Experiment **#6**		Student ID	
Date		Student Name	[@KLWKS_BOT THANOS]

Experiment Title: Implementation of Programs on Greedy method Problems-I.

Aim/Objective: To understand the concept and implementation of Basic programs on Greedy method problems.

Description: The students will understand and able to implement programs on Greedy method problems.

Pre-Requisites:

Knowledge: Greedy Method and its related problems in C/Java/Python

Tools: Code Blocks/Eclipse IDE

Pre-Lab:

Given a set of activities with their start and end times, select the maximum number of activities that can be performed by a single person, assuming that no two activities overlap.

Input: A list of activities, where each activity is represented by a pair of start and end times.

Output: The maximum number of activities that can be selected.

Example: Input:

Activities = [(1, 3), (2, 5), (4, 6), (6, 7), (5, 8)]

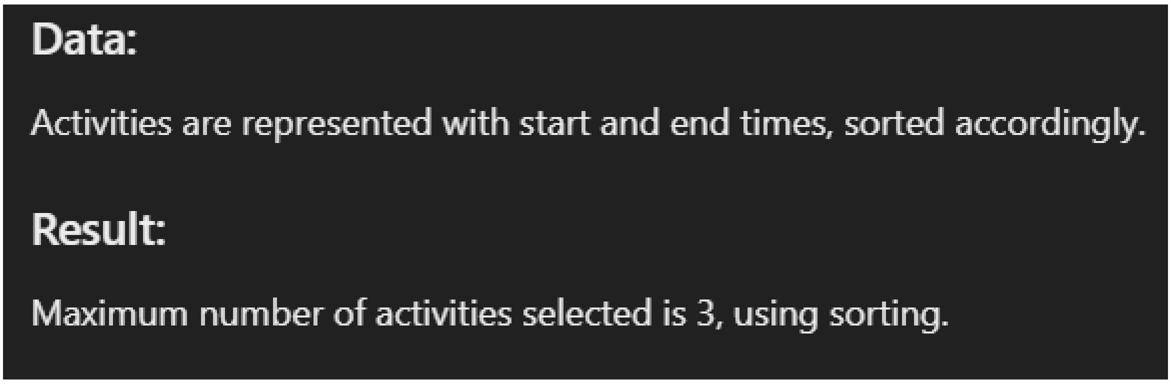
Output: 3

- **Procedure/Function:**

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int start, end;
} Activity;

int compare(const void *a, const void *b) {
    return ((Activity *)a)->end - ((Activity *)b)->end;
}
```

```
int maxActivities(Activity activities[], int n) {
    qsort(activities, n, sizeof(Activity), compare);

    int count = 1;
    int lastEnd = activities[0].end;

    for (int i = 1; i < n; i++) {
        if (activities[i].start >= lastEnd) {
            count++;
            lastEnd = activities[i].end;
        }
    }
    return count;
}

int main() {
    Activity activities[] = {{1, 3}, {2, 5}, {4, 6}, {6, 7}, {5, 8}};

    int n = sizeof(activities) / sizeof(activities[0]);

