Course: CSE 373- Design & Analysis of Algorithm

Group No: 05, Section: 12

Project Title: Analysis of Algorithm for Job Scheduling Problem

Course Instructor:

Mirza Mohammad Lutfe Elahi

Senior Lecturer,
Department of Electrical and Computer Engineering, North South University.

Group Members:

1. Farhan Ishrak Tahmid
   NSU ID: 2031458642

2. Farhana Akbar
   NSU ID: 2012482042

3. Imtiaz Shahriar
   NSU ID: 2012608642

Analysis of Algorithm for Job Scheduling Problem

Introduction:

The job scheduling problem is a well-known problem in computer science and operations research. The problem involves scheduling a set of jobs containing a deadline and profits from it in such a way as to minimize the total completion time on the deadline.

The problem is usually formulated as follows:

Given a set of jobs J = {jobid, deadline, profit}, where each job j has a deadline and a profit. In our project we randomly generated jobs with maximum and minimum values.

Our goal here is to find a sequence of jobs that will maximize the profit. We used Three different approaches to solve this algorithm, and they are Greedy Algorithm, Brute Force, and Dynamic Programming Algorithm.

Analysis of Algorithms:

Greedy Algorithm:

The algorithm's approach is to sort the jobs in descending order of profit and greedily schedule them based on their deadlines. It iterates through the sorted jobs list, finds the nearest available slot to the deadline, and schedules the job if a slot is available. By prioritizing jobs with higher profits, it aims to maximize the total profit while ensuring that each job meets its deadline.

The whole process and analysis of the processes are given below:

1. Sorting the jobs: Sorting the jobs based on their profit takes O(n log n) time complexity, where 'n' is the number of jobs. This is because sorting a list of 'n' elements typically requires O(n log n) operations.
2. Initializing variables: These operations take constant time and do not significantly contribute to the overall time complexity.
3. Initializing slots: The algorithm finds the maximum deadline among the jobs and initializes a slots list of size max_deadline + 1. Hence, it takes O(max_deadline) time complexity to initialize the slots.
4. Scheduling the jobs: This is the main part of the algorithm. It iterates through the sorted jobs list, which takes O(n) time complexity. For each job, it tries to find an available slot closest to the deadline, which can be done in O(1) time if using an array to represent the slots. Hence, scheduling all the jobs takes O(n) time complexity.

Overall, the time complexity of the Job Scheduling Algorithm can be approximated as **O(n log n),** dominated by the sorting operation. The remaining operations have linear time complexity, contributing less to the overall complexity.

Brute Force Algorithm:

The algorithm's approach is to generate all possible permutations of the jobs and compute the profit for each permutation. It checks the slot availability for each job in the current permutation and updates the maximum profit and job sequence accordingly. The algorithm exhaustively searches all permutations to find the one with the maximum profit. However, the time complexity grows rapidly with the number of jobs due to the factorial nature of generating all permutations.

The whole process and analysis of the processes are given below:

1. Generating permutations: The algorithm generates all possible permutations of the given jobs using the itertools.permutations(a python library) function. The number of permutations is given by n!, where 'n' is the number of jobs. Generating all permutations has a time complexity of O(n!).
2. Initializing variables: These operations take constant time and do not significantly contribute to the overall time complexity.
3. Traversing through permutations and checking slot availability: The algorithm iterates through each permutation of jobs, which takes O(n!) time complexity. For each permutation, it checks the slot availability for each job by iterating from the deadline to 1. This nested loop has a time complexity of O(n * m), where 'n' is the number of jobs and 'm' is the maximum deadline among the jobs.
4. Updating maximum profit and job sequence: The algorithm keeps track of the maximum profit and the job sequence with maximum profit. Updating these values has a constant time complexity.

Overall, the time complexity of the Brute Force Algorithm can be approximated as **O(n! * n * m),** where 'n' is the number of jobs and 'm' is the maximum deadline among the jobs. The dominant factor is the generation of all permutations, which has a factorial time complexity and can be computationally expensive for large values of 'n'. The algorithm exhaustively searches all permutations and selects the one with the maximum profit, ensuring that each job is considered in different sequences.

