Warm-up

Problem 1. Come up with an instance showing that SELECTION-SORT takes $\Omega(n^2)$ time in the worst case.

Problem 2. Come up with an instance showing that INSERTION-SORT takes $\Omega(n^2)$ time in the worst case.

Problem solving

Problem 3. Come up with an instance showing that HEAP-SORT takes $\Omega(n \log n)$ time in the worst case.

Problem 4. Given an array A with n integers, an inversion is a pair of indices $i < j$ such that $A[i] > A[j]$. Show that the in-place version of INSERTION-SORT runs in $O(n + I)$ time where I is the total number of inversions.

Problem 5. Given an array A with n distinct integers, design an $O(n \log k)$ time algorithm for finding the kth value in sorted order.

Problem 6. Given k sorted lists of length m, design an algorithm that merges the list into a single sorted lists in $O(mk \log k)$ time.

Problem 7. In this question we're going to analyse a response generated by an AI, in this case ChatGPT. It was asked to design an algorithm for the following problem that runs in linear time, argue its correctness, and analyse its running time.

Problem Description: We have $n \geq 1$ lighthouses of distinct positive integer height on a line and these lighthouses need to communicate with each other. We are given the heights of these lighthouses in an array A, where $A[i]$ stores the height of lighthouse i. Unfortunately, the area these lighthouses are in is so remote that it doesn't have phone lines and cell phone coverage is spotty at best. So the lighthouse keepers have found a different way to talk to each other: paper airplanes.

Assume all lighthouse keepers are (former) olympic champions in paper airplane throwing, meaning that they can reach any lighthouse as long as there's no lighthouse higher than the thrower's lighthouse between them (the lighthouses are equipped with paper airplane catchers, so hitting the wall of the lighthouse counts as reaching it). Of course, because of the wind coming from the ocean, lighthouse keepers can only send their paper planes to the left (towards the land). More formally, lighthouse k can send a paper airplane to lighthouse i ($i < k$), if there is no lighthouse j such that $i < j < k$ and $A[k] < A[j]$. Note that lighthouse i itself can be higher than lighthouse k because of the paper airplane catchers. We are interested in determining the leftmost lighthouse that each of the lighthouse keepers can reach. To distinguish between a paper airplane stopping at lighthouse 0 and passing over all lighthouses to the left of its starting point, we record the latter as -1 (imagining an infinitely high lighthouse preceding all others where the airplane stops).

Example:
A : $[5,3,1,6,4]$: return $[-1,0,1,-1,3]$. A paper airplane thrown by the lighthouse keeper of lighthouse 0 passes over all lighthouses to its left and is thus recorded as -1. Lighthouse 1 can send their airplanes to lighthouse 0, but not beyond. Lighthouse 2 can communicate with lighthouse 1, but not with lighthouse 0, since lighthouse 1 stops any airplanes throw from lighthouse 2. Lighthouse 3 is higher than any of the previous lighthouses and thus its airplanes pass over them. Finally, lighthouse 4 can send airplanes to lighthouse 3, but no further.

Your task is to design an algorithm that computes for each lighthouse the leftmost lighthouse that its keeper can reach. For full marks, your algorithm should run in $O(n)$ time.

AI Output:
Algorithm Description:

1. Initialization: Initialize an array reach of size n to store the results, where reach$[i]$ will eventually hold the index of the leftmost lighthouse that lighthouse i can reach. Initialize a list candidates to store the indices of lighthouses that have been processed and can potentially be reached by future lighthouses.

2. Processing Each Lighthouse:
   - Iterate through each lighthouse i from left to right.
   - For each lighthouse i, use binary search (specifically, 'bisect_right') on the candidates list to find the smallest index j such that $A[j] \geq A[i]$.
   - If such j exists and j is less than i, update reach$[i]$ to j.
   - If j does not exist or $j \geq i$, set reach$[i] = -1$.

3. Maintaining Candidates: After updating reach$[i]$, insert i into the candidates list using binary insertion (inserting into a sorted list using 'bisect.insort').

4. Output: After processing all lighthouses, reach will contain the indices of the leftmost reachable lighthouse for each lighthouse.

Correctness Proof:

- Binary Search and Insertion: Using 'bisect_right' ensures that j is found efficiently in $O(\log n)$ time for each lighthouse i. Inserting into candidates with 'bisect.insort' also maintains the sorted order in $O(\log n)$ time per insertion.

- Updating reach: For each lighthouse i, reach$[i]$ is correctly updated to the smallest index in candidates where the height $A[j] \geq A[i]$, ensuring it's the leftmost reachable lighthouse.

Time Complexity Analysis:

- Traversal: The main loop iterates through n lighthouses.

- Binary Search and Insertion: Each operation (search and insertion) within the candidates list is $O(\log n)$.

- Therefore, the overall time complexity of the algorithm is $O(n \log n)$.

This algorithm efficiently computes the leftmost reachable lighthouse for each lighthouse in $O(n \log n)$ time, meeting the problem's requirements.

Your task: To analyse the above response consider each of the following questions and briefly explain your answer to each.

a) Does the AI response answer the question, i.e., does the algorithm solve the stated problem and does it run in the required time?

b) Is the algorithm described in sufficient detail, i.e., is all information present or are any parts unclear or missing?

c) Is the correctness argument complete or are there leaps or errors in the logic?

d) Is the running time analysis correct for the provided algorithm?