    printf("%d\n", maxActivities(activities, n));
    return 0;
}
```

- **Data and Result:**



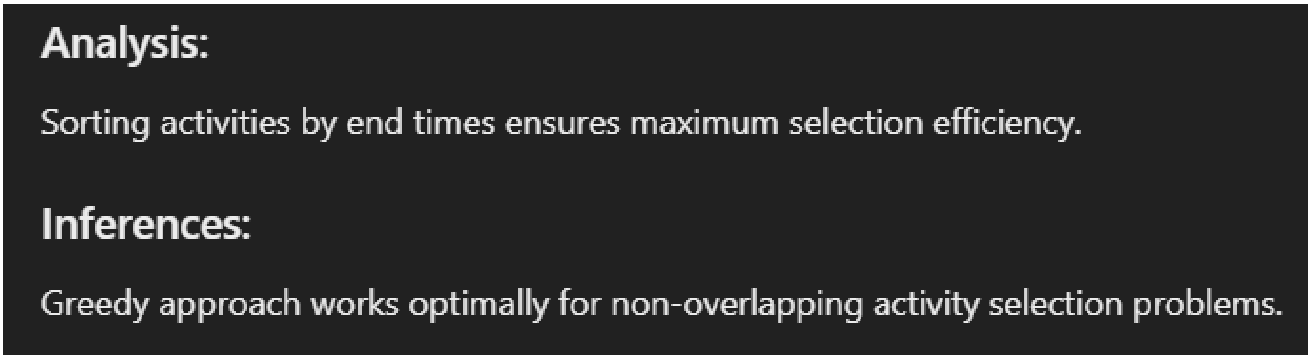
Data:

Activities are represented with start and end times, sorted accordingly.

Result:

Maximum number of activities selected is 3, using sorting.

- **Inference Analysis:**



Analysis:

Sorting activities by end times ensures maximum selection efficiency.

Inferences:

Greedy approach works optimally for non-overlapping activity selection problems.

In-Lab:

Given an array of size n that has the following specifications:

a. Each element in the array contains either a police officer or a thief.
b. Each police officer can catch only one thief.
c. A police officer cannot catch a thief who is more than K units away from the police officer.

We need to find the maximum number of thieves that can be caught.

Input: arr [ ] = {'P', 'T', 'T', 'P', 'T'},   k = 1.

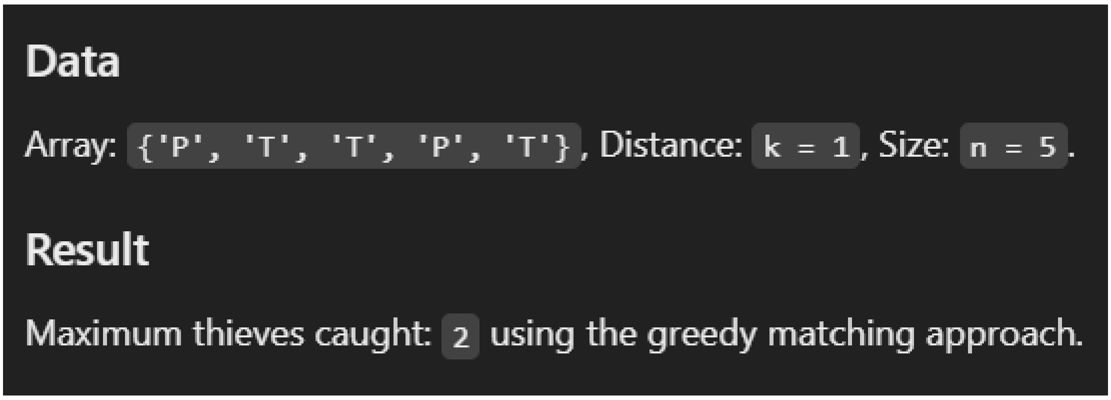
Output: 2

Here maximum 2 thieves can be caught; first police officer catches first thief and second police officer can catch either second or third thief.

- **Procedure/Program:**

```
#include <stdio.h>
#include <stdlib.h>


int maxThievesCaught(char arr[], int n, int k) {
    int police[n], thieves[n];
    int p = 0, t = 0, count = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] == 'P') {
            police[p++] = i;
        } else if (arr[i] == 'T') {
            thieves[t++] = i;
        }
    }

    int i = 0, j = 0;
    while (i < p && j < t) {
        if (abs(police[i] - thieves[j]) <= k) {
            count++;
            i++;
            j++;
        } else if (police[i] < thieves[j]) {
            i++;
        } else {
            j++;
        }
    }

    return count;
}

int main() {
    char arr[] = {'P', 'T', 'T', 'P', 'T'};
    int k = 1;
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("%d\n", maxThievesCaught(arr, n, k));