Dynamic Programming:

The algorithm's approach is to sort the jobs based on profits and iterate through them in a bottom-up manner. It finds the earliest available slot for each job and updates the dynamic array and job sequence lists accordingly. Finally, it calculates the maximum profit by summing the dynamic array values and constructs the job sequence with non-empty entries. The algorithm optimally schedules jobs to maximize profit while ensuring each job is performed before its deadline.

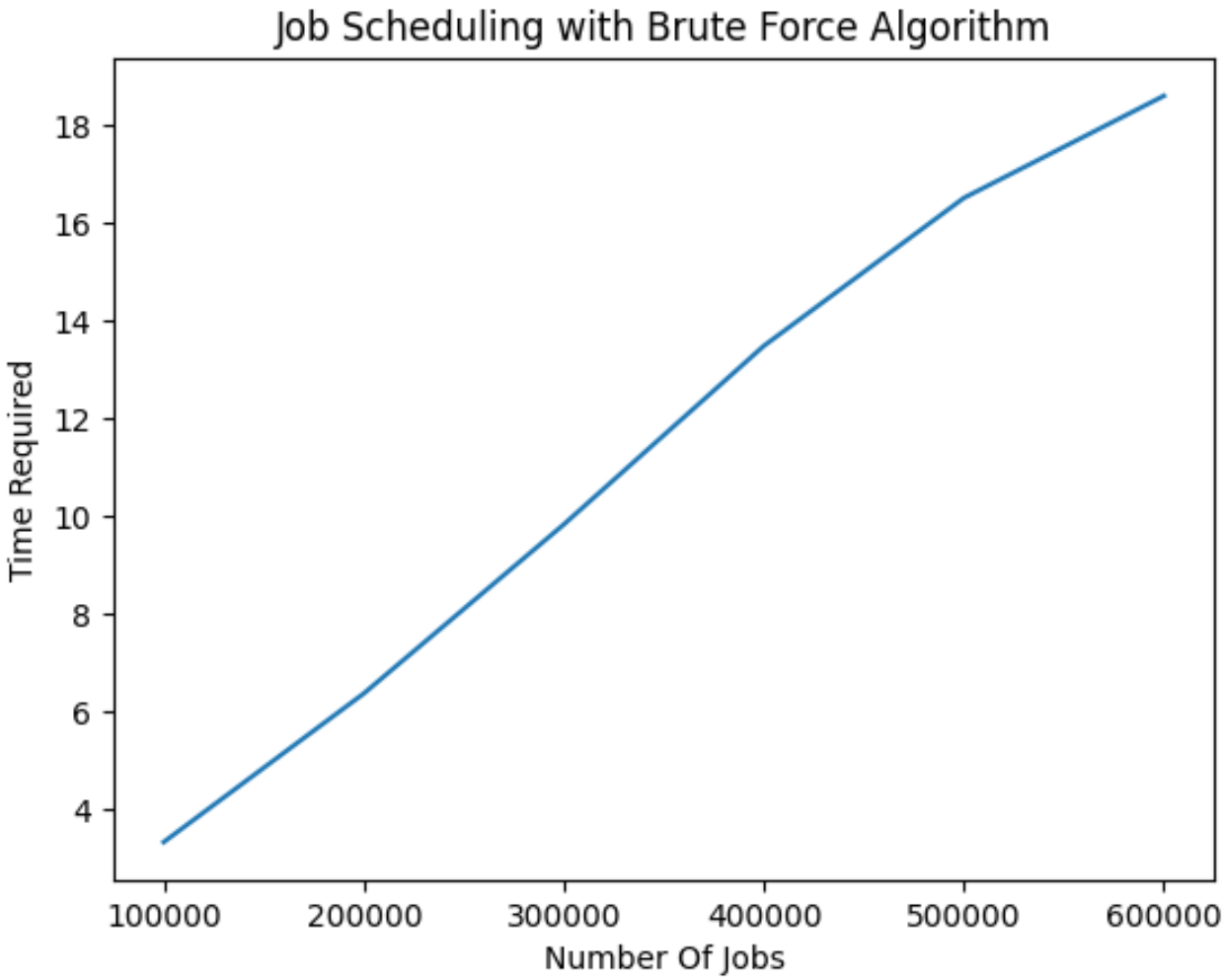
The whole process and analysis of the processes are given below:

