Contents

Introduction:

The Activity Selection Problem involves selecting the maximum number of non-overlapping activities from a given set, each characterized by its start and end times. Its goal is to maximize the number of activities a single person or resource can undertake without any time conflicts. Formally, given n activities with start times $s_1, s_2, ..., s_n$ and end times $e_1, e_2, ..., e_n$, the objective is to find a subset of activities A such that no two activities in A overlap, and $|A|$ is maximized.

The Greedy Activity Selection Algorithm is a commonly used approach to efficiently solve this problem. It sorts the activities based on their finish times and then iteratively selects the activity with the earliest finish time that does not conflict with the previously selected ones. This greedy strategy ensures that at each step, the locally optimal choice is made, leading to a globally optimal solution.

The significance of the Activity Selection Problem extends to various real-world scenarios such as scheduling tasks in a project, allocating resources in manufacturing processes, or organizing events in time-constrained environments. Its efficient solution makes it a fundamental problem in algorithmic design and optimization theory, serving as a basis for developing scheduling algorithms in diverse fields. By applying the principles of greedy algorithms to activity selection, efficient solutions can be obtained, making it a cornerstone in the study and application of combinatorial optimization problems.

Code:

```
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox

def get_user_input():

    root = tk.Tk()
    root.title("Activity Input")
    num_activities = tk.IntVar()
    start_time_vars = []
    end_time_vars = []
    activities = []


    def submit():
        activities.clear()

        for i in range(num_activities.get()):
            start_time = int(start_time_vars[i].get())
```

```
            end_time = int(end_time_vars[i].get())

            if start_time < 0:
                messagebox.showerror("Input Error", f"Activity {i + 1}: Start time cannot be negative.")
                break

            if start_time >= end_time:
                messagebox.showerror("Input Error", f"Activity {i + 1}: Start time cannot be greater than or equal
to end time.")
                break

            activities.append((start_time, end_time, i + 1))

        root.quit()


    def create_activity_entries():
        for widget in root.pack_slaves():
            widget.destroy()

        for i in range(num_activities.get()):
            activity_frame = ttk.Frame(root)
            activity_frame.pack(padx=10, pady=5)

            ttk.Label(activity_frame, text=f"Activity {i + 1}:\t", font=('calibri', 15, 'bold')).pack(side='left', pady=0)
            ttk.Label(activity_frame, text=f"Start Time", font=('Helvetica', 10)).pack(side='left', pady=0)
            start_time_var = tk.StringVar()
            ttk.Entry(activity_frame, textvariable=start_time_var, width=8).pack(side='left', pady=0)
            ttk.Label(activity_frame, text="\tEnd Time", font=('Helvetica', 10)).pack(side='left', pady=0)
            end_time_var = tk.StringVar()
            ttk.Entry(activity_frame, textvariable=end_time_var, width=8).pack(side='left', pady=0)

            start_time_vars.append(start_time_var)
            end_time_vars.append(end_time_var)

        submit_button = ttk.Button(root, text="Submit", command=submit)
        submit_button.pack(pady=5)

        root.bind("<Return>", lambda event: submit())

    input_frame = ttk.Frame(root)
    input_frame.pack(padx=20, pady=10)

    ttk.Label(input_frame, text="Enter the number of activities:  ", font=('Helvetica', 12,
'bold')).pack(side='left', pady=0)
    num_activities_entry = ttk.Entry(input_frame, textvariable=num_activities, width=10, font=('Helvetica',
10, 'bold'))
    num_activities_entry.pack(pady=0)

    create_activities_button = ttk.Button(root, text="Create Activity Entries",
command=create_activity_entries,)
```

```
    create_activities_button.pack(pady=10)


    num_activities_entry.bind("<Return>", lambda event: create_activity_entries())

    root.mainloop()

    return activities, num_activities.get()
def activity_selection(activities):
    n = 0
    activities.sort(key=lambda x: (x[1], x[0]))

    selected_activities = []

    last_end_time = float('-inf')

    for activity in activities:
        start_time, end_time, activity_num = activity

        if start_time >= last_end_time:
            selected_activities.append(activity)
            n = n+1
            last_end_time = end_time
    return selected_activities, activities, n
def plot_all_activities(ax, displayed_activities, selected_activities, n):
    bar_height = 0.3
    for index, activity in enumerate(displayed_activities):
        start, end, num = activity
        color = 'red'
        t = f'✗'
        if activity in selected_activities:
            color = 'green'
            t = f'✓'

        rect = mpatches.Rectangle((start, index), end - start, bar_height, color=color, alpha=0.5)
        ax.add_patch(rect)

        ax.text(start + (end - start) / 2, index + bar_height / 2, f'Activity{num}',
                color='white' if color == 'red' else 'red', ha='center', va='center', fontsize=(35 / n))

        ax.text(end+0.05, index + 0.05, f'{t}', color='black' if color == 'red' else 'black', fontsize=(80/n))
def sort_and_unsorted_activities(ax, displayed_activities, n, color):
    bar_height = 0.3
    for index, activity in enumerate(displayed_activities):
        start, end, num = activity



        rect = mpatches.Rectangle((start, index), end - start, bar_height, color=color, alpha=0.5)
        ax.add_patch(rect)

        ax.text(start + (end - start) / 2, index + bar_height / 2, f'Activity{num}',
```

```
            color='white', ha='center', va='center', fontsize=(37 / n))
def main():
    activities, num_activities = get_user_input()
    min_start_time = min(activities, key=lambda x: x[0])[0]
    max_end_time = max(activities, key=lambda x: x[1])[1]

    fig, ax = plt.subplots(figsize=(12, 6))

    plot_unsort_sort(activities, ax, min_start_time, max_end_time, num_activities, "UNSORTED ACTIVITY",
'red')
    selected_activities, sorted_activity, n = activity_selection(activities)
    plot_unsort_sort(activities, ax, min_start_time, max_end_time, num_activities, "SORTED ACTIVITY",
'blue')
    plot_bar(activities, ax, min_start_time, max_end_time, selected_activities, num_activities)
    plot_unsort_sort(selected_activities, ax, min_start_time, max_end_time, num_activities, "SELECTED
ACTIVITY", 'purple')
    plt.close(fig)
def plot_bar(activities, ax, min_start_time, max_end_time, selected_activities, num_activities):
    ax.set_xlim(min_start_time - 1, max_end_time + 5)
    ax.set_ylim(-0.5, len(activities) - 0.5)
    ax.set_yticks(range(len(activities)))
    ax.set_yticklabels([f'{i + 1}' for i in range(len(activities))])

    ax.set_xlabel('Time')
    ax.set_xticks(range(min_start_time - 1, max_end_time + 6))
    ax.set_xticklabels(range(min_start_time - 1, max_end_time + 6))

    ax.grid(True, axis='x', linestyle='--', alpha=0.5)
    index = 0
    displayed_activities = []

    while index < len(activities):
        displayed_activities.append(activities[index])

        ax.clear()

        ax.set_xlim(min_start_time - 1, max_end_time + 2)
        ax.set_ylim(-0.5, len(activities) - 0.5)
        ax.set_yticks(range(len(activities)))
        ax.set_yticklabels([f'{i + 1}' for i in range(len(activities))])
        ax.set_ylabel('Activities→', fontsize=20)
        ax.set_xlabel('Time→', fontsize=20)
        ax.set_xticks(range(min_start_time - 1, max_end_time + 2))
        ax.set_xticklabels(range(min_start_time - 1, max_end_time + 2))
        ax.grid(True, axis='x', linestyle='--', alpha=0.5)

        plot_all_activities(ax, displayed_activities, selected_activities, num_activities)


        ax.set_title('ACTIVITY SELECTION')
```

```
        plt.draw()

        plt.waitforbuttonpress()

        index += 1
    return

def plot_unsort_sort(activities, ax, min_start_time, max_end_time, num_activities, text, color):
    ax.set_xlim(min_start_time - 1, max_end_time + 5)
    ax.set_ylim(-0.5, len(activities) - 0.5)
    ax.set_yticks(range(len(activities)))
    ax.set_yticklabels([f'{i + 1}' for i in range(len(activities))])

    ax.set_xlabel('Time')
    ax.set_xticks(range(min_start_time - 1, max_end_time + 6))
    ax.set_xticklabels(range(min_start_time - 1, max_end_time + 6))

    ax.grid(True, axis='x', linestyle='--', alpha=0.5)
    index = 0
    displayed_activities = []

    while index < len(activities):
        displayed_activities.append(activities[index])

        ax.clear()

        ax.set_xlim(min_start_time - 1, max_end_time + 2)
        ax.set_ylim(-0.5, len(activities) - 0.5)
        ax.set_yticks(range(len(activities)))
        ax.set_yticklabels([f'{i + 1}' for i in range(len(activities))])
        ax.set_ylabel('Activities→', fontsize=20)
        ax.set_xlabel('Time→', fontsize=20)
        ax.set_xticks(range(min_start_time - 1, max_end_time + 2))
        ax.set_xticklabels(range(min_start_time - 1, max_end_time + 2))
        ax.grid(True, axis='x', linestyle='--', alpha=0.5)

        sort_and_unsorted_activities(ax, displayed_activities, num_activities, color)

        ax.set_title(f'{text}')

        plt.draw()


        index += 1
    plt.waitforbuttonpress()

    return


main()
```