    return 0;
}
```

- **Data and Results:**



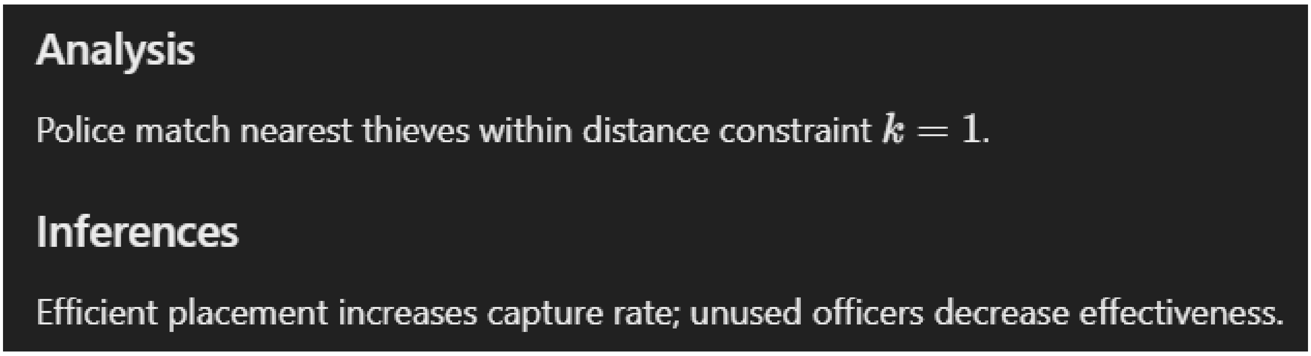
Data

Array: `{'P', 'T', 'T', 'P', 'T'}`, Distance: `k = 1`, Size: `n = 5`.

Result

Maximum thieves caught: `2` using the greedy matching approach.

- **Analysis and Inferences:**



Analysis

Police match nearest thieves within distance constraint $k = 1$.

Inferences

Efficient placement increases capture rate; unused officers decrease effectiveness.

Post-Lab:

Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes a single unit of time, so the minimum possible deadline for any job is 1. How to maximize total profit if only one job can be scheduled at a time.

Input

4

Job ID	**Deadline**	**Profit**
A	4	20
B	1	10
C	1	40
D	1	30

Output

60

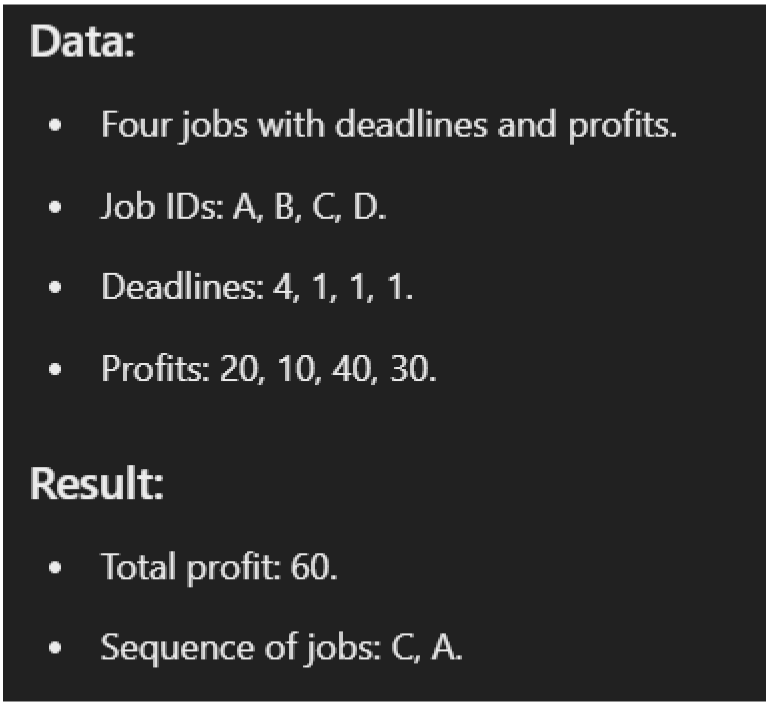
Profit sequence of jobs is c, a

- **Procedure/Program:**

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
   char jobId;
   int deadline;
   int profit;
} Job;

int compareJobs(const void* a, const void* b) {
   return ((Job*)b)->profit - ((Job*)a)->profit;
```

```
}

int jobScheduling(Job jobs[], int n) {
    qsort(jobs, n, sizeof(Job), compareJobs);

    int maxDeadline = 0;
    for (int i = 0; i < n; i++) {
        if (jobs[i].deadline > maxDeadline) {
            maxDeadline = jobs[i].deadline;
        }
    }

    int slots[maxDeadline + 1];
    for (int i = 0; i <= maxDeadline; i++) {
        slots[i] = -1;
    }

    int totalProfit = 0;
    int jobSequence[n];
    int jobCount = 0;

    for (int i = 0; i < n; i++) {
        for (int j = jobs[i].deadline; j > 0; j--) {
            if (slots[j] == -1) {
                slots[j] = i;
                totalProfit += jobs[i].profit;
                jobSequence[jobCount++] = i;
                break;
            }
        }
    }

    printf("Total Profit: %d\n", totalProfit);
    printf("Profit sequence of jobs: ");
    for (int i = 0; i < jobCount; i++) {
        printf("%c ", jobs[jobSequence[i]].jobId);
    }
    printf("\n");
    return totalProfit;
}

int main() {
    Job jobs[] = {
        {'A', 4, 20},
        {'B', 1, 10},
        {'C', 1, 40},
        {'D', 1, 30}
    };
    int n = sizeof(jobs) / sizeof(jobs[0]);

    jobScheduling(jobs, n);
    return 0;
}
```

