: Menambahkan tugas ke Daftar Tugas Aktif



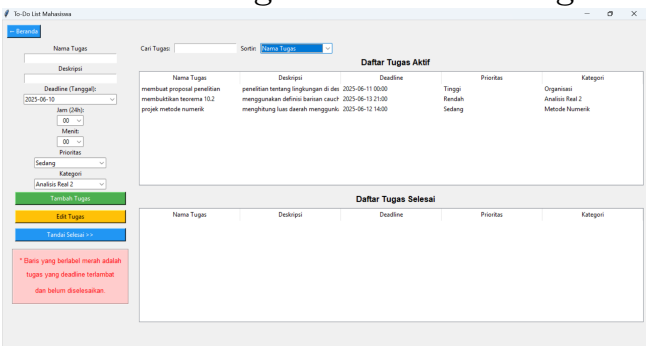
Gambar 4: Notifikasi Pengingat



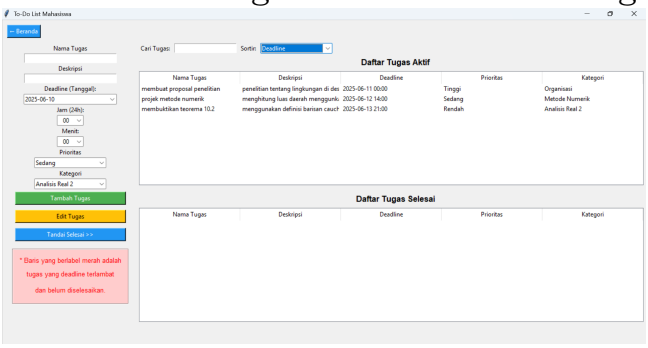
Gambar 5: Searching berdasarkan nama tugas



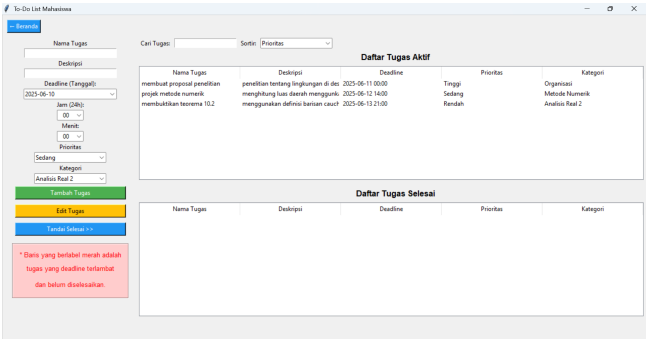
Gambar 6: Searching berdasarkan kategori tugas



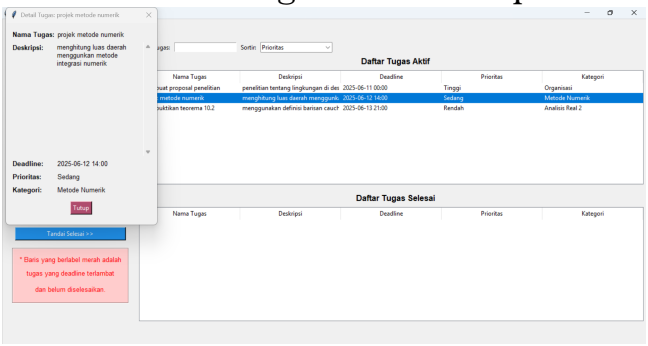
Gambar 7: Sorting berdasarkan nama tugas



