Part 1.

1. What are the three primary criteria we seek when designing an algorithm
   ans: correctness, effectiveness in terms of speed, memory, and optimality, and ease of implementation

2. List the five basic data structure types and their avg time and space complexities
   ans:

	Space	Access	Search	Insert	Delete
Array	O(n)	O(1)	O(n)	*	*
Linked List	O(n)	O(n)	O(n)	*	*
Stack	O(n)	-	-	O(1)	O(1)
Queue	O(n)	-	-	O(1)	O(1)
Tree	O(n)	-	O(logn)	O(logn)	O(logn)

Array
  insert/delete (append/pop) = O(1) amortized
  " at beginning/middle = O(n)
Linked List
  insert/delete at head/known node = O(1)
  " tail = O(1)

3. What is the specific objective of the Interval Scheduling Problem?
   ans: The objective is to find the maximum number of non overlapping events from a given set of intervals

4. List three different types of greedy strategies that could be used for interval scheduling
   ans: Earliest Start Time - events in asc order of start time
   Earliest Finish Time - " finish time
   Shortest Distance/Interval - " interval length

5. What is the fundamental difference between an algorithm and a heuristic?
   ans: algorithms are a set list of steps that results in a guaranteed correct solution whereas, a heuristic is an approach that quickly finds a good solution without a guarantee for its correctness and optimality

Part 2:

1. Describe the EFT greedy algorithm steps

ans: step 1 - sort events by their finish time in asc order
step 2 - choose the first item i.e. the event with the earliest finish time
step 3 - set the finish time of chosen interval as "current finish time"
step 4 - Loop through the remaining sorted events where
if current finish time ≤ Start time
then set new finish time of the new event as "current finish time"
step 5 - Keep doing step 4 until all events are checked
step 6 - Print all selected intervals

2. Why is the nearest neighbour approach for the TSP considered a heuristic rather than a correct algorithm?

ans: Because this particular approach results in a valid tour but not necessarily the optimal/shortest tour

3. Explain the concept of a Loop invariant and its relation to mathematical induction

ans: Invariant means that something is always true. A loop invariant is a condition where something is always true at the beginning and end of every iteration of a loop; Used to prove that an algorithm is correct.

As a form of proof, a loop invariant contains three main steps:
1. Initialization where the invariant is true before the first iteration
2. Loop maintainance " , and still is after that iteration
3. Termination where the invariant + loop stopping condition implies that the algo's result is correct

On the other hand, mathematical induction is a logical structure that is used to prove a statement is true for all all integers. Therefore both loop invariant and mathemical induction are the same in different contexts.

4. What does it mean for a problem like the TSP to be NP-hard?

ans: NP-hard means that no efficient algorithm is known to solve all instances of a problem quickly as the number of points/nodes increases

5. How does modularity contribute to an algorithm's ease of implementation?

ans: modularisation is the practice of building programs using independent modules/components to simplify implementations. In addition, these components, or building blocks can be reused and tested independently thus, simplifying the overall implementation of an algorithm.

Part 3:

1. Proof by Contradiction: Summarize the arguement used to prove that the greedy algorithm for interval scheduling cannot select fewer jobs than an optimal schedule

ans:

1. assumption: greedy algorithm for interval scheduling can select fewer jobs than an optimal schedule
2. by using EFT as our greedy approach,
   Let g_1 be the first job chosen by greedy
   o_1 " " optimal schedule
3. Since greedy will definitely choose earliest finish time,
   $finish(g_1) \leq finish(o_1)$
4. ∴ we can replace o_1 with g_1 without breaking feasability since g_1 finishes no later thus, leaving as much room for remaining jobs
5. As a result, we can construct a new optimal schedule that starts with g_1 where there would still be the same number of jobs as the optimal algorithm
6. By repeating the o_1 to g_1 replacement, we can see that greedy can be converted to an optimal schedule but still would not reduce the number of jobs
7. ∴ the assumption where greedy can select fewer jobs fails

∴ Greedy cannot select fewer jobs than optimal schedule

2. Counterexample Challenge: Draw or describe a set of intervals where the Shortest Distance/Interval greedy strategy fails to provide a maximum size subset of compatible jobs

ans: Given the following intervals

$\{(0,1),(0,5),(1,5),(5,10)\}$

Optimum = $\{(0,1),(1,5),(5,10)\}$

shortest distance = $\{(0,1),(1,5)\}$

3. Exhaustive Search Complexity: Why exhaustive (trying n! permutation) impractical for a Robot Tour Optimization w 20 points? Provide the approximate number of permutations involved

ans: Because exhaustive search would try all n! permutation which would be very slow as the number of points increases

when $n = 20$

$n! = 20!$

$\approx 2.4 \times 10^{18}$

4. Algorithm Correctness: The slides state that "Failure to find a counterexample... does not mean the algorithm is correct." Explain why mathematical induction is preferred over trial-and-error for proving correctness

ans: Mathematical induction is an approach that is used to prove the truthfulness of an algorithm or statement for every possible input. Thus, a reliable way to prove correctness. Trial-and-error, on the other hand, is a way of testing algorithms on a case to case basis meaning that there could exist cases (i.e. inputs) where the algorithm fails.

