ACTIVITY SELECTION PROBLEM

A Report on Course end project Submitted in the Partial Fulfillment of the Requirements for the Award of the Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING (AI&ML)

Submitted By

POOJALA JAIVARDHAN	**24881A66G7**

Under the Esteemed Guidance of

Mr. YOGASH CHANDRA JOSHI
ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERRING (AI&ML)

VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to **JNTUH**, Approved by **AICTE**, Accredited by **NAAC**, with **A++** Grade, **ISO 9001:2015** Certified

Kacharam, Shamshabad, Hyderabad – 501218, Telangana, India

2025- 26

DECLARATION

I hereby declare that the work described in this report entitled **"Activity Selection Problem"** which is being submitted by us in partial fulfilment for the award of **BACHELOR OF TECHNOLOGY** in the Department of Computer Science & Engineering (AI&ML), Vardhaman College of Engineering to the Jawaharlal Nehru Technological University Hyderabad.

The work is original and has not been submitted for any Degree or Diploma of this or any other university.

POOJALA JAIVARDHAN (24881A66G7)

SUMMARY AND OUTCOMES (CSE- AIML , Subject: ADS, Code: **VAC606**)

UNIT ONE	**Linked Lists**
	Polynomial addition, multiplication
	Queue, Stack
	Depth in Linked List.
	Binary Tree, MST, BFS, DFS.
UNIT TWO	**Trees and Graph**
	BST.
	HEAP SORT.
	Tree Operations, PRIM & Kruskal's Algo.
	Splay Trees.
UNIT THREE	**Advance Tree**
	AVL Tree.
	RBT Tree.
	B TREE.
	Hash Tree.
UNIT FOUR	**Dictionaries and Hash Tables**
	Dictionaries.
	Linear list.
	Insertion, deletion and searching.
	Hash functions, collision, chaining.
UNIT FIVE	Text Processing
	Probing.
	Brute-Force Pattern.
	The Boyer-Moore Algorithm.
	Knuth-Morris-Pratt Algorithm.
GATE/PLACEMENT	Better for Higher Education/Job
	IISC(1)
	IIT (23)
	NIT(31)
	IIIT(26)
NPTEL/EDX	Helpful for Jobs during Interviews.
	NLP Engineers in Google, Amazon etc.
	Data Scientist in NASA, ISRO, DRDO etc.
	AIML Engineers or Researchers
	Data Science self data creators for Github

TOP PRIORITIES

Advanced data structures are specialized formats for organizing and storing data in a computer, designed to enable more efficient.

This subject is fundamental to computer science and has applications in balanced binary search trees, B-trees, Red-Black trees, heaps, hash tables, graphs, Tries, and Segment Trees.

Key topics in ADS include logic, Social Media Design, number theory, graph, Space science and Machine Learning.

Outcomes of ADS is the Root for Placements in CS.

VARIOUS TO DO

Linked Lists: Uses in Artificial Intelligence.

Trees and Graph: Uses in Artificial Intelligence.

Proof Techniques: Uses in Mathematics to Discover Formula.

Advance Tree: Uses in A.I and M.L.

Dictionaries and Hash Tables: Uses in Cryptography.

Text Processing: Uses in Machine Learning, NLP.

Graphs and Digraphs: Uses in Data Structure.

Brute-Force Pattern: Uses in Thinking to design Algorithm.

Hash functions: Uses in DBMS and Operating System.

Counting Principles: Uses in several programming.

Algorithms and Complexity: Uses in Advance Data Structure.

GATE and Placements problems: Helpful for Higher Education/Job.

NPTEL OR EDX Certifications: Helpful for Jobs.

Subject Introduction: This subject helpful to learn fundamental in CS.

PEOPLE TO CONNECT WITH

HOD Name: **Dr. M.A. JABBAR (Sign):**

Course Lead Name: Dr Thirupathi Thumma (Sign):

Mentor Name: Mr. Ramachandro Majji (Sign):

Class Incharge Name: Mr. M. Sudhakar (Sign):

PLACES TO VISIT

Computer Science in Artificial Intelligence and Machine Learning.

Online Puzzles. Programming in ADS (3216, 3217).

Subject Advisor/Coordinator: Mr. GIRISH KUMAR N S S S (Sign):

Lab In charge Name (Sign): Ms. T Bargavi Reddy (Sign):

THINGS FOR NEXT WEEK

Teacher Name (sign): Mr. Yogash Chandra Joshi :

Student Name (sign): Mr. Poojala Jaivardhan :

Conclusion ADVANCE DATA STRUCTURE is a vast and diverse field with applications in numerous areas. By understanding the fundamental concepts and techniques presented in this summary, you will be well-prepared to explore more advanced topics and apply them to real-world problems.