Explanation of Code:

Libraries Used:

- ***matplotlib.pyplot:*** *Used for creating plots and visualizations.*
- ***matplotlib.patches:*** *Provides functionality to draw shapes such as rectangles (used for representing activities).*
- ***tkinter:*** *Used for creating GUI (Graphical User Interface) components.*
- ***tkinter.ttk:*** *Provides themed widgets that can be used to create more attractive GUIs.*
- ***tkinter.messagebox:*** *Used for displaying message boxes for input validation.*

Functions:

- *get_user_input():*
  - *This function creates a Tkinter window where the user can input details about activities such as start time and end time. It returns the activities entered by the user along with the number of activities.*

- ***activity_selection(activities):***
  - *This function sorts the activities based on their end times and selects the maximum number of non-overlapping activities. It returns the selected activities along with the total number of selected activities.*

- ***plot_all_activities(ax, displayed_activities, selected_activities, n):***
  - *This function plots all activities on the graph, with selected activities shown in green and unselected ones in red. It also displays a checkmark (✓) for selected activities and a cross mark (✗) for unselected ones.*

- ***sort_and_unsorted_activities(ax, displayed_activities, n, color):***
  - *This function plots activities either in sorted or unsorted order, depending on the color specified.*

- ***plot_bar(activities,ax,min_start_time,max_end_time,selected_activities, num_activities):***
  - *This function plots the bars representing activities on the graph along the time axis.*

- ***plot_unsort_sort(activities, ax, min_start_time, max_end_time, num, activities, text, color):***
  - *This function plots activities either in sorted or unsorted order based on the color specified, along with appropriate labels and axes.*

- ***main():***
  - *This function is the entry point of the program. It orchestrates the flow of execution by calling other functions and handling the plotting of activities.*

Overall Flow:

- *The main() function orchestrates the flow of the program by calling other functions in sequence.*
- *User inputs for activities are obtained using the get_user_input() function.*
- *Activities are sorted and selected using the activity_selection() function.*

- *Activities are plotted on the graph using various plotting functions, providing visual feedback to the user.*

GUI:

Tkinter is used to create a simple GUI window where users can input activity details.