5. Complexity Tradeoffs: Compare Quicksort and Merge sort based on their worst-case time complexity and space complexity

ans:

	Worst case time complexity	" space
Quicks.	$O(n^2)$	$O(\log n)$
merges.	$O(n \log n)$	$O(n)$

comparison: merge sort results in a more favourable complexity of no more than $O(n \log n)$ in comparison to Quicksort. However, it can potentially use a larger amount of a maximum space of $O(n)$ in comparison to Quicksort.

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.

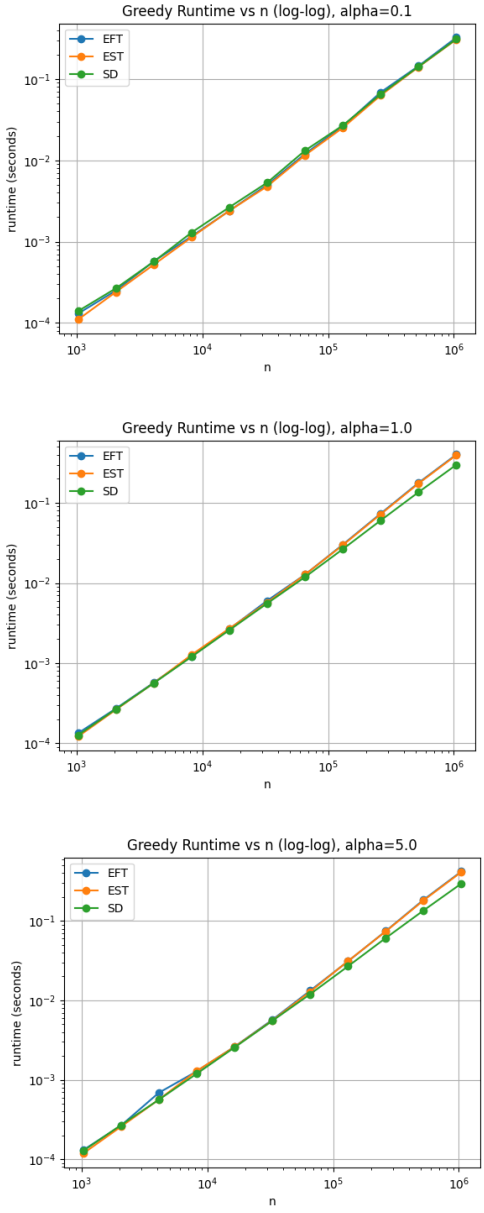

Fig. 1. Greedy runtime growth under different overlap levels.

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 $O(nlogn)$.

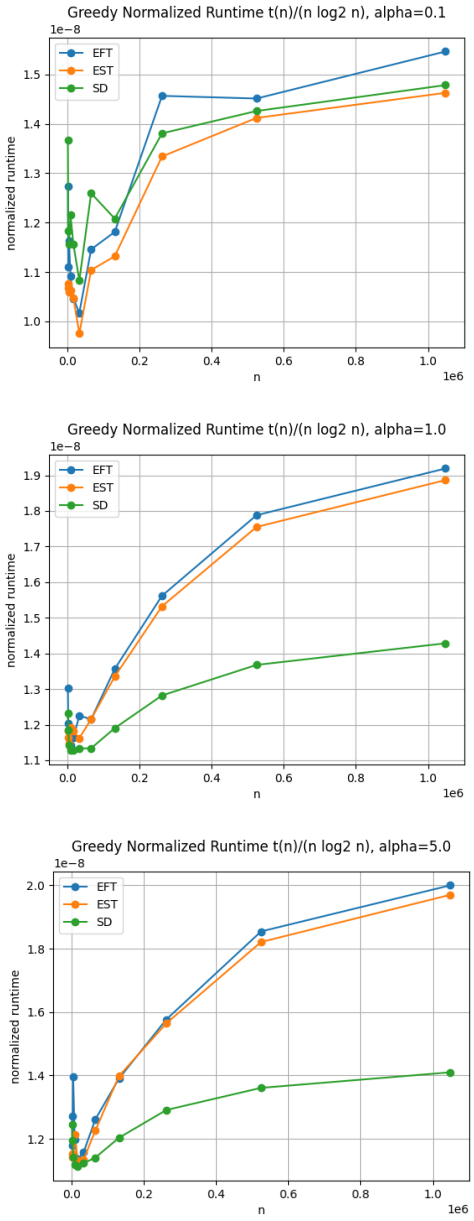

Fig. 2. Normalized greedy runtime growth under different overlap levels.