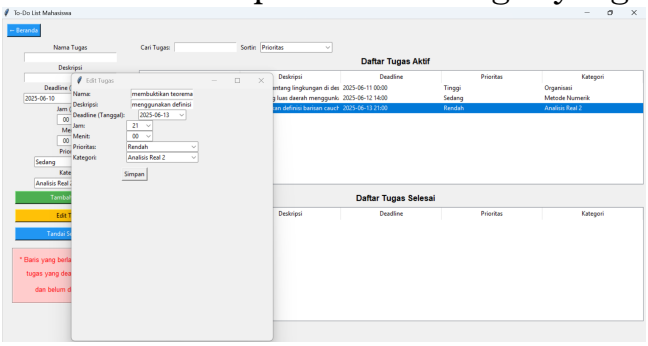
Gambar 8: Sorting berdasarkan deadline



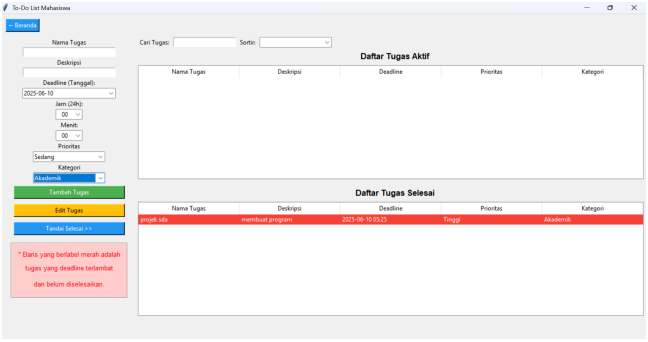
Gambar 9: Sorting berdasarkan prioritas



Gambar 10: Menampilkan detail tugas yang aktif



Gambar 11: Edit tugas yang aktif



Gambar 12: Tugas yang terlambat diselesaikan

4 Penjelasan

Program kami dirancang dengan arsitektur multi-halaman (`HomePage` dan `TaskPage`) yang dikelola oleh kelas `MainApp`. Tkinter adalah *event-driven*, artinya program menunggu pengguna melakukan sesuatu (klik tombol, ketik teks) dan kemudian meresponsnya.

4.1 Cara Kerja Program dengan GUI Tkinter

- **Inisialisasi GUI:** Ketika `MainApp` dimulai, ia membuat jendela utama dan dua *frame* (`HomePage` dan `TaskPage`). Setiap *frame* memiliki *widget* (tombol, *entry*, *combobox*, *treeview*) yang diatur di dalamnya. `HomePage` menampilkan informasi awal dan tombol Mulai, sementara `TaskPage` adalah pusat manajemen tugas.
- **Navigasi Antar Halaman:** Tombol mulai Mengelola Tugas di `HomePage` memanggil `controller.show_frame("TaskPage")`. Fungsi `show_frame` hanya menaikkan *frame* yang diminta ke bagian atas tumpukan, sehingga hanya *frame* itu yang terlihat.
- **Interaksi Pengguna dan Pembaruan Data:**
  - **Input Tugas:** Saat pengguna mengisi *entry* dan *combobox* di `TaskPage` lalu mengklik "Tambah Tugas", informasi diambil dari *widget* GUI (misalnya `self.entry_nama.get()`). Data ini kemudian diolah dan disimpan ke struktur data *backend*.
  - **Pembaruan Tampilan:** Setelah data *backend* berubah (tugas ditambahkan, diedit, atau selesai), fungsi `update_treeview()` dipanggil. Fungsi ini bertanggung jawab untuk:
    - Mengosongkan isi `Treeview` yang lama.
    - Memfilter dan mengurutkan ulang data dari struktur data *backend*.
    - Menyisipkan data yang sudah diolah kembali ke `Treeview`, termasuk menerapkan *tag* warna untuk prioritas atau status *overdue*.
  - **Pencarian dan Pengurutan Otomatis:** Ketika pengguna mengetik di *search bar* atau memilih opsi *sort* dari *combobox*, *event* (`<KeyRelease>` atau `«ComboboxSelected»`) akan memicu pemanggilan `self.update_treeviews()` secara otomatis, sehingga tampilan `Treeview` langsung disesuaikan.
  - **Notifikasi Waktu Nyata:** Fungsi `cek_pengingat()` dijalankan secara berkala oleh `root.after()`. Ini adalah cara Tkinter menjalankan fungsi di latar belakang tanpa memblokir GUI. Jika ada tugas yang mendekati *deadline* atau *overdue*, jendela notifikasi baru (`tk.Toplevel`) akan muncul.
  - **Detail Tugas:** Saat pengguna melakukan *double-klik* pada item di `Treeview`, informasi detail tugas yang sesuai akan diambil dari struktur data dan ditampilkan di jendela baru (`tk.Toplevel`), memberikan pengalaman pengguna yang informatif.

Secara ringkas, program kami menggunakan GUI Tkinter sebagai jendela ke data yang disimpan dan dikelola di *backend* menggunakan struktur data dan algoritma. Setiap interaksi di GUI memicu logika di *backend*, dan setiap perubahan di *backend* direfleksikan kembali ke GUI untuk tampilan yang *up-to-date*.

4.2 Struktur Data yang Digunakan dan Implementasinya

Program kami memilih struktur data sebagai berikut :