Users can enter the number of activities and their start and end times.

Plotting:

Matplotlib is used to create a bar plot representing activities along the time axis.

Activities are represented as rectangles on the plot, with different colors indicating their selection status.

Text labels are added to indicate activity numbers, and checkmarks or cross marks are added to indicate selection status.

Execution Procedure:

- ***User Input Collection:***
  - *The process begins with the '**get_user_input()**' function, which opens a Tkinter GUI window to interactively collect information from the user. This includes the number of activities and their corresponding start and end times. The GUI interface provides a user-friendly way to input data, enhancing the overall user experience.*
  - *Upon submitting the input, the function performs validation checks to ensure the correctness of the provided data. Specifically, it verifies that the start times are not negative and that they are less than the corresponding end times. If any input is invalid, an error message is displayed using Tkinter's messagebox, prompting the user to correct the input.*
- ***Activity Selection:***
  - *Following the input collection, the '**activity_selection()**' function is invoked to sort the activities based on their end times and select the maximum number of non-overlapping activities. This function implements a greedy algorithm to efficiently select the activities.*
  - *The activities are sorted in ascending order of their end times, ensuring that activities with earlier end times are considered first. Then, the function iterates through the sorted activities, selecting those whose start times are greater than or equal to the end time of the last selected activity. This ensures that only non-overlapping activities are chosen, maximizing the number of activities selected.*

- ***Plotting Functions:***
  - *Several plotting functions are defined to visualize different aspects of the activity selection process. These functions utilize Matplotlib, a powerful plotting library in Python, to create bar graphs representing the activities.*
  - *'**plot_all_activities()**': This function plots all activities on the graph, with different colors indicating whether they are selected or not. It also displays activity numbers and indicators (✓or ✗) to denote selection status.*
  - *'**sort_and_unsorted_activities()**': This function plots both sorted and unsorted activities on separate graphs, providing a comparison between the original order and the sorted order based on end times.*
  - *'**plot_bar()**': This function plots the activities on a bar graph, with the x-axis representing time and the y-axis representing individual activities. It also sets labels and ticks for better visualization.*

❖ ***Main Function Execution:***

- *The '**main()**' function orchestrates the entire process. It first collects user input using '**get_user_input()**', calculates the minimum start time and maximum end time of the activities, and initializes the plot using Matplotlib.*
- *Next, it sequentially calls different plotting functions to visualize various aspects of the activity selection process, such as unsorted activities, sorted activities, selected activities, and the process of selecting activities. Each plot is displayed one after another, waiting for user interaction before proceeding to the next step.*

❖ ***Plotting Visualization:***

- *The plots generated by the plotting functions provide a visual representation of the activity selection process. The use of colors, labels, and indicators enhances the clarity and interpretability of the graphs. Selected activities are highlighted to distinguish them from others, aiding in understanding the selection criteria.*

❖ ***User Interaction:***

- *After each plot is displayed, the program waits for user interaction, typically in the form of a button press. This interactive approach allows users to follow the execution flow at their own pace, analyze the visualizations, and gain insights into the activity selection process.*

Steps:

1. ***Start:** The program execution begins with the 'main()' function.*

2. ***User Input Collection:***
   - *'main()' calls the 'get_user_input()' function to collect user input.*
   - *Inside 'get_user_input()', a Tkinter GUI window titled "Activity Input" is created to prompt the user.*
   - *Users enter the number of activities and their start/end times.*
   - *Input validation checks are performed to ensure correctness.*
   - *Once valid input is obtained, 'get_user_input()' returns the activities and their count back to 'main()'.*

3. ***Activity Selection:***
   - *'main()' then calls the 'activity_selection()' function.*
   - *Within 'activity_selection()', activities are sorted based on their end times.*
   - *A greedy algorithm selects non-overlapping activities with maximum coverage.*
   - *The selected activities, along with other relevant data, are returned to 'main()'.*

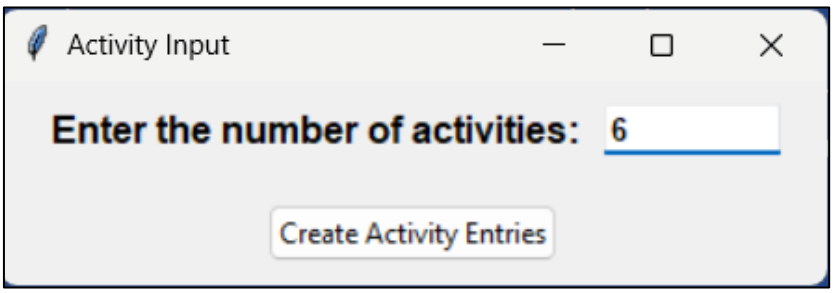
4. ***Plotting Functions:***
   - *Back in 'main()', various plotting functions are sequentially called to visualize different aspects of the activity selection process.*
   - *'plot_unsort_sort()' displays unsorted and sorted activities.*
   - *'plot_all_activities()' visualizes all activities, highlighting selected ones.*
   - *'plot_bar()' creates a bar plot representing the selected activities over time.*
   - *The visualizations aid in understanding the activity selection algorithm and its results.*

