SAAI MADHUMITHAA
240200097

1. Next Greater Element :
The task is to find for each element, the nearest greater value to its right. A stack is used to efficiently track elements whose next greater value is not yet found. We iterate through the array from left to right. For each element, we compare it with the index at the top of the stack. If the current element is greater; it beomes the next greater element for the index on top. We update the result, it becomes the next greater element for the index on top. We update the result, pop from the stack and repeat until the condition breaks. Finally, indices still in the Stack have no greater element to the right, so they remain -1.

2. A string of brakets is valid if every bracket has a correctly ordered closing bracket. A stack tracks all opening brackets as we scan the string from left to right. when an opening bracket appears, we push it onto the stack. When an closing bracket appears we ensure the stack isn't empty. we check whether the closing bracket matches the top of the stack. If it matches, we pop; otherwise, the string invalid. After processing the entire string, the stack must be empty for the string to be valid.

3. Sliding window
The goal is to find the maximum in every continous window of size k. A deque is used to efficiently store indices of elements in decreasing order of value. For each element, we first pop elements from the front that fall outside the current window. Then, we remove all elements smaller than the current element from the back because they can't be future maximums

Push the current index into the deque. When we reach at least k-1, the front of the deque holds the index into the dequeue of the current window's maximum. Collect these maxima for each valid window.

4. Longest Increasing Subsequence:

We maintain a vector LIS that stroes the smallest possible tail for increasing subsequences of each length. For every character, we find its position in LIS using std::lower bound. If the character is largr than all elements, we append it, extending the LIS. Otherwise, we replace the first element ≥ subsequences optimal. This ensures lis remains sorted and supports binary search. The final size of LIS gives the length of the longest increasing subsequence in O(nlog n).

5. Dijkstra's Algorithm:

It finds the shortest path from source to all nodes in a weighted graph with non-negative edge weights. We store the graph using an adjacency list where each entry contains neighor and weight. A min-priority queue stores pairs of (distance, node) to always process the closet unvisited node next. Initially, all distances are set to infinity except the source, which is 0. For each extracted node, we relax all its outgoing edges. If a shorter path is found, we update the distance and push it into the priority queue. When the priority queue is empty, the dist array contains the shortest distances from the source.