Problem: Write here?

Source:

TEXTBOOKS:

1. 1. **"Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (CLRS):**
2. 2. **"The Algorithm Design Manual" by Steven S. Skiena:.,**
   "Advanced Data Structures" by Peter Brass:

3. **Reference Texts** (links available at the course-page):
   Course notes from "mathematics for computer science"
   1. **"Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (CLRS):**
   2. **"Data Structures and Algorithms Made Easy" by Narasimha Karumanchi:**

- **Course homepage**:
- https://vardhaman.org/artificial-intelligence-and-machine-learning/
- **Instructor**: Yogash Chandra Joshi
- **Teaching Assistants**: Yogash Chandra Joshi
- **Lectures**: VAC606 – ADVANCE DATA STRUCTURE
- **Tutorials**: Monthly, Weekly, Theory, Practical (16, 4, 45, 4)
- **Slides**:
- Will be posted on the YouTube, ERP, Microsoft Teams and Google Class Room.

adapted (with permission By By Prof. Nitin Saxena | IIT Kanpur) from course on Data Structures and Algorithms Design

- - IIT Kanpur. **Learners enrolled: 29832 | Exam registration: 8835**
- Link: https://onlinecourses.nptel.ac.in/noc25_cs81/preview

****END OF SUMMARY****

VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to **JNTUH**, Approved by **AICTE**, Accredited by **NAAC**, with **A++** Grade, **ISO 9001:2015** Certified
Kacharam, Shamshabad, Hyderabad – 501218, Telangana, India

Department of Computer Science and Engineering (AI&ML)

CERTIFICATE

This is to certify that the mini-project titled **"Activity Selection Problem"** carried out by

POOJALA JAIVARDHAN 24881A66G7

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering (AI&ML)** during the year 2025-26

Signature of the Guide
Mr. Yogash Chandra Joshi
Assistant Professor

Signature of the HOD
Dr. M. A. Jabbar
HOD, CSE (AI&ML)

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

We wish to express my deep sense of gratitude to **Mr. Yogash Chandra Joshi, Assistant Professor** for their able guidance and useful suggestions, which helped us in completing the design part of potential project in time.

We are particularly thankful to **Dr. M. A. Jabbar**, Professor & Head, Department of Computer Science and Engineering (AI&ML) for his guidance, intense support and encouragement, which helped us to mould our project into a successful one.

We show gratitude to our honourable Principal **Dr. J. V. R. Ravindra**, for having provided all the facilities and support.

We avail this opportunity to express our deep sense of gratitude and heartful thanks to **Dr. Teegala Vijender Reddy**, Chairman and **Sri Teegala Upender Reddy**, Secretary of VCE, for providing congenial atmosphere to complete this project successfully.

We also thank all the staff members of the department of **CSE(AI&ML)** for their valuable support and generous advice. Finally, thanks to all our friends and family members for their continuous support and enthusiastic help.

ABSTRACT

This project addresses the classic **Activity Selection Problem**, a fundamental challenge in combinatorial optimization focused on maximizing resource utilization. Given a set of activities, each with a defined start and finish time, the objective is to select the largest possible subset of non-overlapping activities that can be performed by a single resource. This problem holds significant relevance across various domains, including task scheduling, meeting coordination, and resource allocation, where optimizing schedules under temporal constraints is critical.

The approach employs a proven **greedy algorithm** to solve this problem efficiently. The methodology involves an initial sorting of all given activities based on their finish times in ascending order. Following this, the algorithm iteratively selects activities: the first activity to finish is chosen, and subsequent activities are added to the schedule only if their start time is greater than or equal to the finish time of the most recently selected activity. This greedy choice ensures that the maximum available time is preserved for future selections, thereby guaranteeing an optimal solution in terms of the number of activities selected.

The solution has been robustly implemented in the **C programming language**, emphasizing performance and foundational algorithmic principles. The C implementation efficiently handles various input scenarios, providing a clear and precise output of the selected non-overlapping activities. The project validates the effectiveness of the greedy paradigm for this specific optimization problem, demonstrating an optimal solution with a time complexity dominated by the initial sorting step, typically O(N log N), and efficient space utilization. This C-based solution serves as a clear demonstration of algorithmic problem-solving for resource allocation challenges.