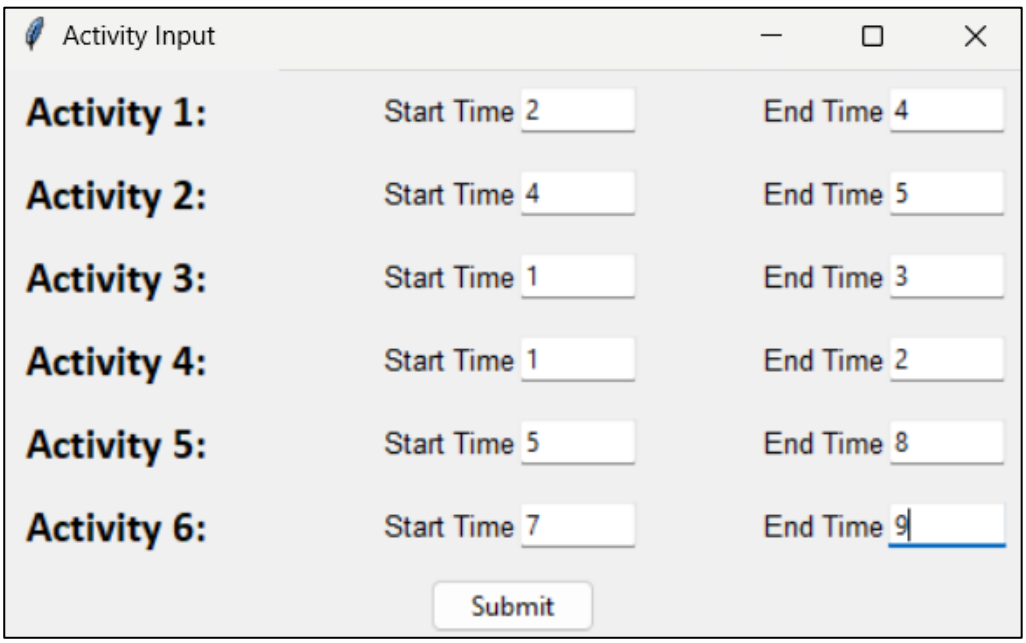
5. ***Main Function Execution:***

- *Throughout the execution, 'main()' controls the flow of the program.*
- *It coordinates user input collection, activity selection, and plotting.*
- *After each visualization, control returns to 'main()' for further processing.*

6. ***End:*** *Once all visualizations are displayed and user interaction (if any) is complete, the program execution concludes.*

Screenshots:





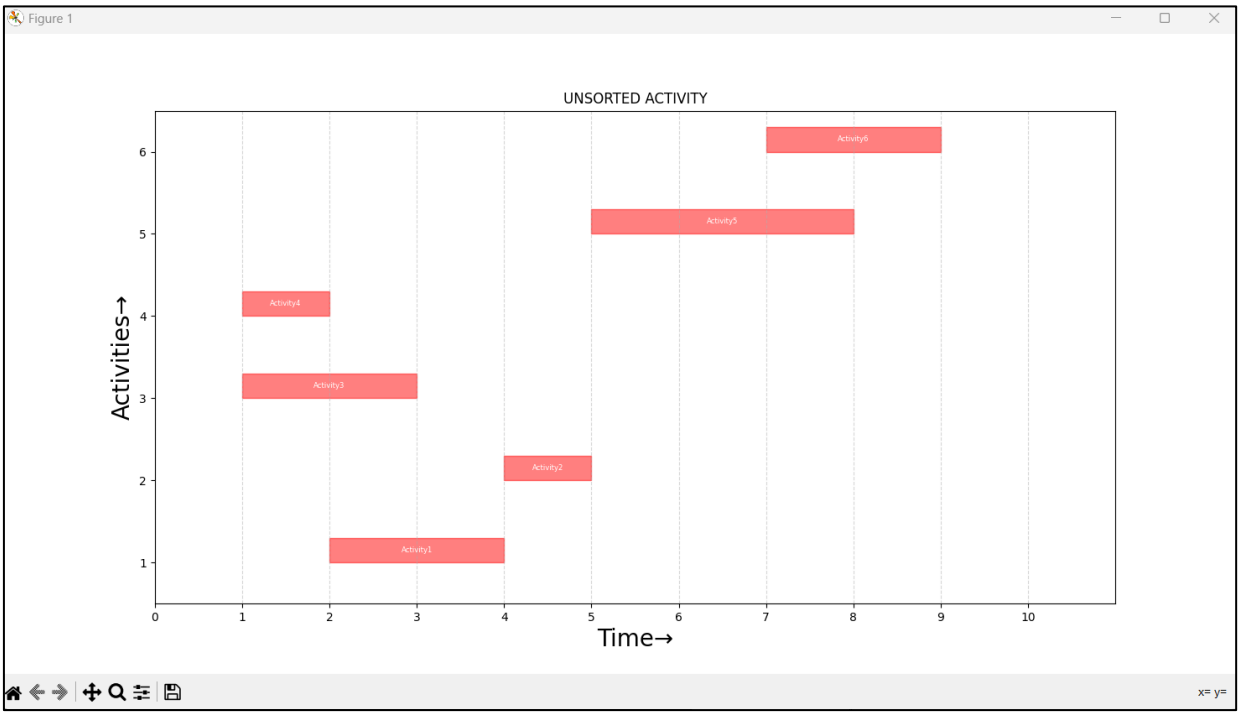


Figure 1
UNSORTED ACTIVITY
Activity6
Activity5
Activity4
Activity3
Activity2
Activity1
Activities→
Time→
x= y=

