Assignment 1

Algorithms Design and Analysis

Applegate T. Tun Oo, st126690

***Abstract*—This report investigates the interval scheduling problem through the implementation of three greedy approaches (i.e. Earliest Start Time, Earliest Finish Time, and Shortest Duration), as well as, an exhaustive algorithm to calculate the true optimal solution for smaller inputs.**

Runtime measurements show that the greedy algorithms scales at $O(nlogn)$, while the exhaustive algorithm scales at $O(n2^n)$.

I. Algorithm Descriptions

A. Earliest Start Time

THIS algorithm is built to first sort given intervals into ascending order according to their start time s_i. Interval selection then occurs in this same order as long as the selected intervals do not overlap. This method does not guarantee optimality, as it has the potential to block multiple later intervals since it prioritizes early-starting intervals.

B. Earliest Finish Time

Similar to the previous algorithm, this approach also first sorts given intervals in ascending order according to finish time f_i. Interval selection then proceeds as long as the finish time is greater than or equal to the start time of the next selected interval $f_i \geq s_j$. This algorithm is known to always produce an optimal maximum-size set of non-overlapping intervals.

C. Shortest Duration

The Shortest Duration algorithm also performs the same sorting of given intervals in ascending order according to duration (f_i - s_i). This approach has the potential to not produce optimal results because it prioritizes the selection of short intervals. This, in turn, can block more efficient long-term scheduling choices.

D. Exhaustive Search

This approach is used to calculate the true optimal solution by enumerating all possible subsets of intervals. For each subset, it checks whether the subset is feasible based on its non-overlapping quality. The largest feasible subset is returned as the optimal solution. This algorithm guarantees correctness but is only practical for smaller values of n because it becomes infeasible for larger n values due to its exponential growth.

II. Complexity Analysis

A. Greedy Algorithms (EST, EFT, SD)

The three greedy algorithms are comprised of two steps:

- The sorting of given intervals ($O(nlogn)$)
- A linear scan to select compatible intervals ($O(n)$)

This gives us an overall runtime of $O(nlogn)$

B. Exhaustive Algorithm

This type of algorithm enumerates all possible subsets of the n intervals, which is 2^n. This algorithm is only practical for small values of n because feasibility checking for each subset can take up to $O(n)$ or $O(n^2)$.

This gives us an overall runtime of $O(n2^n)$

III. Justification of T

It is important to ensure that T scales with n to produce meaningful datasets because if T were to be kept constant while n increases, unrealistic results could be produced due to the increase in overlap.

To maintain consistent overlap, the following formula was used within the interval generation algorithm: $T = alpha * n * D$

where:

- D is the maximum interval duration
- n is the number of intervals
- $alpha$ would be an iteration through the given overlap regimes [0.1, 1.0, 5.0]

Experiments were tested across these three regimes in the same manner.

A Smaller T increases overlaps and conflicts whereas, a larger T makes intervals more compatible. This ensures fair comparisons across the different values of n.

IV. Plots and Discussion

A. Solution Quality vs Optimal

The solution quality was calculated by dividing each greedy solution size with the optimal solution size. The results are as follows:

High overlap (alpha = 0.1)

- EFT/OPT: 1.000 ± 0.000
- EST/OPT: 0.809 ± 0.206
- SD/OPT: 0.442 ± 0.133

Medium overlap (alpha = 1.0)

- EFT/OPT: 1.000 ± 0.000
- EST/OPT: 0.964 ± 0.038
- SD/OPT: 0.237 ± 0.103

Low overlap (alpha = 5.0)

- EFT/OPT: 1.000 ± 0.000
- EST/OPT: 1.000 ± 0.000
- SD/OPT: 0.157 ± 0.073

As shown, the EFT approach matches the optimal solution in all regimes. EST performs well in low overlaps but becomes suboptimal as the overlap increases. SD performs poorly throughout thus, showing that it is not the most effective approach overall.

B. Greedy Runtime Growth

As shown in the plots below, the runtime growth of all three greedy algorithms increases as n increases.