1. Sorting jobs: The algorithm sorts the jobs in descending order based on profits using the sort function. Sorting has a time complexity of O(n log n), where 'n' is the number of jobs.
2. Finding the maximum deadline: The algorithm finds the maximum deadline among the jobs, which requires traversing through the jobs once. This operation has a time complexity of O(n).
3. Initializing dynamic array and job sequence: These operations take constant time and do not significantly contribute to the overall time complexity.
4. Traversing through the jobs: The algorithm iterates through each job once, which takes O(n) time complexity. For each job, it finds the earliest available slot by iterating from the job's deadline to 1.
5. Updating dynamic array and job sequence: The algorithm updates the dynamic array and job sequence lists based on the earliest available slot for each job. These operations take constant time and complexity.
6. Finding maximum profit and job sequence: The algorithm calculates the maximum profit by summing the values in the dynamic array, which takes O(n) time complexity. It also constructs the job sequence with non-empty entries in the job sequence list, which takes O(n) time complexity.

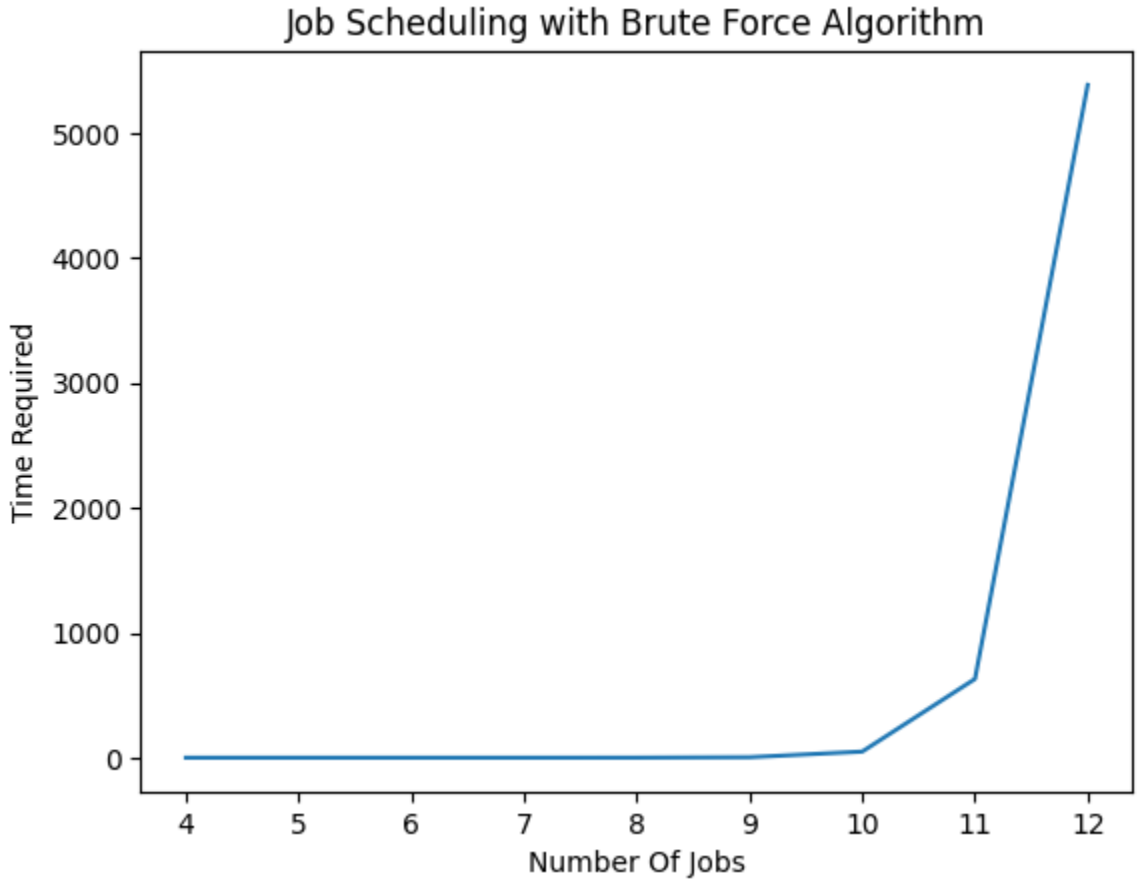
Overall, the time complexity of the Dynamic Programming algorithm can be approximated as **O(n log n),** where 'n' is the number of jobs. The dominant factors are the sorting of jobs and finding the maximum deadline, both of which take O(n log n) and O(n) time complexity, respectively. The algorithm employs a bottom-up dynamic programming approach to determine the earliest available slot for each job and compute the maximum profit and job sequence.

