Assignment-9

Ans-1) **Longest Increasing Subsequence**

To solve the longest increasing subsequence we use a 1D-dp where each index of dp indicates the Longest Increasing Subsequence that precedes the vector till that index if we take the given index as the final value of the LIS.

Dp[i] = LIS till i^{th} index if we take arr[i] as the ending value.

The final answer for LIS we can get by lis.size() = $\max_i$ (dp[i]), 0 <= i <= n-1

To find the vector of LIS, instead of just storing the current max value of LIS at that point we find the store a pair where the first index of the pair points to the usual first index and the second element of the pair points to the previous location which stores the max value before that.
Thus, we can construct the LIS by going to the previous elements and ultimately ending at the point where dp[i].F = 1 implying that this is the only element of the lis or dp[i].S = -1, similarly implying that there are no elements preceding the given one in the LIS.

Overlapping Substructures:-
They occurs when there exists element in the form of:
arr[i] < arr[j], and i<j.
Here the value of arr[i] will be computed multiple for each such j that exist after the given index i.

In the brute force recursion code:-

```
int LIS(j):
    mx=1;
    for(i<j){
        if(a[i]<a[j]) mx=max(mx,LIS(i)); // LIS(i) is computed multiple times.
    }
    return 1+mx;
```

Thus we store a dp for each index such that if one of them is already computer then we don't have to compute it again.

Optimal Substructure:
For a given index the optimal substructure will consist of the index preceding that index and having the max value of LIS and so on till the value of dp[i] equals i.

Time Complexity:- O(n^2)
Time Complexity Analysis:-
Here we can clearly see that there are two nested for loops contributing to the n^2 complexity of the code and the final LIS can be reconstructed from the dp in O(n).
The outer loop is for iterating through index one by one and the inner loop iterates for all indices before that index and finding the value less than value of given index and finding the max value of dp/LIS among them.

Space Complexity:- O(n)
Space Complexity Analysis:-
Since we only maintain a 1D-dp and the final answer of lis is also at most consisting of n element, it's space complexity is O(n).

Illustration for given input:
n = 9
arr[] = [10 20 0 1 5 50 80 34 1000]

Iteration 1:
Initialised with Dp[0]={1,-1}
No inner loop

Iteration 2:
Initialised with Dp[1]={1,-1}
We go to previous element 0 and find the arr[0]<arr[1] and thus it is added to new value of dp[1]={2,0}

Iteration 3:
dp[2]={1,-1}

Iteration 4:
Here only value less than 1 is 0(having index 2) thus
Dp[3]={2,2}

.
.
.

Iteration 7:
dp[6]={1,-1}
We iterate through all previous elements and find that all elements are less than 80 and among them the max value of dp[i] is is 50 (index 5).
Thus dp[6]={5,5}

.
.

Finally, the dp array will be:
Dp[] = [{1,-1} {2,0} {1,-1} {2,2} {3,3} {4,4} {5,5} {4,4} {6,6}]

As mentioned previously we can construct lis from the given dp array starting from the max value of dp i.e. 6 (at index 7) going to index=dp[7].S
Until index = -1 and then we break it.
After this, we reverse the resulting vector since elements were appended in reverse order.
Hence, resulting in the final
Lis = [0 1 10 50 80 1000]