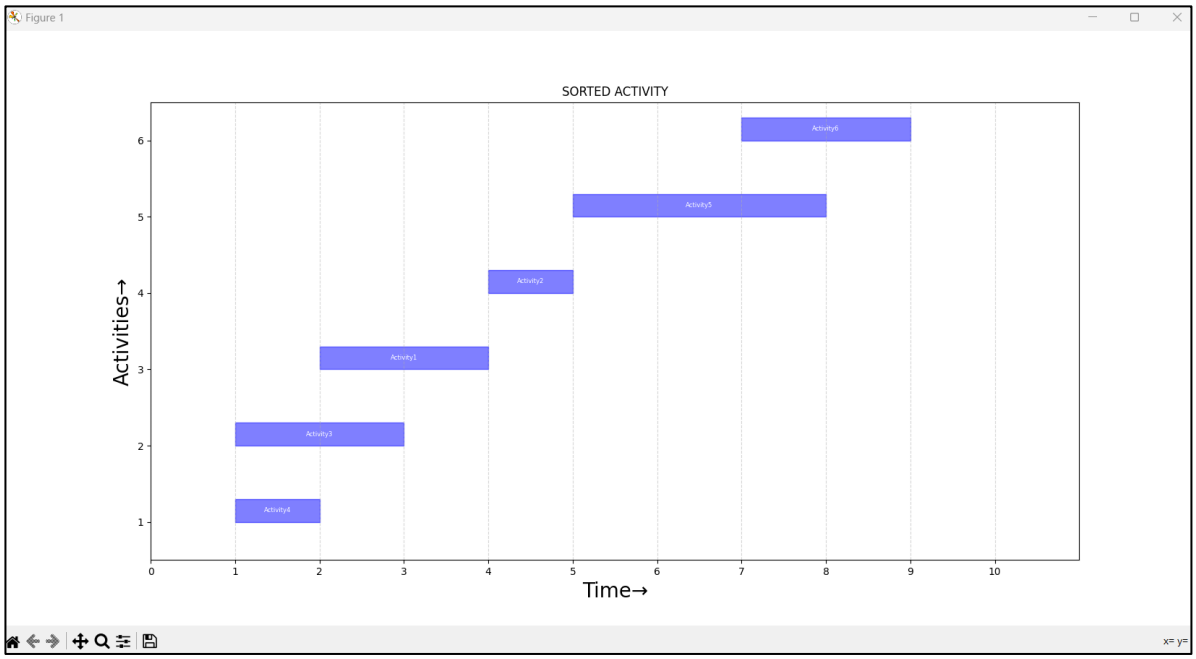


Figure 1
SORTED ACTIVITY
Activity6
Activity5
Activity2
Activity1
Activity3
Activity4
Activities→
Time→
x= y=