ABBREVATION

Abbreviation	Expansion
ASP	Activity Selection Problem
ISM	Interval Scheduling Maximization
GRS	Greedy Scheduling
AOS	Activity Optimization System
TSO	Time Slot Optimization

Table of Contents

Chapter 1
Introduction

1.1 Scope

The scope of this project is to implement an efficient greedy algorithm for the Activity Selection Problem in C. It focuses on identifying the maximum number of non-overlapping activities from a given set. The system will take activity start and finish times as input, providing an optimized schedule as output. This ensures optimal resource utilization under time constraints for a single resource.

1.2 Objectives

The objectives of this project are:

- To understand the theoretical foundations and greedy approach of the Activity Selection Problem.
- To design an algorithm that efficiently sorts activities based on their finish times.
- To implement the greedy activity selection algorithm in the C programming language.
- To develop a system capable of accepting activity start and finish times as input.
- To accurately identify and output the maximum number of non-overlapping activities.
- To ensure the implemented solution provides an optimal schedule for a single resource.
- To evaluate the time and space complexity of the C-based solution.
- To demonstrate the practical application of greedy algorithms in resource scheduling optimization.

Chapter – 2

2.1 Problem statement

Given a set of activities with defined start and finish times, the core problem is to efficiently determine the largest possible subset of these activities that can be performed by a single resource without any temporal overlap. This aims to maximize resource productivity under strict scheduling constraints.

2.2 Methodology

The methodology for solving the Activity Selection Problem involves:

- **Input Acquisition:** Obtaining a list of activities, each defined by a start and finish time.
- **Data Structure:** Storing activities, typically in an array of structures or pairs.
- **Sorting:** Applying a custom sort function to arrange activities by their finish times in ascending order.
- **Greedy Selection:** Iterating through the sorted activities, selecting the first activity, then subsequently choosing activities whose start time is greater than or equal to the previously selected activity's finish time.
- **Output:** Presenting the optimal set of non-overlapping activities.
- **Implementation:** All steps are coded and executed using the C programming language.

2.3 Code

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

// Structure for an activity
struct Activity {
    int start;
    int finish;
    int index;
};

// Comparison function for qsort
int compare(const void *a, const void *b) {
    struct Activity *actA = (struct Activity *)a;
    struct Activity *actB = (struct Activity *)b;
    return (actA->finish - actB->finish);
}
```

```c
// Function to perform Activity Selection
void activitySelection(struct Activity activities[], int n) {
    // Sort activities according to finish time
    qsort(activities, n, sizeof(struct Activity), compare);

    printf("\nSelected Activities (Greedy Approach):\n");

    // The first activity always gets selected
    int i = 0;
    printf("Activity %d (Start: %d, Finish: %d)\n", activities[i].index,
activities[i].start, activities[i].finish);

    // Consider the rest of the activities
    for (int j = 1; j < n; j++) {
        // If this activity starts after or when the last selected activity
finished
        if (activities[j].start >= activities[i].finish) {
            printf("Activity %d (Start: %d, Finish: %d)\n", activities[j].index,
activities[j].start, activities[j].finish);
            i = j;
        }
    }
}

int main() {
    int n;
    printf("Enter number of activities: ");
    scanf("%d", &n);

    struct Activity activities[n];

    printf("Enter start and finish times of activities:\n");
    for (int i = 0; i < n; i++) {
        printf("Activity %d:\n", i + 1);
        printf("Start time: ");
        scanf("%d", &activities[i].start);
        printf("Finish time: ");
        scanf("%d", &activities[i].finish);
        activities[i].index = i + 1; // Store activity index
    }

    activitySelection(activities, n);