1. **Heap (via `heapq` module) untuk `tugas_aktif`**

- Sebuah heap adalah struktur data berbasis pohon yang memenuhi properti heap: jika itu adalah min-heap (seperti yang digunakan `heapq`), nilai dari setiap node harus kurang dari atau sama dengan nilai anak-anaknya. Ini berarti elemen terkecil selalu ada di akar (puncak heap). Dalam program kami, `tugas_aktif` adalah list Python yang diperlakukan sebagai min-heap oleh fungsi-fungsi `heapq`. Setiap elemen dalam heap adalah tuple `(prioritas_numerik, deadline_datetime, counter_unik, tugas_dictionary)`.
- **Implementasi:** `tugas_aktif = []` (list kosong yang diperlakukan sebagai heap).
  - `heapq.heappush(tugas_aktif, (prioritas_num, deadline_dt, count, tugas))` digunakan di `tambah_tugas` dan `edit_tugas` untuk menambahkan tugas baru. Tuple `(prioritas_num, deadline_dt, count, tugas)` memastikan bahwa tugas diurutkan berdasarkan prioritas (1=Tinggi, 2=Sedang, 3=Rendah) lalu *deadline*, dan `count` untuk *tie-breaking*.
  - `heapq.heappop(tugas_aktif)` digunakan di `tandai_selesai` untuk mengambil dan menghapus tugas dengan prioritas tertinggi (yang paling 'penting').
- **Cara Kerja:** Min-heap memastikan bahwa elemen terkecil (tugas dengan prioritas tertinggi dan *deadline* terdekat) selalu mudah diakses di bagian atas heap. Ini sangat cocok untuk skenario **priority queue** di mana kami ingin selalu menyelesaikan tugas yang paling mendesak terlebih dahulu. Operasi penambahan (`heappush`) dan pengambilan (`heappop`) elemen prioritas tertinggi sangat efisien ($O(\log n)$).

2. **List Python untuk `tugas_selesai`**

- `tugas_selesai` adalah list Python standar.
- **Implementasi:** `tugas_selesai = []` (list kosong standar).
  - `masukkan_tugas_terurut_prioritas(tugas_selesai, tugas)` digunakan di `tambah_tugas` dan `tandai_selesai` untuk menambahkan tugas ke dalam list `tugas_selesai`.
- **Cara Kerja:** List adalah struktur data yang fleksibel untuk menyimpan koleksi item. Untuk `tugas_selesai`, karena tidak ada kebutuhan konstan untuk mengambil item "tertinggi" berdasarkan kriteria tertentu secara *real-time* seperti pada `tugas_aktif`, list sederhana sudah memadai. Fungsi `masukkan_tugas_terurut_prioritas` memastikan bahwa tugas yang selesai juga tetap terlihat rapi dan terurut berdasarkan prioritas untuk tampilan.

4.3 Algoritma Sorting (Pengurutan) yang Digunakan dan Implementasinya

Program kami menggunakan algoritma **Insertion Sort**.

1. **Implementasi**
   Berikut adalah pseudocode umum untuk Insertion Sort:

Listing 1: Pseudocode Insertion Sort

```
function insertion_sort(array):
    n = length of array
    for i from 1 to n-1:
        current_value = array[i]
        position = i

        while position > 0 AND current_value < array[position -
            1]:
            array[position] = array[position - 1] // Geser
               elemen ke kanan
            position = position - 1

        array[position] = current_value // Sisipkan
           current_value
```

2. **Cara Kerja**
   Insertion Sort bekerja mirip dengan cara kita mengurutkan kartu di tangan kita:

   (a) **Iterasi Awal**: Algoritma memulai dengan menganggap elemen pertama dari daftar (pada indeks 0) sebagai bagian yang sudah terurut.

   (b) **Mengambil Elemen Berikutnya**: Kemudian, ia mengambil elemen berikutnya dari daftar yang belum terurut (mulai dari indeks 1). Ini adalah `current_value`.

   (c) **Membandingkan dan Menggeser**:

   - `current_value` ini kemudian dibandingkan dengan elemen-elemen di bagian yang sudah terurut, bergerak mundur dari kanan ke kiri.
   - Jika `current_value` lebih kecil (untuk ascending) atau lebih besar (untuk descending) dari elemen yang sedang dibandingkan di bagian terurut, elemen yang sedang dibandingkan itu digeser satu posisi ke kanan. Proses pergeseran ini terus berlanjut.

   (d) **Menyisipkan**: Ketika algoritma menemukan posisi di mana `current_value` tidak lagi lebih kecil (atau lebih besar) dari elemen di sebelah kirinya, atau ketika ia mencapai awal dari daftar (indeks 0), `current_value` disisipkan ke posisi tersebut.

   (e) **Ulangi**: Langkah 2-4 diulang sampai semua elemen dari daftar yang belum terurut telah disisipkan ke posisi yang benar di bagian yang terurut.