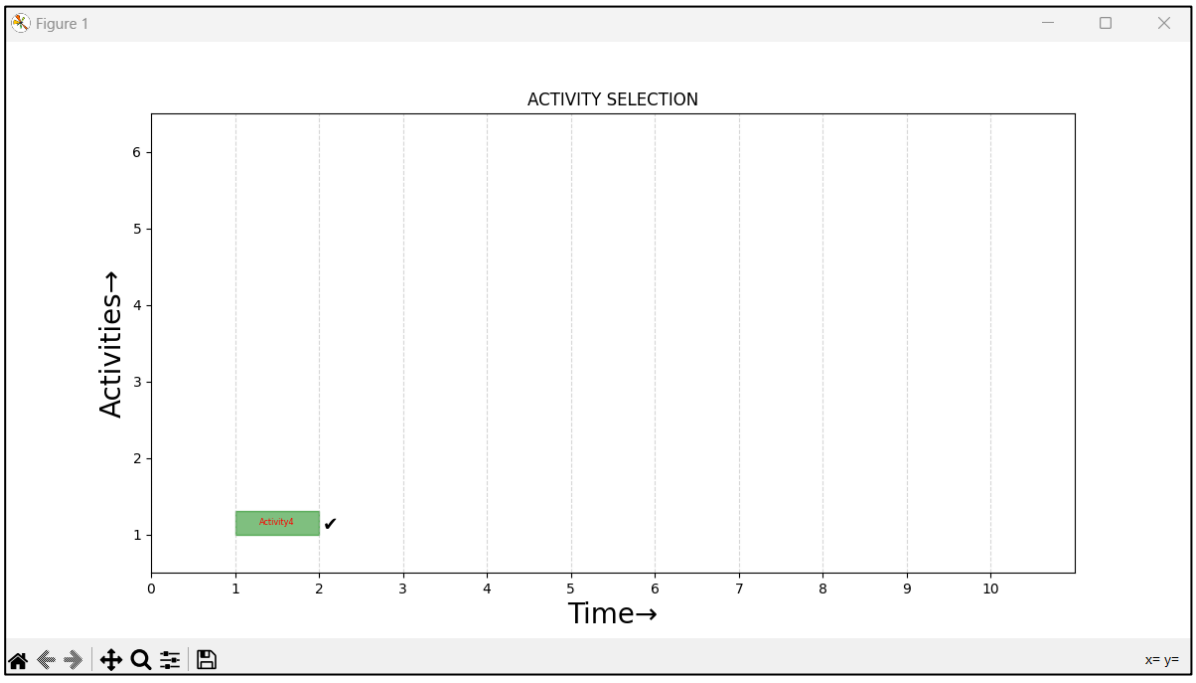


Figure 1
ACTIVITY SELECTION
Activity4
✔
Activities→
Time→
x= y=

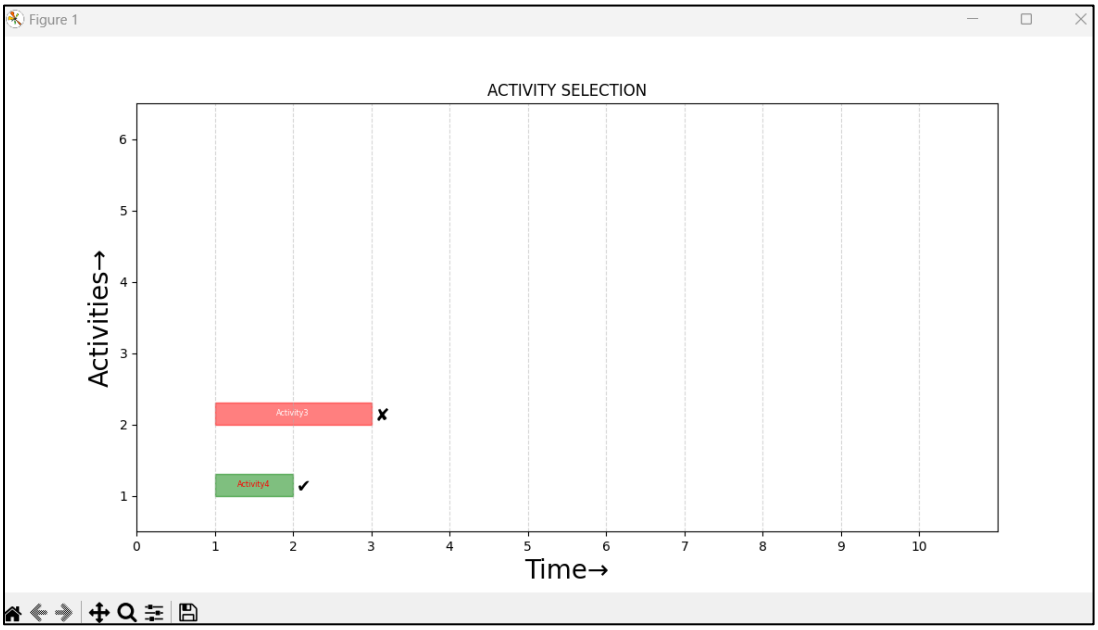


Figure 1
ACTIVITY SELECTION
Activities→
Time→
Activity3
Activity4

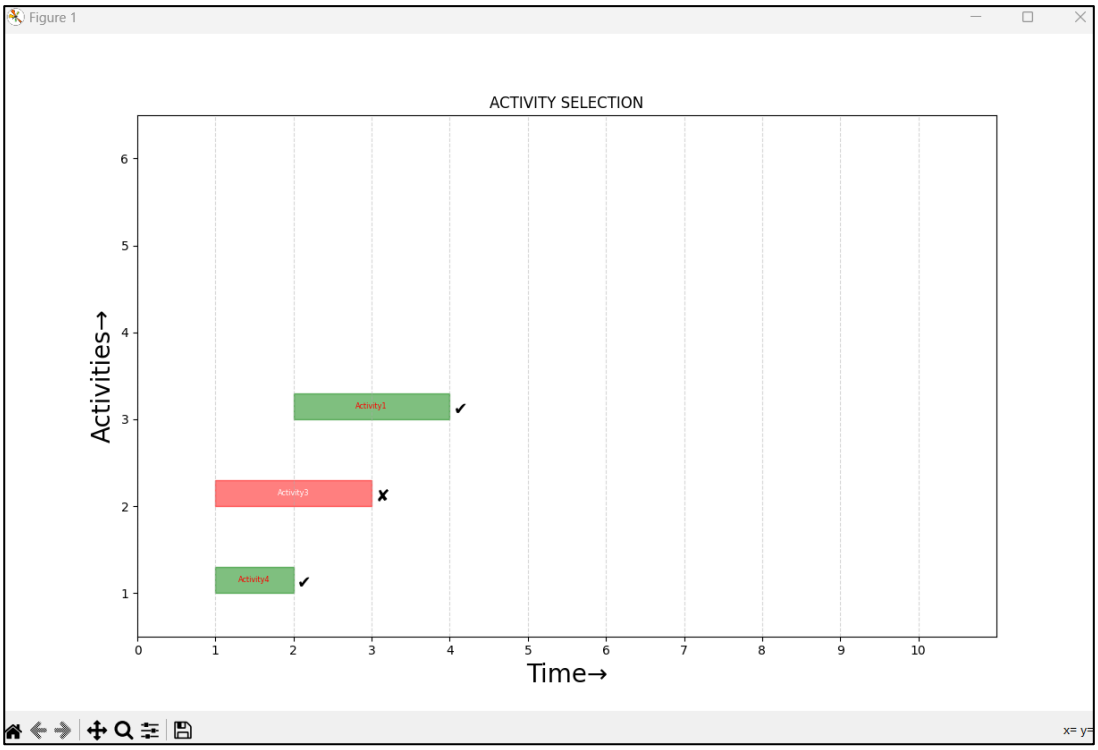


Figure 1
ACTIVITY SELECTION
Activities→
Time→
Activity1
Activity3
Activity4
x= y=

