Experiment 9

WAP to Perform the Comparative analysis of Naïve algorithm and greedy algorithm to solve the Job sequencing problem.

<u>Naïve algorithm</u>

Program:-

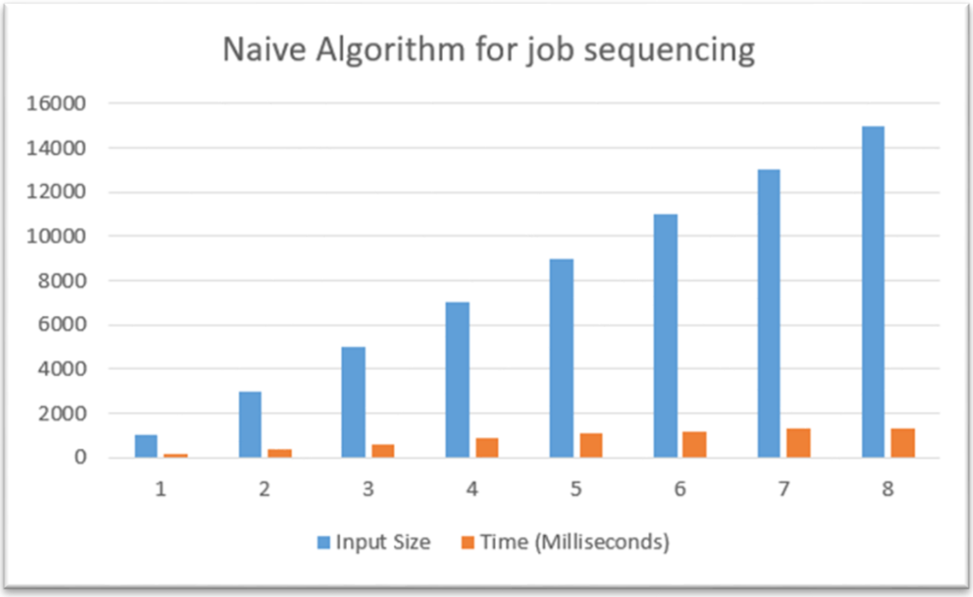
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
struct Job {
    int id;
    int deadline;
    int profit;
};
int compareJobs(const void *a, const void *b) {
    return ((struct Job *)b)->profit - ((struct Job *)a)->profit;
}
int main() {
    int i, n, max_deadline = 0;
    struct Job jobs[50000];  // Adjust array size if needed
    printf("Enter the total number of jobs: ");
    scanf("%d", &n);
    srand(time(NULL));  // Seed the random number generator
    // Generate random job details
    for (i = 0; i < n; i++) {
        jobs[i].id = i + 1;
        jobs[i].deadline = rand() % 10000;  // Random deadline between 1 and n
        jobs[i].profit = rand() % 10000;     // Random profit between 0 and 999
        max_deadline = (jobs[i].deadline > max_deadline) ? jobs[i].deadline : max_deadline;
    }
    // Sort jobs based on profit in descending order
    qsort(jobs, n, sizeof(struct Job), compareJobs);
    int result[max_deadline];
    for (i = 0; i < max_deadline; i++) {
        result[i] = -1;  // Initialize with -1 to indicate no job scheduled
    }
    clock_t start = clock();
    for (i = 0; i < n; i++) {
        for (int j = jobs[i].deadline - 1; j >= 0; j--) {
            if (result[j] == -1) {
                result[j] = jobs[i].id;
                break;
            }
        }
    }
    printf("\nSelected jobs (in order of execution):\n");
    for (i = 0; i < max_deadline; i++) {
        if (result[i] != -1) {
            printf("Job %d (deadline %d, profit %d)\n", result[i], jobs[result[i]].deadline, jobs[result[i]].profit);
        }
    }
    clock_t end = clock();
    double time_taken = ((double)(end - start)*1000) / CLOCKS_PER_SEC;
    printf("\nTime taken to execute algorithm: %lf miliseconds\n", time_taken);
    return 0;
}
```

Result Analysis and Discussion:

This experiment has been conducted in a 64-bit system with 16 GB RAM and Processor 12th Gen Intel(R) Core (TM) i5-12500H 3.10 GHz. The algorithm is implemented in C programming language in Visual Studio Code 1.85.1 Code Editor. In this experiment the algorithm to solve the Job sequencing problem of an array of size “n” using Naïve algorithm has been implemented and executed for different value of n. During this experiment for different value of n, the time taken by the algorithm has been measured and tabulated as shown in table below.

Input Size	Time (Milliseconds)
1000	125
3000	341
5000	569
7000	883
9000	1076
11000	1145
13000	1295
15000	1339

The graph shown below is the plot of input n and the time in milliseconds taken by the algorithm while running on a system recorded in table above.



Based on the above table and graph it is clearly seen that the size of array n has linear relationship with the time taken by the system to sort an array.

Greedy algorithm

Program:-

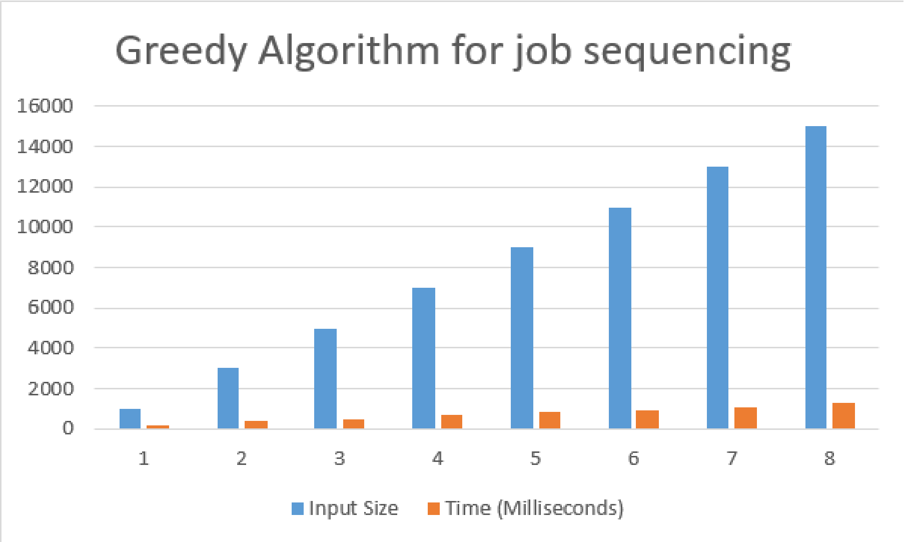