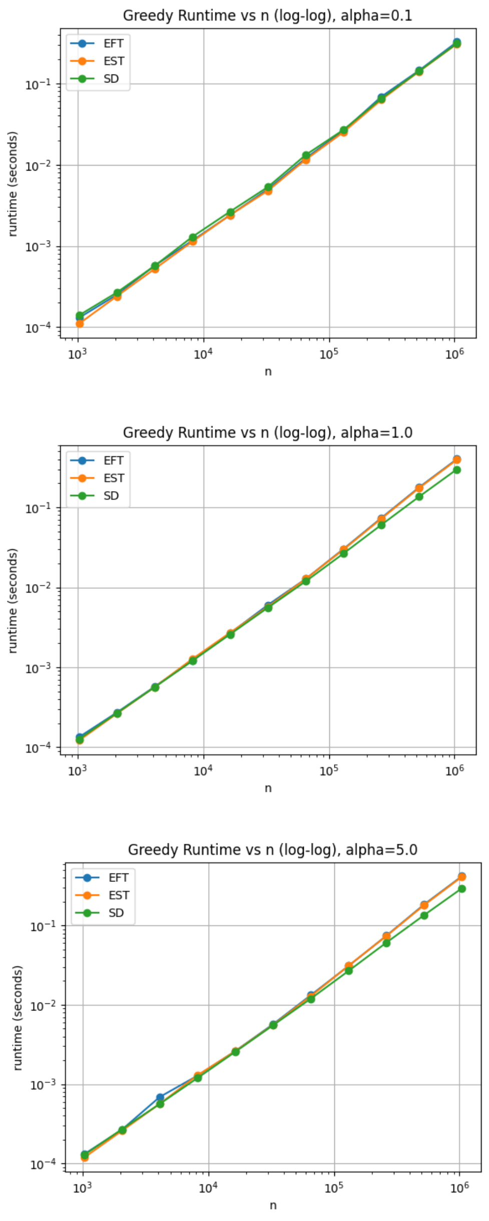

Fig. 1. Greedy runtime growth under different overlap levels.

C. Normalized Greedy Runtime Growth

From the following results, the plots of each algorithm increases as n increases while being in the same scale. SD consistently plots below EST and EFT in medium to low overlaps but is still asymptotically $O(nlogn)$.

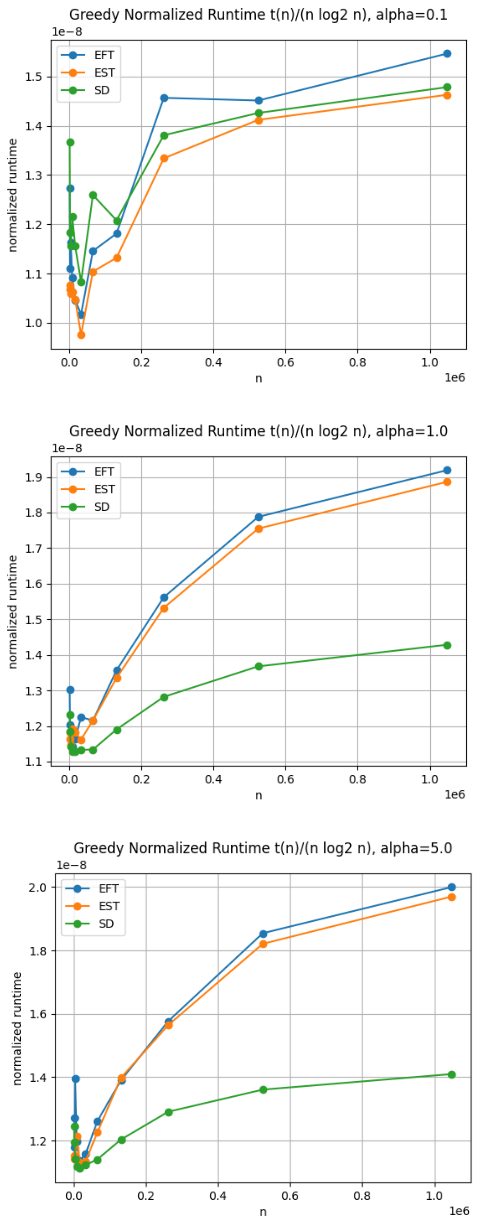

Fig. 2. Normalized greedy runtime growth under different overlap levels.

D. Exhaustive Runtime Growth

The plots for the exhaustive approach explodes rapidly which matches the expected behaviour $O(n2^n)$

E. Normalized Exhaustive Runtime Growth

In the plots for normalized exhaustive runtime, the plot decreases with n, which indicates that the exhaustive algorithm is performing better than the theoretical worst case $O(n2^n)$. This can happen due to a process called pruning. This usually happens during feasibility checking which allows the algorithm to discard many subsets quickly once an overlap is detected so the average amount of work per subset decreases as n increases.