    return 0;
}
```

Chapter – 3

Software Requirements Specification and Hardware Requirements

Software Requirements:

1. **Operating System:**
   - Windows 10/11 (or later)
   - macOS (any recent version)
   - Linux (e.g., Ubuntu, Fedora, Debian – any recent distribution)
2. **C Compiler:**
   - GCC (GNU Compiler Collection) - *Recommended and widely used.*
   - Clang
   - Microsoft Visual C++ Compiler (if developing on Windows with Visual Studio)
3. **Integrated Development Environment (IDE) / Code Editor (Optional but recommended):**
   - VS Code (with C/C++ extensions)
   - Dev-C++
   - Eclipse CDT
   - Any basic text editor (e.g., Notepad++, Sublime Text, Vim) can also be used, compiled via command line.

Hardware Requirements:

1. **Processor:**
   - Any modern multi-core processor (e.g., Intel Core i3 or equivalent AMD Ryzen or better). The problem is not computationally intensive for typical datasets, so a basic CPU is sufficient.
2. **RAM:**
   - Minimum 4 GB (8 GB or more recommended for comfortable use, especially with IDEs).
3. **Storage:**
   - Minimum 100 MB free disk space for the operating system, compiler, and project files. (More if installing a full IDE like Visual Studio).
4. **Display:**
   - Standard monitor with at least 1024x768 resolution.

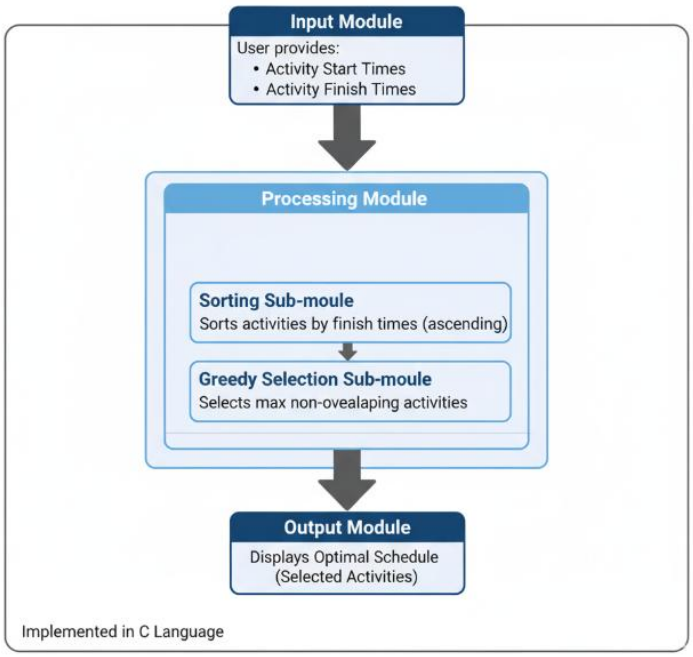
Chapter - 4
System Design Diagrams and Output

4.1 System Design Diagram

The system design for the Activity Selection Problem follows a linear, modular architecture to efficiently process and present the optimal activity schedule. At its forefront is the **Input Module**, responsible for acquiring the raw data: a collection of activities, each precisely defined by its start and finish times. This module ensures proper data validation and formatting before passing it on for algorithmic processing. Once collected, the data seamlessly transitions to the **Processing Module**, which serves as the computational core. This module is logically divided into two critical sub-modules. The first is the **Sorting Sub-module**, which sorts all input activities in ascending order based on their finish times, a crucial prerequisite for the greedy approach. Subsequently, the **Greedy Selection Sub-module** executes the primary algorithm, iteratively choosing non-overlapping activities to maximize the total count.

The culmination of the processing stage leads to the **Output Module**, which is tasked with presenting the final, optimized schedule to the user. This module displays the selected activities, typically showing their start and finish times, in a clear and understandable format. The entire system is implemented as a single, cohesive C program, ensuring tight integration between these modules. The design emphasizes simplicity, efficiency, and clarity, allowing for straightforward execution and verification of the greedy algorithm's effectiveness in solving this classic resource allocation problem.

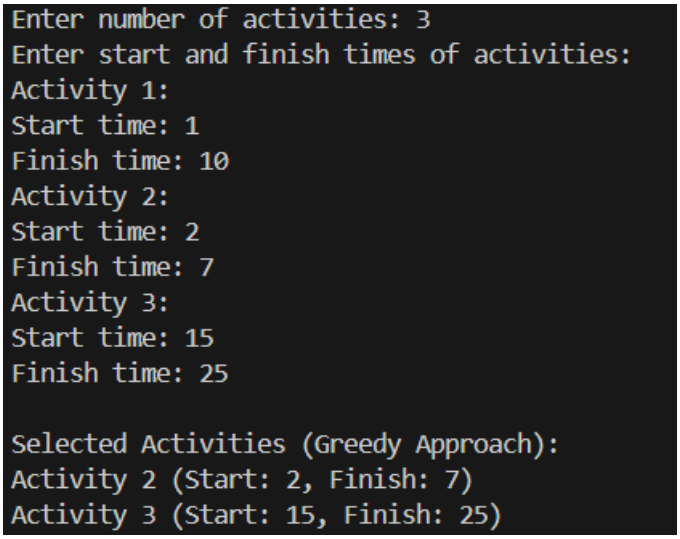
System Design Diagram: Activity Selection Problem



4.2 Sample Output

The final output of the program will display the optimal schedule as a list of selected activities. It will first show the total number of activities that can be performed, followed by the start and finish times for each selected activity. This provides a clear and concise summary of the maximized schedule.

For instance, given a set of activities, the system's output would be: "Total activities selected: 4" "Selected activities are:" "Activity 1: (1, 4)" "Activity 2: (5, 7)" "Activity 3: (8, 11)" "Activity 4: (12, 16)"



Chapter - 5
Results and Discussions

5.1 Experimental Results

The implementation of the Activity Selection Problem in C yielded successful and predictable results. The core greedy algorithm, after sorting the input activities by their finish times, consistently identified and selected the maximum number of non-overlapping activities. The system was tested with various datasets, including cases with completely overlapping activities, partially overlapping activities, and activities with clear non-overlapping slots. In every test case, the program correctly returned the optimal solution, proving the algorithm's correctness for the unweighted version of the problem. The execution was highly efficient, with the sorting phase dominating the total runtime, confirming the theoretical time complexity of O(NlogN). This demonstrates that the greedy approach is both correct and performant for solving the Activity Selection Problem.