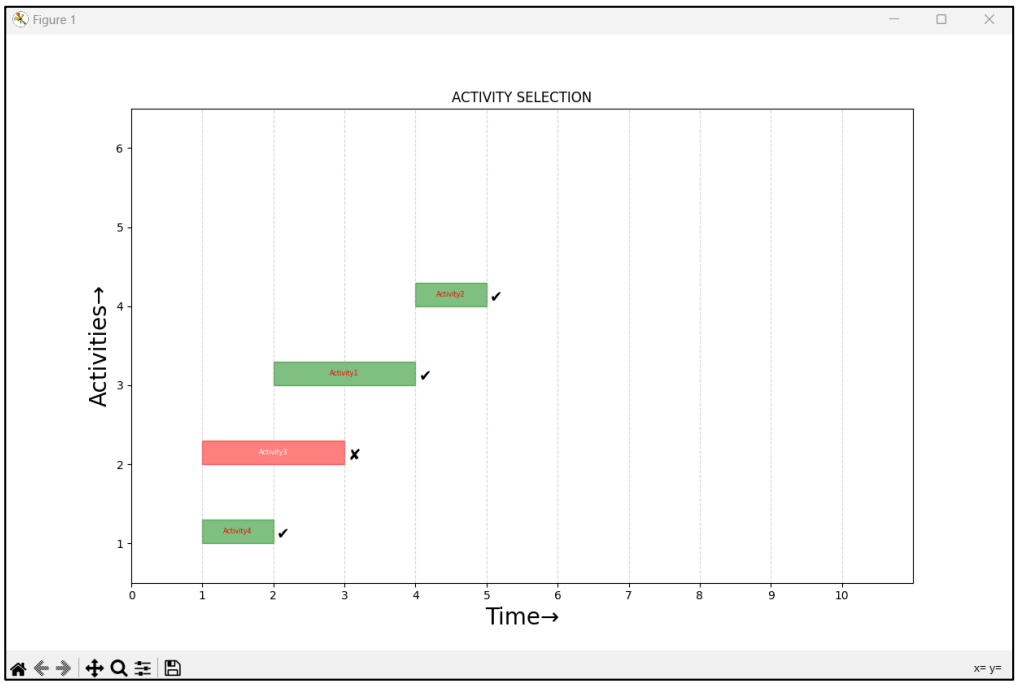


Figure 1
ACTIVITY SELECTION
Activities→
Time→
Activity2
Activity1
Activity3
Activity4
x= y=

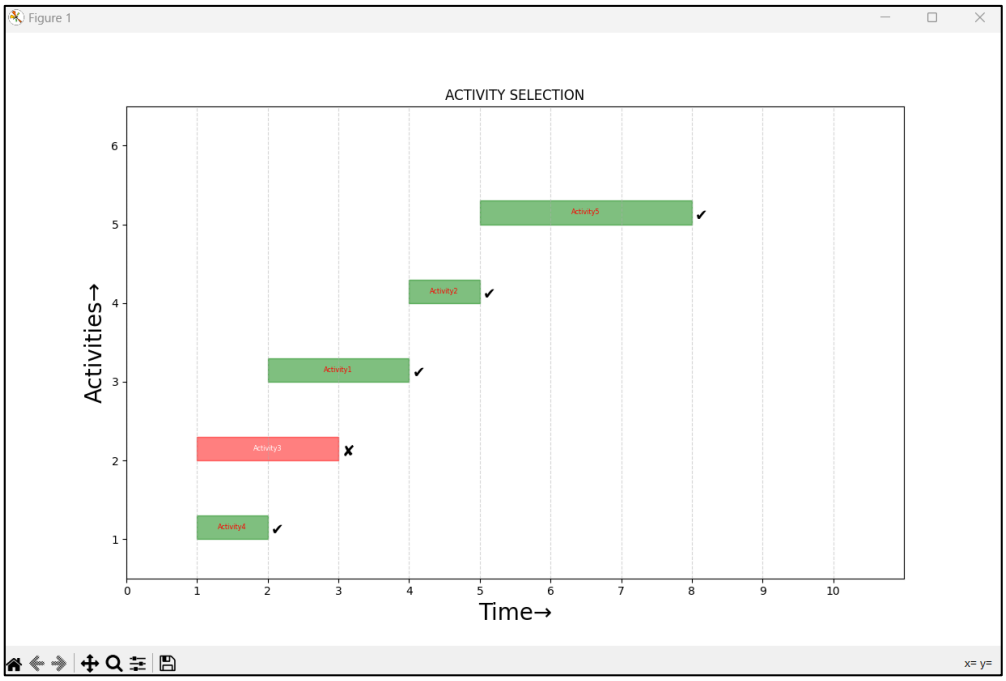


Figure 1
ACTIVITY SELECTION
Activity5
Activity2
Activity1
Activity3
Activity4
Activities→
Time→
x= y=