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
struct Job {
    int id;
    int deadline;
    int profit;
};
int compareJobs(const void *a, const void *b) {
    return ((struct Job *)b)->profit - ((struct Job *)a)->profit;
}
int main() {
    int i, n, max_deadline = 0;
    struct Job jobs[50000];
    printf("Enter the total number of jobs: ");
    scanf("%d", &n);
    srand(time(NULL));
    for (i = 0; i < n; i++) {
        jobs[i].id = i + 1;
        jobs[i].deadline = rand() % 10000;  n
        jobs[i].profit = rand() % 10000;
        max_deadline = (jobs[i].deadline > max_deadline) ? jobs[i].deadline : max_deadline;
    }

    qsort(jobs, n, sizeof(struct Job), compareJobs);
    int result[max_deadline];
    for (i = 0; i < max_deadline; i++) {
        result[i] = -1;
    }
    clock_t start = clock();
    for (i = 0; i < max_deadline; i++) {
        int max_profit_index = -1;
        int max_profit = -1;
        for (int j = 0; j < n; j++) {
            if (jobs[j].deadline >= i + 1 && jobs[j].profit > max_profit) {
                max_profit_index = j;
                max_profit = jobs[j].profit;
            }
        }

        if (max_profit_index != -1) {
            result[i] = jobs[max_profit_index].id;
            jobs[max_profit_index].deadline = 0;
        }
    }

    printf("\nSelected jobs (in order of execution):\n");
    for (i = 0; i < max_deadline; i++) {
        if (result[i] != -1) {
            printf("Job %d (deadline %d, profit %d)\n", result[i], jobs[result[i]].deadline, jobs[result[i]].profit);
        }
    }

    clock_t end = clock();
    double time_taken = ((double)(end - start)*1000) / CLOCKS_PER_SEC;      printf("\nTime taken to execute algorithm: %lf miliseconds\n", time_taken);
    return 0;
}
```

Result Analysis and Discussion:

This experiment has been conducted in a 64-bit system with 16 GB RAM and Processor 12th Gen Intel(R) Core (TM) i5-12500H 3.10 GHz. The algorithm is implemented in C programming language in Visual Studio Code 1.85.1 Code Editor. In this experiment the algorithm to solve the Job sequencing problem of an array of size "n" using greedy algorithm has been implemented and executed for different value of n. During this experiment for different value of n, the time taken by the algorithm has been measured and tabulated as shown in table below.

Input Size	Time (Milliseconds)
1000	149
3000	376
5000	486
7000	691
9000	837
11000	926
13000	1091
15000	1286

The graph shown below is the plot of input n and the time in milliseconds taken by the algorithm while running on a system recorded in table above.



Based on the above table and graph it is clearly seen that the size of array n has linear relationship with the time taken by the system to sort an array.

Conclusion:

In this experiment, it has been found that the size of input “n” has linear relationship with the time taken by the system to solve the Job sequencing problem. So, Greedy algorithm is better than Naive algorithm to solve the Job sequencing problem.