Submitted By - Sanyam Rajpal
CSCI-B-505 Applied Algorithms
(FA19 - BL - 11503)

Programming Assignment - III

WRITE-UP

I used Python language for my code because it's easy to understand and reading the text files and storing it in apt format is very simple. I printed out the sum of the optimal weights to be visible in the terminal. During my coding, I did make some syntax errors which I resolved and then ran the code perfectly.

Problem-II

I chose to work on this problem too because of its a very practical and realistic problem. Besides, it has plenty of applications in daily life and thus has a lot of utility.

I initialized a class to a job with weight, begin_time and end_time. I read the text files and stored it in a list of the list. After that, I stored each element in the list of the file as a job with weight, begin_time and end_time. Then I called the Job_Schedule function for the job.

The jobs were initially sorted using finish time. An empty array which has elements equal to the number of elements in the list(number of jobs), which will be the dynamic programming array. Then there was an iterating loop through all the jobs which were then called on for the binary search algorithm. Here bottom-up dynamic programming has been used, which is usually faster than the top-down approach although coding needs to be more precise.

Iterative Binary search(better than Recursive Binary Search) to find the latest job which doesn't conflict with the current job. The inclusive weight is calculated including the current job and the maximum of the exclusive and inclusive is stored in the dynamic programming array. The last element of the array is returned after iterating through the loop.
The user is also asked whether they want to test text file 1 or 2.

As can be seen from the code the for loop in the Job_Schedule iterates over n times where n is the number of jobs as given from the input file, so its time complexity is O(n). And for every iteration in the for loop, the Job_Binary_Search function is called iteratively. The time complexity of Binary Search is O(log(n)) where n is the number of elements in the input file. Time complexity is constant for base case O(1). So the overall time complexity is Iteration of For Loops x Binary Search Complexity which is
O(n) * O(log(n)) = **O(nlog(n))** is the Time Complexity.(where n is the number of jobs in the input file).

For the input1.txt file, the code gave output weight **811**. So, since the file was moderately, still the code ran and gave the output in less than 1 second in python. So the algorithm is pretty efficient and I have tested its correctness and tried on even sample test datasets like { (1 , 2 , 50) , (3 , 5 , 30) , (6 , 19 , 100) , (2 , 100 , 200) } and it gave output 250. The basic logic is sorting it as per finish time and then using binary search to find the current job not conflicting with the current index and giving optimum output.

For the input2.txt file, the code gave the output weight **9380**. So, this file was relatively bigger this code took some time to give the output in python. Again, this output is also correct and the use of binary search and dynamic programming makes the model very optimum. The sorting logic is already stated above.