Experimental Results:

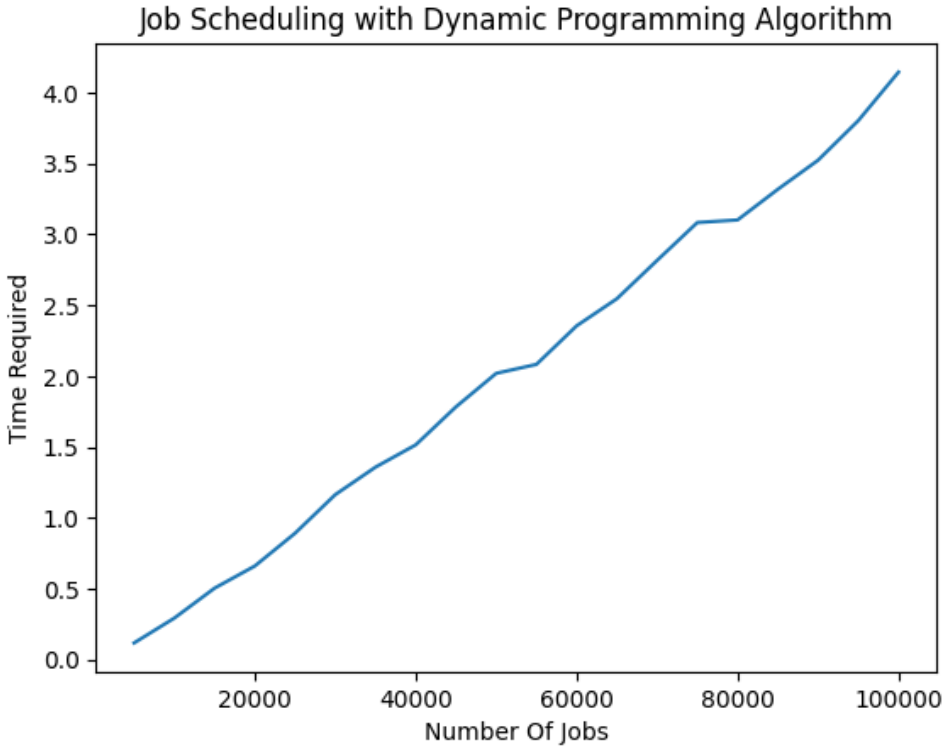
Greedy Algorithm:

