Analyzing loops-10pts Consider the following algorithm, which takes as input two sorted (from smallest to largest) arrays $A$ and $B$ of size $n$ and for each $1 \leq I \leq n$ computes the minimum $1 \leq J \leq n$ for which $B[J] > A[I]$ and stores it in $C[I]$. (If no such $J$ exists, it sets $C[I]$ equal to $n + 1$.)

Give a time analysis of the algorithm, up to $\Theta$.

SmallestLargerElement($A[1..n], B[1..n]$)

1. $J \leftarrow 1$
2. FOR $I=1$ to $n$ do:
3. While $J \leq n$ and $B[J] \leq A[I]$ do $J++$
4. $C[I] \leftarrow J$.

Each time the While loop in line 3 executes, it increments $J$. Since $J$ is never decreased, and the loop terminates when $J > n$, the while loop executes at most $n$ times, and takes constant time. The rest of the FOR loop is constant time, so the FOR loop is also linear time total. Thus, the whole algorithm is $O(n)$. Since the FOR loop also executes exactly $n$ times, the time complexity is also $\Omega(n)$. Thus, it is $\Theta(n)$ time.

Greedy Algorithms and use of data structures in algorithms : 30 points total

Consider the following problem: You are making probes of an asteroid to reveal its chemical composition, by shooting lasers at certain frequencies at the asteroid. Each substance $s_i$ on a list of $n$ possible constituent substances has a range $[l_i, h_i]$ where it will react to the laser. You need to pick the smallest set of frequencies $f_1, \ldots, f_k$ so that each $s_i$ will react to one of the frequencies $f_j$, i.e., for each $i$ there is at least one $j$ with $l_i \leq f_j \leq h_i$.

Part 1 : 5 points for counter-example Below is a greedy strategy for this problem that is not guaranteed to produce optimal solutions. Show that it fails to produce the optimal solution on the following input: $[0,6], [1,6], [2,6], [3,5], [6,11], [6,10], [8,9]$.

Candidate Strategy one: Pick a frequency $f$ that the maximum number of substances react to. Remove substances that react to $f$ and recurse, until all substances are covered.
The number of substances reacting to the different frequencies are:
f=0: 1, f=1: 2, f=2: 3, f=3: 4, f=5: 4, f=6: 5, f=8: 3. Thus, we would pick f=6 as our first frequency, and need to cover [3,5] and [8,9] with two other frequencies, e.g., f = 3 and f = 8. The final output would be {3, 6, 8}.

However, we could also pick f = 5 and f = 8 and cover all substances. Therefore, the greedy solution picks 3 whereas the minimum solution only requires 2, so this greedy algorithm is not optimal.

**Part 2: 5 pts** Here is a greedy strategy that does produce optimal solutions. Illustrate the algorithm on the counter-example from Part 1.

Candidate Strategy two: Let $S_i$ be the substance with the smallest value of $h_i$. Pick frequency $f = h_i$. Remove substances that react to $f$, and recurse, until all substances are covered.

**Part 2: 10 points** For the optimal strategy, Candidate Strategy 2, prove that it is correct. You can use any valid proof method, but here are two possible outlines as hints:

Approach 1: Achieves the bound: Let $S_1, S_2, \ldots, S_k$ be the substances chosen by the greedy strategy (i.e., $S_1$ has the smallest $h_i$, $S_2$ has the smallest $h_i$ once substances reacting to $h_1$ are removed, etc.) First prove that the corresponding intervals $[l_i, h_i]$ are disjoint. Use this to prove that any solution requires at least $k$ frequencies.

Let $S_1, S_2, \ldots, S_k$ be the substances chosen by the greedy schedule, with corresponding frequencies $f_i = h_i$.

We prove that the corresponding intervals are disjoint by induction on the number of substances total. If there is only one substance, then there is only one such interval, and there is nothing to prove.

Assume that the intervals are disjoint when the strategy is used on any set of $n' < n$ substances. Since we remove all substances covered by $f_1$ when proceeding recursively, no substance $S_2, \ldots, S_k$ can be covered by $f_1 = h_1$. Thus, either $h_j < f_1$ or $l_j > f_1$ for $j = 2, \ldots, k$. But since $f_1 = h_1$ is the smallest value of $h_i$ (from the description of the greedy strategy), the first option isn’t possible. Therefore, $l_j > f_1 = h_1$ for each $S_2, \ldots, S_k$, so the intervals for $S_2, \ldots, S_k$ are disjoint from $[l_1, h_1]$ for each $j > 2$. The intervals for $S_2, \ldots, S_k$ were produced by the same strategy for the smaller collection of substances once we remove those covered by $f_1$. So the intervals $S_2, \ldots, S_k$ are pairwise disjoint, and disjoint from $S_1$. Thus, by strong induction on the number of substances $n$, the intervals produced by the greedy strategy are disjoint.

On the other hand, if there are $k$ disjoint intervals, any frequency can cover at most one of these $k$ intervals. Thus, any solution needs at
least $k$ frequencies, the number that the greedy solution uses. Thus, the greedy solution uses the minimal possible number of frequencies.

**Part 4: 10 points** For the optimal strategy (Strategy 2), describe an efficient algorithm that carries out the strategy. Your description should specify which data structures you use, and any pre-processing steps. Give a time analysis.

Now, we can implement the algorithm faster without recursion. Namely, if we order the technologies according to $l$, then any technology with $l < L$ is either in $Span$ or has $h \leq L$ and so will never be in $Span$, since $L$ only increases in recursive calls. So if we trace forward in order of $l$, until we reach an $l > L$, keeping track of the largest $h$ encountered, this largest $h$ will either be from $T_y$ or if $h \leq L$ indicates that Span is empty. So if $h \leq L$, we return “Impossible”, otherwise, we delete the earlier part of the list, add the corresponding $T$ to our list of bought technologies, and continue. This will be total time $O(n \log n)$ to sort the list, and then $O(n)$ time because we delete as we scan through the list. So the total time is $O(n \log n)$. 