DESIGN AND ANALYSIS OF ALGORITHM

PRACTICAL-4

NAME: VEDANT BHUTADA

ROLL: 69

BATCH: A4

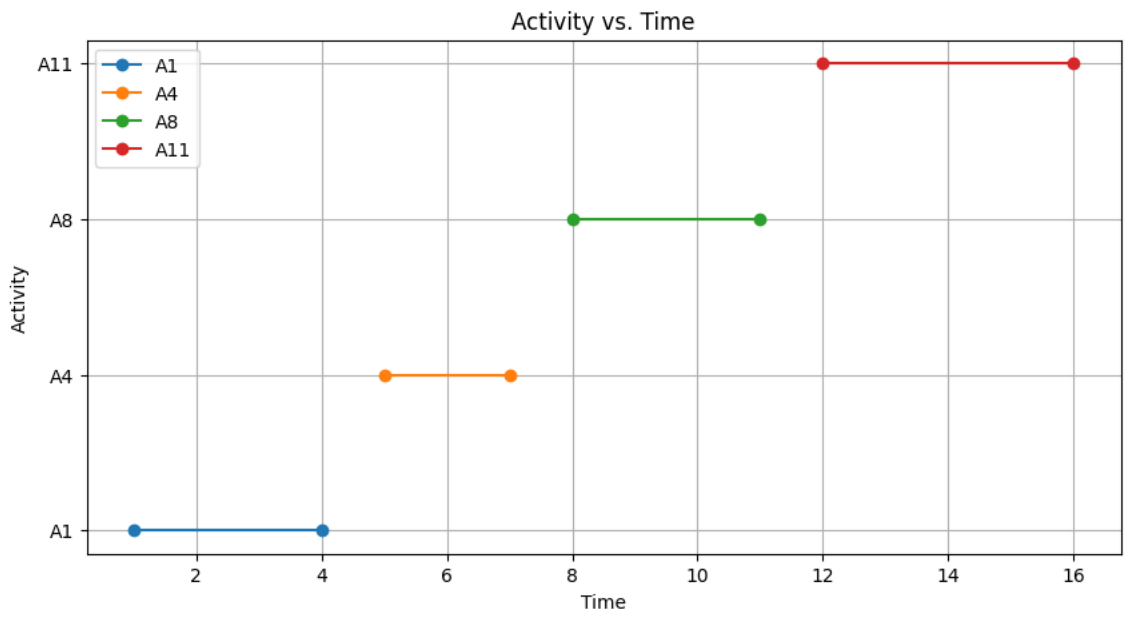
Part A

AIM: Implement activity selection algorithm for the given scenario.

Problem Definition: In the single-machine scheduling problem, we are given a set of n activities Ai. Each job i has a starting time si, deadline di and profit pi. At any time instant, we can do only one job. Doing a job i earns a profit pi. Generate a solution to select the largest set of mutually compatible jobs and calculate the total profit generated by the machine. The greedy algorithm for single-machine scheduling selects the job using activity selection algorithm.

```
import matplotlib.pyplot as plt
from tabulate import tabulate  # Import the tabulate library
#sorting
def selection_sort(activities):
    n = len(activities)
    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
            if activities[j][1] < activities[min_index][1]:
                min_index = j
        activities[i], activities[min_index] = activities[min_index], activities[i]

#activity selection
def activity_selection(activities):
    selection_sort(activities)

    selected_activities = []
    total_profit = 0
    current_time = 0

    for activity in activities:
        start_time, finish_time, activity_name, profit = activity
        if start_time >= current_time:
            selected_activities.append(activity)
            total_profit += profit
            current_time = finish_time

    return selected_activities, total_profit

# Define activities as a list of tuples (start_time, finish_time, activity_name, profit)
activities = [
    (1, 4, 'A1', 10),
    (3, 5, 'A2', 15),
    (0, 6, 'A3', 14),
    (5, 7, 'A4', 12),
    (3, 9, 'A5', 20),
    (5, 9, 'A6', 30),
    (6, 10, 'A7', 32),
    (8, 11, 'A8', 28),
    (8, 12, 'A9', 30),
    (2, 14, 'A10', 40),
    (12, 16, 'A11', 45)
]

selected, profit = activity_selection(activities)

# Extract start times, finish times, and activity names
start_times = [activity[0] for activity in selected]
finish_times = [activity[1] for activity in selected]
activity_names = [activity[2] for activity in selected]

# Display selected activities in tabular form
table = []
for activity in selected:
    start_time, finish_time, activity_name, activity_profit = activity
    table.append([activity_name, start_time, finish_time, activity_profit])

print("Selected Activities:")
headers = ["Activity Name", "Start Time", "Finish Time", "Profit"]
print(tabulate(table, headers, tablefmt="grid"))
print(tabulate(table, headers, tablefmt='grid'))

print("Total Profit:", profit)

# Plot activity vs. time graph
plt.figure(figsize=(10, 5))
for i in range(len(selected)):
    plt.plot([start_times[i], finish_times[i]], [i, i], label=activity_names[i], marker='o')

plt.xlabel('Time')
plt.ylabel('Activity')
plt.title('Activity vs. Time')
plt.yticks(range(len(selected)), activity_names)
plt.legend()
plt.grid(True)
plt.show()
```

```
Selected Activities:
+-----------------+--------------+--------------+----------+
| Activity Name   |   Start Time |  Finish Time |   Profit |
+=================+==============+==============+==========+
| A1              |            1 |            4 |       10 |
+-----------------+--------------+--------------+----------+
| A4              |            5 |            7 |       12 |
+-----------------+--------------+--------------+----------+
| A8              |            8 |           11 |       28 |
+-----------------+--------------+--------------+----------+
| A11             |           12 |           16 |       45 |
+-----------------+--------------+--------------+----------+
Total Profit: 95
```



Part B

Aim: Given three stacks of the positive numbers, the task is to find the possible equal maximum sum of the stacks with the removal of top elements allowed.

```
def equalMaxSumOfStacks(stack1, stack2, stack3):
    sum1 = sum(stack1)
    sum2 = sum(stack2)
    sum3 = sum(stack3)

    top1 = 0
    top2 = 0
    top3 = 0

    while sum1 != sum2 or sum2 != sum3:
        if sum1 >= sum2 and sum1 >= sum3:
            sum1 -= stack1[top1]
            top1 += 1
        elif sum2 >= sum1 and sum2 >= sum3:
            sum2 -= stack2[top2]
            top2 += 1
        else:
            sum3 -= stack3[top3]
            top3 += 1

    print("Equal Maximum Sum:", sum1)  # or sum2 or sum3, they are all equal

    print("Remaining elements in Stack 1:", stack1[top1:])
    print("Remaining elements in Stack 2:", stack2[top2:])
    print("Remaining elements in Stack 3:", stack3[top3:])

stack1 = [5, 8, 5, 3]
stack2 = [5, 6, 9, 4, 2, 2]
stack3 = [5, 3, 2, 1, 2]
equalMaxSumOfStacks(stack1, stack2, stack3)
```

```
Equal Maximum Sum: 8
Remaining elements in Stack 1: [5, 3]
Remaining elements in Stack 2: [4, 2, 2]
Remaining elements in Stack 3: [3, 2, 1, 2]
```