- **Data and Results:**



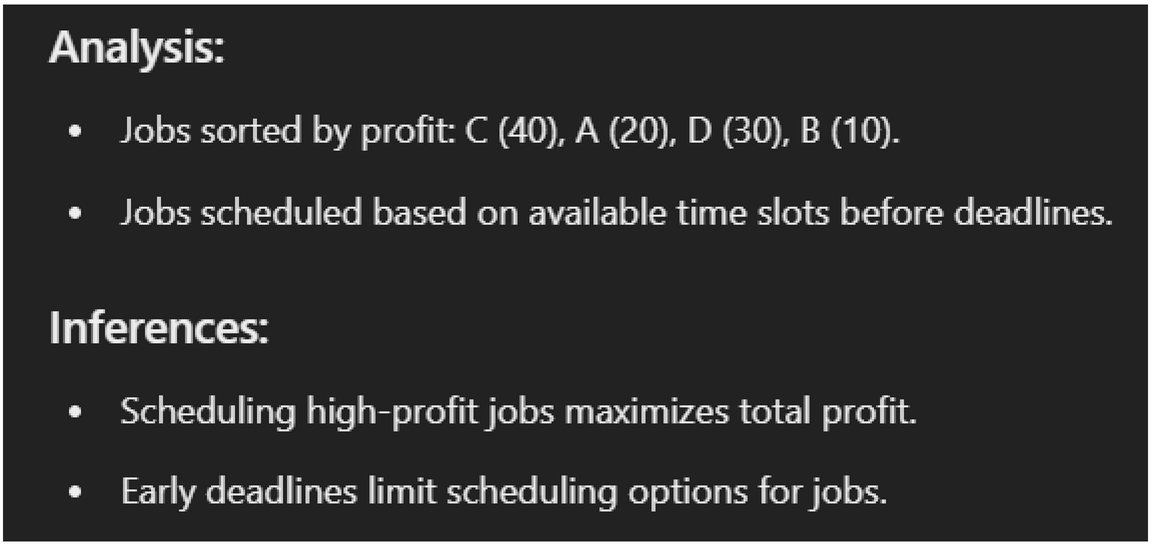
Data:

- Four jobs with deadlines and profits.
- Job IDs: A, B, C, D.
- Deadlines: 4, 1, 1, 1.
- Profits: 20, 10, 40, 30.

Result:

- Total profit: 60.
- Sequence of jobs: C, A.

- **Analysis and Inferences:**



Analysis:

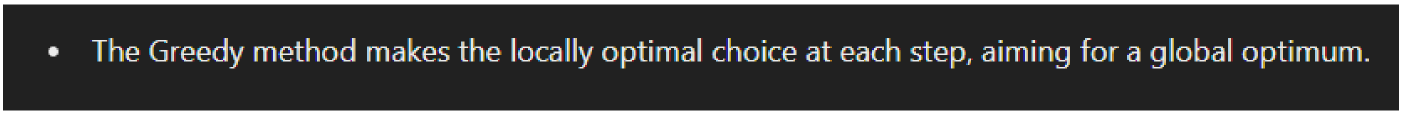
- Jobs sorted by profit: C (40), A (20), D (30), B (10).
- Jobs scheduled based on available time slots before deadlines.

Inferences:

- Scheduling high-profit jobs maximizes total profit.
- Early deadlines limit scheduling options for jobs.

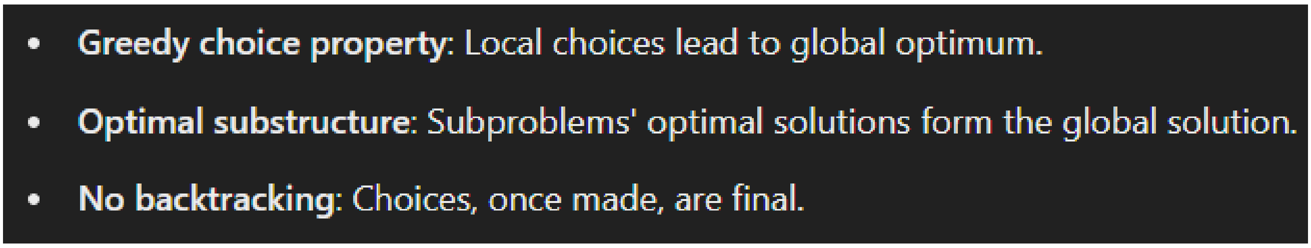
- **Sample VIVA-VOCE Questions (In-Lab):**

1. Explain the basic idea behind the Greedy method.



- The Greedy method makes the locally optimal choice at each step, aiming for a global optimum.

2. What are the characteristics of a problem that make it suitable for a Greedy algorithm?



- **Greedy choice property**: Local choices lead to global optimum.
- **Optimal substructure**: Subproblems' optimal solutions form the global solution.
- **No backtracking**: Choices, once made, are final.

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.