Problem Set 8

Assigned: July 15
Due: July 22
Name: Subhankari Mishra

Problem 1

A.
Ans:

Construct 4 tables:
T1 [I] -> length of the longest increasing subsequence in A [1...I] which includes A [I].
P1 [I] = J, if this subsequence has A [J] as the second to last.
T2 [I] -> length of the longest decreasing subsequence in A [I...N] which includes A [I]
P2 [I] = J, if this subsequence has A [J] as the next element to A [I]

```
SubSeq(S) {
        LIS (S);
        LDS (S);
        Max = T1 [1] + T2 [1] – 1;
        MaxPos = 1;
        For (I = 2 to N) {
                If (Max < (T1 [I] + T2 [I] – 1)) {
                        Max = T1 [I] + T2 [I] – 1;
                        MaxPos = I;
                }
        }
        Return Max, MaxPos;
}
LIS(S) {
        T1 [1] = 1;
        P1 [1] = Null;
        For (I = 2 to N) {
                T1 [I] = 1;
                P1 [I] = Null;
                For (J = 1 to I – 1) {
                        If ( ( A[J] < A[I] ) And ( T1[J] >= T1[I] ) ){
                                T1 [I] = T [J] + 1;
                                P1 [I] = J;
                        }
                }
        }
}
LDS(S) {
        T2 [N] = 1;
        P2 [N] = Null;
        For (I = N - 1 down to 1) {
                T2 [I] = 1;
                P2 [I] = Null;
                For (J = N down to I + 1) {
                        If ((A [J] < A [I]) and (T2 (J) >= T2 (I))) {
                                T2 (I) = T2 (J) + 1;
                                P2 (I) = J;
                        }
                }
        }
```

}

The subset can be traced by following P1 to the left from MaxPos for the increasing part and then, P2 to the right from the MaxPos for the decreasing part.

B.
Ans:

Because there is no way to trace if a value is already used in the subsequence. As we are not storing anywhere that which subsequence corresponds to the corresponding longest subsequence. Moreover, while computing the longest increasing subsequence we have no knowledge of the details about the items already consumed or to be consumed while computing the longest decreasing subsequence.
After identifying the longest subsequence, if we try to find the subsequence and eliminate the duplicates we might end up with a solution which is not the longest subsequence there could be another subsequence in the given sequence which would be greater than the solution. This can be proved by having multiple duplicates in the given sequence.
For example,

S = {10, 4, 5, 11, 2, 7, 5, 4, 9}

T1 = {1, 1, 2, 3, 1, 3, 2, 2, 4}

P1 = {-, -, 2, 3, -, 3, 2, 5, 6}

T2 = {4, 2, 2, 4, 1, 3, 2, 1, 1}

P2 = {6, 5, 8, 6, -, 7, 8, -, -}

As per the above algorithm the longest increasing decreasing subsequence for S would be of length 6 at MaxPos would be 4.

Therefore by following P1 and P2 for the subsequence we would get {4, 5, 11, 7, 5, 4} upon deleting the duplicates we have {4, 5, 11, 7}, which is not the longest increasing decreasing subsequence in S where as we have {10, 11, 7, 5, 4} which is longer than the solution found without any repetition.

Problem 2

Ans:

Let Val [1...n] be the list of values of the objects.
Let Wt [1...n] be the list of weights of the objects.
Let W be the maximum weight allowed for the knapsack.

```
KNAPSACK (W,Wt[1..n], Val[1..n]){
        For (w = 0 to W) {
                V [0, w] = 0;
        }
        For (v = 0 to n) {
                V [v, 0] = 0;
        }
        For (i = 1 to n) {
                For (j = 1 to W) {
                        If (Wt[i] <= j) {
                                If (V [i-1, j] > = (Val[i] + V [i-1, j – Wt[i]]) {
                                        V [i, j] = V [i-1, j];
                                }
                                Else {
                                        V [i, j] = Val[i] + V [i-1, j - Wt[i]];
                                }
```

```
                }
                 Else {
                        V [i, j] = V [i-1, j];
                }
        }
}
```

The best value would be the value at V [n,W].
The running time of the algorithm is O(nW).
Let T be the subset resulting in the best value, then below would be the algorithm to recover the path.

```
Initialize T = Ø;
Recover (V,Val[1..n],Wt[1..n],W){
        I = n, K = W
        While (I > 0 and K > 0) {
                If (V [I, K] ≠ V [I-1, K] {
                        T = T + V [I];
                        I = I -1;
                        K = K – Wt[I];
                }
                Else {
                        I = I – 1;
                }
        }
        Return T;
}
```

For example let Val = {10,40,30,50}
Wt = {5,4,6,3}
W = 10

i/w	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0	0	0	40	40	40	40	40	50	50
3	0	0	0	0	40	40	40	40	40	50	70
4	0	0	0	50	50	50	50	90	90	90	90

Here T (n,W) = 90.
Upon running the recovery algorithm we get the subset to be {40,50}.
Below is the trace for the recovery algorithm.
As 90 ≠ 70 so V [4] is added to T.
T = {50}
Now, K = W – Wt[4] = 7
I = 3
V [I, K] = V [3, 7] = 40 which is equal to V [2, 7] i.e. 40
I = 2
Now, V [2, 7] ≠ V [1, 7]; i.e. 40 ≠ 10
So, V [2] is added to T.
T = {50, 40}
K = 3
I = 1

V [I, K] = 0 which equal to V [I-1, K] i.e. 0

` I = 0 <-- the loop exits and the program return T which is (50, 40}

Problem 3

Ans:

Let S [i,j] be the sum of elements i through j and B[i,j] be the overall imbalance of the best subtree spanning elements i through j.

Let Nums be the array containing the value for the leaf nodes.

Let difference be the function returning the arithmetic difference between the values at the nodes passed to the function.

Let N = |Nums|

```
Best_tree{
    For (I = 1 to N){
        S[I,I] = Nums[I];
    }
    For (length = 2 to N) {
        For (I = 1 to (N + 1 – length)) {
            Best = ∞;
            J = I + length – 1;
            For (M = I to J – 1) {
                New = difference (V [I, M], V [M + 1, J]);
                If (New < Best) {
                    Best = New;
                    S [I, J] = S [I, M] + S [M+1, J];
                    B [I, J] = New;
                }
            }
        }
        Return S,B;
    }
```