However, since worst-case inputs can exist, like when most intervals are mutually compatible, pruning becomes ineffective

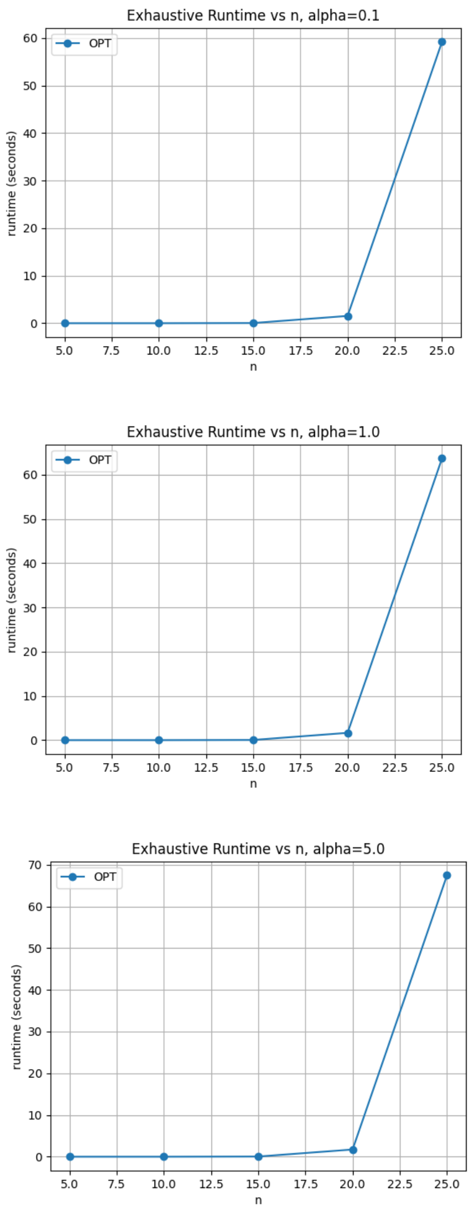

Fig. 3. Exhaustive runtime growth under different overlap levels.

and the algorithm would be forced to examine all 2^n subsets. Hence, while pruning can reduce runtime for small n, it still does not change the worst-case complexity.

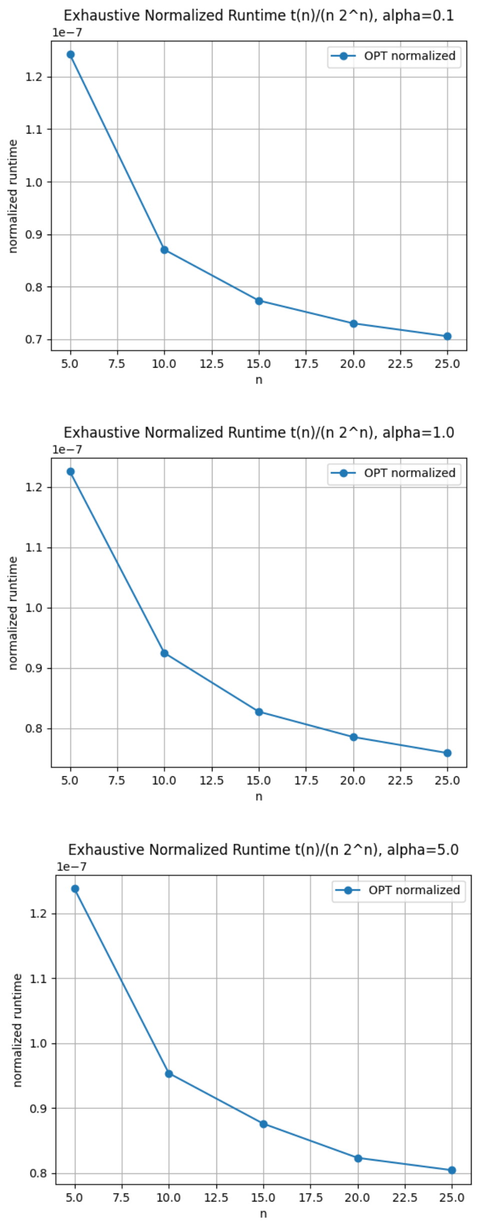

Fig. 4. Normalized exhaustive runtime growth under different overlap levels.