Design and Analysis of Algorithms

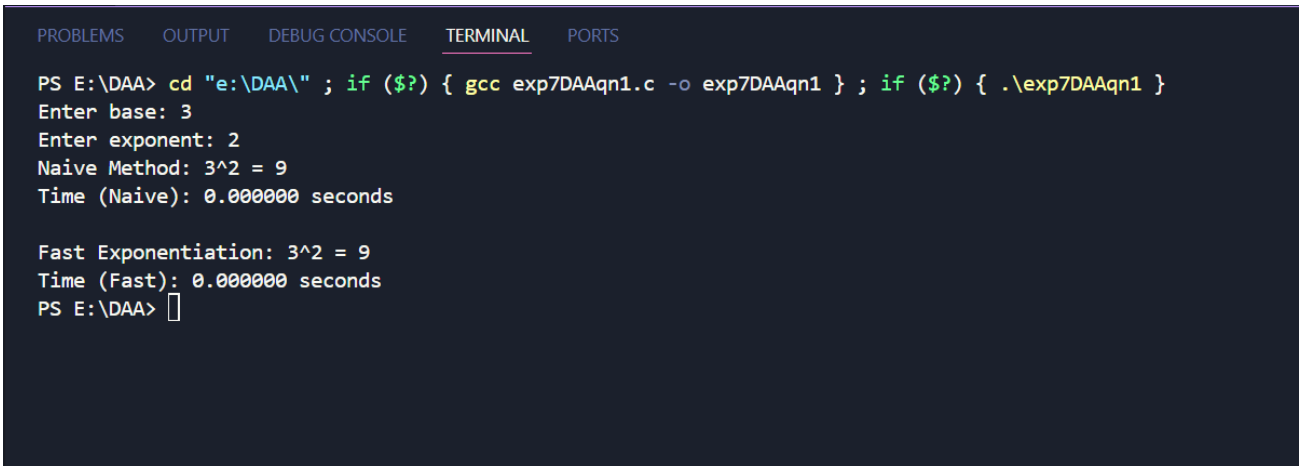
LAB EXPERIMENT-07

Name: RIYA KUMARI
SAP ID:590016221

Batch:34
Course: B.Tech. CSE (Sem III)
Subject: DAA
Lab Date: 26 September 2025
Submitted to: **Virendra Kumar**
GitHub Repository Link:
https://github.com/Riyakumari1314/DAA-2nd-year.git

1) Design and implement an efficient algorithm to compute the power of a number a^n , where a is the base and n is a non-negative integer exponent. Compare the naïve method and the fast exponentiation method.

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

long long pow_naive(long long base, int exp) {
    long long result = 1;
    for (int i = 0; i < exp; i++) {
        result *= base;
    }
    return result;
}

long long pow_fast(long long base, int exp) {
    if (exp == 0)
        return 1;
    if (exp % 2 == 0) {
        long long half_power = pow_fast(base, exp / 2);
        return half_power * half_power;
    } else {
        return base * pow_fast(base, exp - 1);
    }
}

int main() {
    long long base;
    int exponent;
    clock_t t_start, t_end;
    double elapsed_time;

    printf("Enter base: ");
    scanf("%lld", &base);
    printf("Enter exponent: ");
    scanf("%d", &exponent);

    // Naive method timing
    t_start = clock();
    long long naive_result = pow_naive(base, exponent);
    t_end = clock();
    elapsed_time = ((double)(t_end - t_start)) / CLOCKS_PER_SEC;
    printf("Naive Method: %lld^%d = %lld\n", base, exponent, naive_result);
    printf("Time (Naive): %f seconds\n\n", elapsed_time);