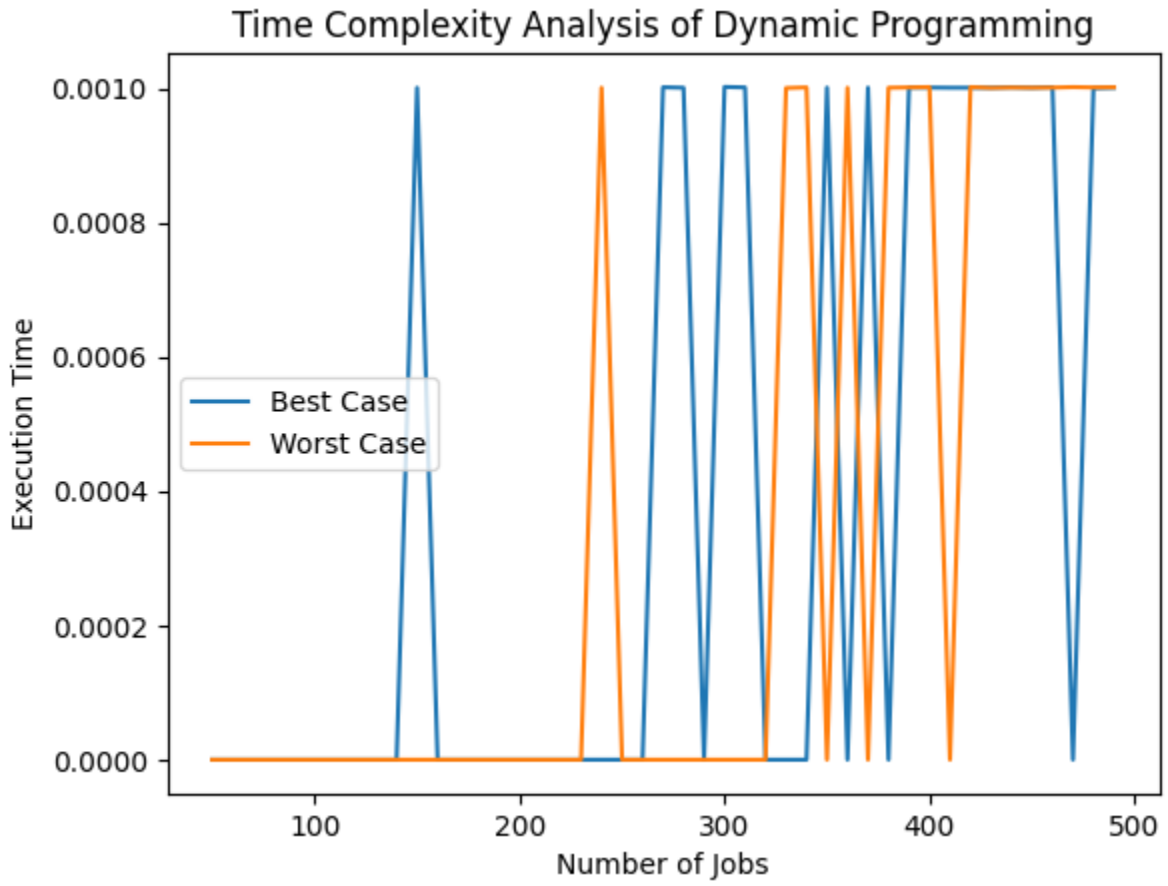


Brute Force Algorithm:



Dynamic Programming:





Conclusion:

In conclusion, we have discussed three algorithms for the job scheduling problem with associated profits and deadlines. Let's summarize each algorithm briefly:

1. Brute Force Algorithm:

- The brute force algorithm exhaustively generates all possible permutations of the jobs.
- It iterates through each permutation, simulating the execution and computing the profit.
- The algorithm selects the permutation with the maximum profit as the optimal job sequence.
- Time Complexity: The time complexity of the brute force algorithm is O(n!), where 'n' is the number of jobs. This is because it generates all permutations of the jobs.

2. Greedy Algorithm:

- The greedy algorithm sorts the jobs based on profits in descending order.
- It iterates through the sorted jobs and selects the earliest available slot for each job.
- The algorithm maximizes the profit by greedily scheduling jobs with the highest profit and available slots.
- Time Complexity: The time complexity of the greedy algorithm is O(n log n), where 'n' is the number of jobs. This is due to the initial sorting of the jobs.

3. Dynamic Programming Algorithm:

- The dynamic programming algorithm sorts the jobs based on profits in descending order.
- It iterates through the sorted jobs and determines the earliest available slot for each job using a bottom-up approach.
- The algorithm constructs a dynamic array to store the maximum profit at each deadline.
- It computes the maximum profit and constructs the optimal job sequence based on the dynamic array.
- Time Complexity: The time complexity of the dynamic programming algorithm is O(n log n), where 'n' is the number of jobs. This is due to the initial sorting of the jobs.

Overall, the dynamic programming algorithm offers an efficient solution with time complexity of O(n log n) and optimally schedules jobs to maximize profit while respecting deadlines. The greedy algorithm provides a faster solution with the same time complexity but may not always guarantee the optimal solution. The brute force algorithm explores all possible permutations but has an exponential time complexity, making it inefficient for large input sizes. The choice of algorithm depends on the trade-off between optimality and efficiency in the given job scheduling problem.

Codes:

https://github.com/FarhanTahmid/Job-Scheduling-Problem-Analysis-of-Algorithms-CSE-373