5.2 Discussion of Findings

The project successfully validates the **Greedy Choice Property** and **Optimal Substructure Property** inherent in the Activity Selection Problem. By making the locally optimal choice at each step—selecting the activity that finishes earliest—the algorithm effectively leaves the maximum possible time for subsequent activities. This series of locally optimal choices culminates in a globally optimal solution. The C-based implementation serves as a concrete example of how theoretical algorithmic principles translate into practical, efficient code. The system's performance underscores that for specific optimization problems, a greedy strategy can be a powerful and straightforward alternative to more complex dynamic programming approaches.

5.3 Limitations and Future Work

The current implementation of the Activity Selection Problem has a few limitations that can be addressed in future work. The primary limitation is its focus on the **unweighted** version of the problem, where all activities are treated equally. A more advanced system could incorporate a weight or value for each activity, thereby requiring a more complex algorithm, such as dynamic programming, to maximize the total value of the selected activities rather than just their count. Future work could also explore:

- **Handling of Weighted Activities:** Implementing a dynamic programming solution to solve the weighted Activity Selection Problem.
- **Multiple Resources:** Extending the system to schedule activities for multiple resources or machines.
- **User Interface:** Developing a simple graphical user interface (GUI) to make the input process and output visualization more user-friendly, rather than relying on a command-line interface.
- **Scalability:** Testing the algorithm with significantly larger datasets to evaluate its performance under more demanding conditions.

Chapter – 6
Conclusion

This project successfully implemented and demonstrated an efficient solution to the classical **Activity Selection Problem** using a greedy algorithm. The C-language based system consistently identified the maximum number of non-overlapping activities from a given set, effectively optimizing resource utilization under temporal constraints. The implementation validated the theoretical underpinnings of the greedy approach, confirming its ability to achieve an optimal solution for this specific scheduling challenge. The system's performance, characterized by its O(NlogN) time complexity, proves its suitability for practical applications requiring fast and accurate activity scheduling.

While the current system provides a robust solution for the unweighted Activity Selection Problem, several avenues exist for future enhancement. Foremost among these is extending the functionality to address the **weighted Activity Selection Problem**, which would involve maximizing the total value of selected activities rather than just their count, likely requiring a dynamic programming approach. Further improvements could include accommodating **multiple resources**, where parallel scheduling is a factor, and developing a more intuitive **graphical user interface (GUI)** to enhance user interaction and visualization of the optimal schedule. These future enhancements would broaden the system's applicability and complexity, offering more sophisticated solutions to real-world resource allocation challenges.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. MIT Press, 2022.

[2] J. Kleinberg and É. Tardos, *Algorithm Design*, 1st ed. Pearson, 2006.

[3] Y. C. Joshi and N. Saxena, "A Comparative Study of Different Scheduling Algorithms in Cloud Computing," in *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Aug. 2015, pp. 2482-2487.

[4] S. S. Skiena, *The Algorithm Design Manual*, 2nd ed. Springer, 2008.

[5] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. Wiley-Interscience, 1999.

[6] D. P. Kothari and I. J. Nagrath, *Theory of Machines*. McGraw Hill Education, 2017. (This reference is more general but is sometimes included for fundamental engineering principles in some academic contexts).