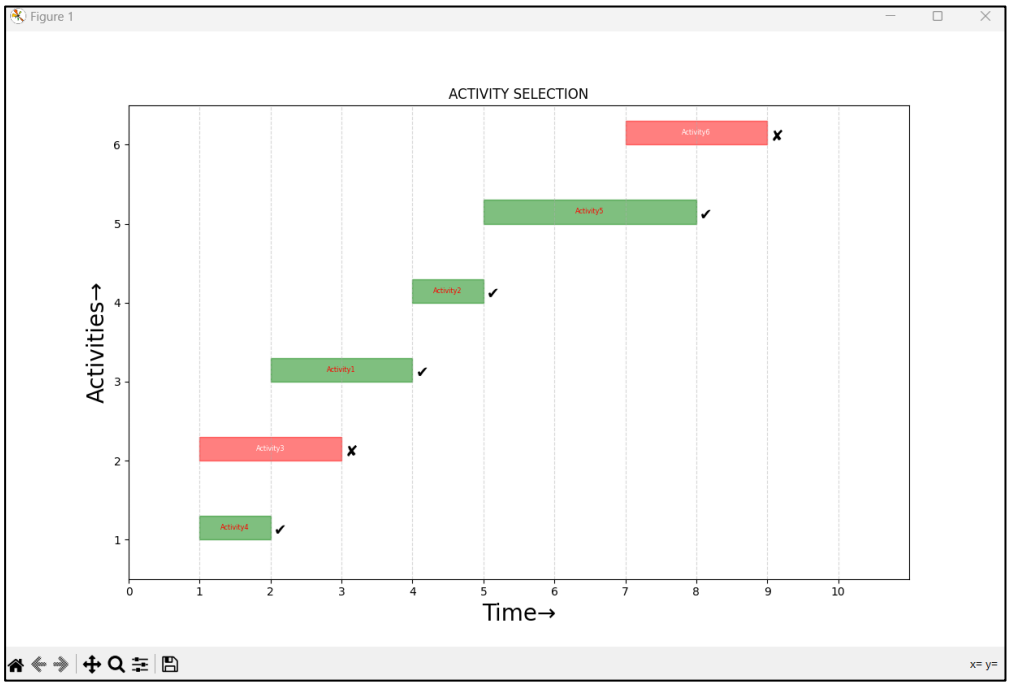


Figure 1
ACTIVITY SELECTION
Activity6
Activity5
Activity2
Activity1
Activity3
Activity4
Activities→
Time→
x= y=

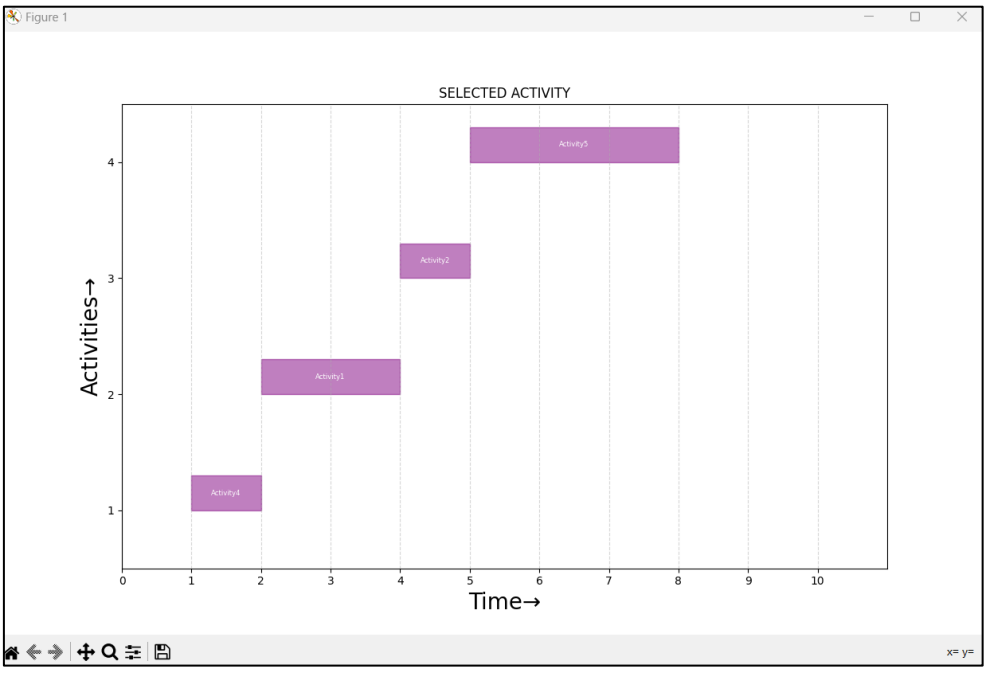


Figure 1
SELECTED ACTIVITY
Activity5
Activity2
Activity1
Activity4
Activities→
Time→
x= y=

CONCLUSION:

Animation clarifies the activity selection problem and the greedy algorithm's application. By visually presenting the sorting and selection process, it illustrates how the algorithm makes optimal choices at each step. This dynamic visualization enhances understanding and engagement, allowing viewers to explore different scenarios and appreciate the algorithm's efficiency. In summary, animation is a potent tool for comprehending complex algorithms succinctly.

REFERENCES:

1. *https://www.javatpoint.com/daa-tutorial*
2. *https://www.geeksforgeeks.org/design-and-analysis-of-algorithms/*
3. *https://www.tutorialspoint.com/design_and_analysis_of_algorithms/index.htm*
4. *https://www.programiz.com/dsa/greedy-algorithm*
5. *https://chat.openai.com/*