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

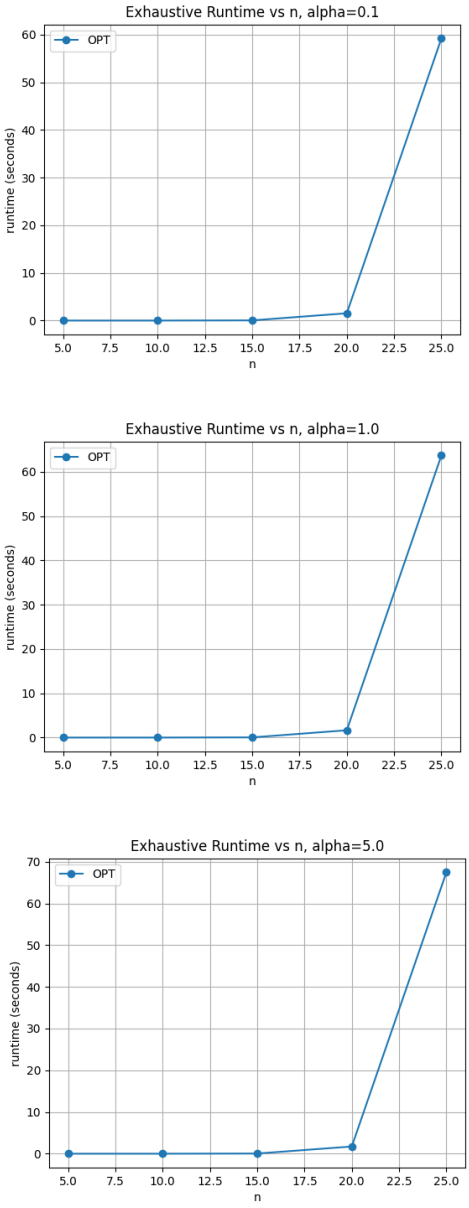

Fig. 3. Exhaustive runtime growth under different overlap levels.

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.

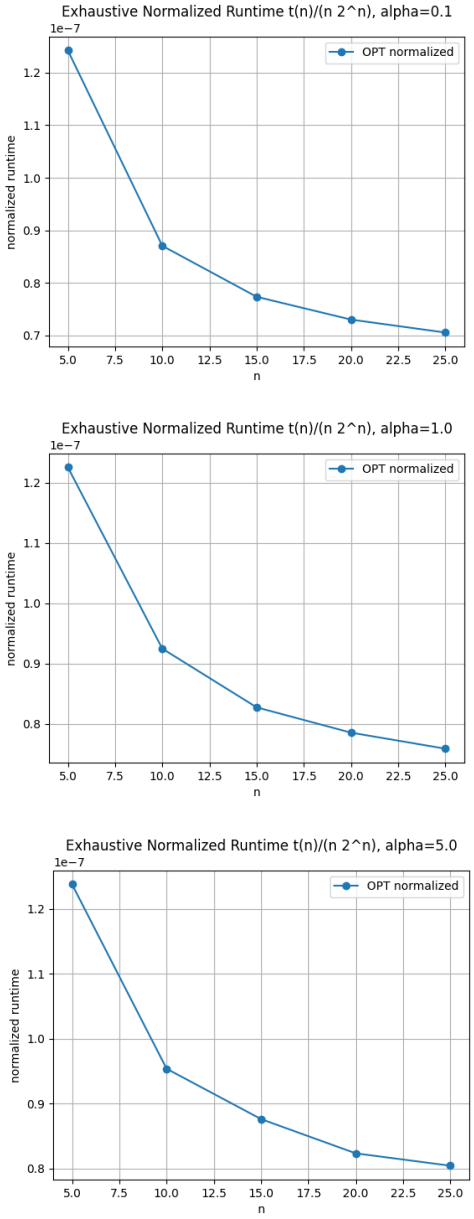

Fig. 4. Normalized exhaustive runtime growth under different overlap levels.