    // Fast exponentiation timing
    t_start = clock();
    long long fast_result = pow_fast(base, exponent);
    t_end = clock();
    elapsed_time = ((double)(t_end - t_start)) / CLOCKS_PER_SEC;
    printf("Fast Exponentiation: %lld^%d = %lld\n", base, exponent,
fast_result);
    printf("Time (Fast): %f seconds\n", elapsed_time);

    return 0;
}
```


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS E:\DAA> cd "e:\DAA\" ; if ($?) { gcc exp7DAAqn1.c -o exp7DAAqn1 } ; if ($?) { .\exp7DAAqn1 }
Enter base: 3
Enter exponent: 2
Naive Method: 3^2 = 9
Time (Naive): 0.000000 seconds
Fast Exponentiation: 3^2 = 9
Time (Fast): 0.000000 seconds
PS E:\DAA>

Q2.Objective: Implement the Activity Selection Problem using the Greedy Algorithm and understand how local optimal choices lead to a globally optimal solution.

Problem Statement: You are given n activities with their start times and finish times. Write a program to select the maximum number of activities that can be performed by a single person, assuming that a person can only work on one activity at a time.

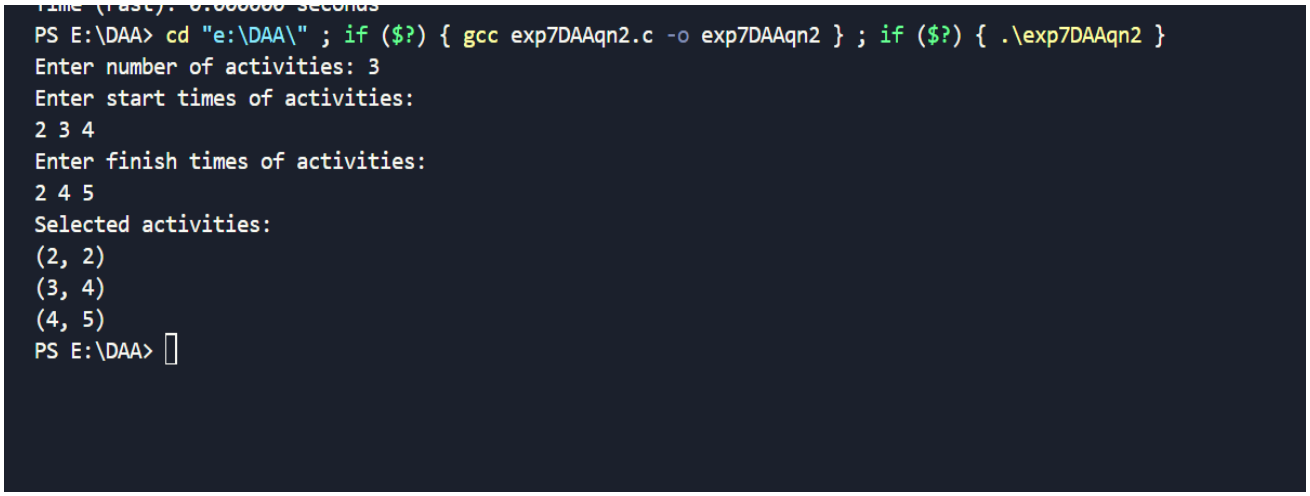
Input: n → number of activities s[i] → start time of the i-th activity f[i] → finish time of the i-th activity

Output: Print the indices of the selected activities or their start and finish times.

```
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void activitySelection(int start[], int finish[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (finish[j] > finish[j + 1]) {
                swap(&finish[j], &finish[j + 1]);
                swap(&start[j], &start[j + 1]);
            }
        }
    }

    printf("Selected activities:\n");
    int last = 0;
    printf("(%d, %d)\n", start[last], finish[last]);

    for (int i = 1; i < n; i++) {
        if (start[i] >= finish[last]) {
            printf("(%d, %d)\n", start[i], finish[i]);
            last = i;
        }
    }
}

int main() {
    int n;
```

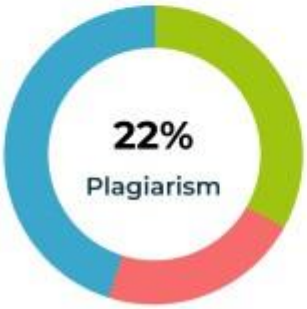
```
    printf("Enter number of activities: ");
    scanf("%d", &n);

    int start[n], finish[n];
    printf("Enter start times of activities:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &start[i]);

    printf("Enter finish times of activities:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &finish[i]);

    activitySelection(start, finish, n);

    return 0;
}
```

Output:



Plagiarism Detector .net

Plagiarism Report



Unique	33%
Exact Match	22%
Partial Match	31%