3. **Contoh Ilustrasi (*Ascending Order*)**
   Misalkan kami memiliki daftar: `[5, 2, 8, 1, 9]`

   (a) **Iterasi 1 (i=1)**:

   - `current_value = 2`. Bagian terurut: `[5]`.
   - Bandingkan `2` dengan `5`. $2 < 5$, jadi geser `5` ke kanan. Daftar sementara: `[_ , 5]`.
   - Sisipkan `2` ke posisi kosong: `[2, 5]`.
   - Daftar saat ini: **`[2, 5],`** `8, 1, 9`

(b) **Iterasi 2 (i=2)**:

- `current_value = 8`. Bagian terurut: `[2, 5]`.
- Bandingkan `8` dengan `5`. `8` tidak lebih kecil dari `5`.
- Sisipkan `8` di posisinya.
- Daftar saat ini: **`[2, 5, 8],`** `1, 9`

(c) **Iterasi 3 (i=3)**:

- `current_value = 1`. Bagian terurut: `[2, 5, 8]`.
- Bandingkan `1` dengan `8`. $1 < 8$, geser `8` ke kanan: `[2, 5, _ , 8]`.
- Bandingkan `1` dengan `5`. $1 < 5$, geser `5` ke kanan: `[2, _ , 5, 8]`.
- Bandingkan `1` dengan `2`. $1 < 2$, geser `2` ke kanan: `[_ , 2, 5, 8]`.
- Mencapai awal daftar. Sisipkan `1`.
- Daftar saat ini: **`[1, 2, 5, 8],`** `9`

(d) **Iterasi 4 (i=4)**:

- `current_value = 9`. Bagian terurut: `[1, 2, 5, 8]`.
- Bandingkan `9` dengan `8`. `9` tidak lebih kecil dari `8`.
- Sisipkan `9` di posisinya.
- Daftar saat ini: **`[1, 2, 5, 8, 9]`**

Daftar sekarang sepenuhnya terurut.

4. **Peran `key_func` dalam Implementasi**
Parameter `key_func` sangat penting dalam implementasi `insertion_sort` ini. Ini memungkinkan algoritma untuk mengurutkan objek-objek kompleks (seperti dictionary tugas) berdasarkan atribut spesifik mereka, bukan objek itu sendiri.

   - `key_func=lambda x: datetime.datetime.strptime(x["deadline"], '%Y-%m-%d %H:%M')`: Untuk sorting berdasarkan deadline, `key_func` akan mengembalikan objek `datetime` dari string deadline.
   - `key_func=lambda x: x["nama"].lower()`: Untuk sorting berdasarkan nama, `key_func` akan mengembalikan nama tugas dalam huruf kecil untuk perbandingan yang tidak *case-sensitive*.
   - `key_func=lambda x: prioritas_level.get(x["prioritas"], 99)`: Untuk sorting berdasarkan prioritas, `key_func` akan mengembalikan nilai numerik dari prioritas (1 untuk Tinggi, 2 untuk Sedang, 3 untuk Rendah), sehingga "Tinggi" diurutkan di atas "Sedang" dan "Rendah".

   Dengan demikian, Insertion Sort dalam program kami secara efektif mengurutkan tampilan daftar tugas berdasarkan kriteria yang dipilih pengguna, sambil tetap menjaga stabilitas urutan di antara tugas-tugas dengan kunci sorting yang sama.

4.4 Algoritma Searching (Pencarian) yang Digunakan dan Implementasinya

Program kami menggunakan algoritma **Pencarian Sekuensial (Linear Search)**.

- **Implementasi:** Terjadi di dalam fungsi `update_treeview()`, tepatnya pada baris ini:

```
filtered_tasks = [
    t for t in display_list
    if (search_filter in t["nama"].lower()
        or search_filter in t["kategori"].lower()
        or search_filter in t["deadline"].lower())
]
```

  Ketika pengguna mengetik di `self.entry_search`, nilai `search_filter` akan terisi. Loop ini kemudian akan memeriksa *setiap* tugas dalam `display_list` (yang bisa berupa tugas aktif atau selesai) untuk melihat apakah nama, kategori, atau *deadline*-nya mengandung `search_filter`.

- **Cara Kerja:** Pencarian sekuensial adalah algoritma paling dasar di mana setiap elemen dalam koleksi diperiksa satu per satu hingga elemen yang dicari ditemukan atau seluruh koleksi telah diperiksa. Ini efektif untuk mencari data yang tidak terurut atau ketika kriteria pencarian kompleks (misalnya, mencari *substring* di beberapa bidang seperti yang Anda lakukan). Untuk jumlah tugas yang relatif kecil, pencarian sekuensial sangat memadai dan mudah diimplementasikan, tanpa perlu struktur data pencarian yang lebih kompleks seperti *hash table* atau *binary search tree*.

5 Lampiran

https://www.overleaf.com/project/6847603e749967b7fd1d481d

Pustaka

[1] Taylor, H. L. (1990). Manajemen Waktu: Suatu Pedoman Pengelolaan Waktu yang Efektif dan Produktif. Jakarta: Binarupa Aksara.

[2] Gea, A. A. (2015). Time Management: Menggunakan Waktu Secara Efektif. Jakarta: Penerbit Media.