CS 660 HOMEWORK - II

Q1. A[1...m] B[1...n]

Let the array A be "AWESOME" and the array B be "ASSUME".

∴ Forming the dynamic prog. matrix

		A	W	E	S	O	M	E
	0	0	0	0	0	0	0	0
A	0	(1)	1	1	1	1	1	1
S	0	1	1	1	(2)	2	2	2
S	0	1	1	1	2	2	2	2
U	0	1	1	1	2	2	2	2
M	0	1	1	1	2	2	(3)	3
E	0	1	1	2	2	2	3	(4)

LCS: ASME

Base cases:

→ After forming the above similarity table, where we increase the count if there is a new similar element between 0 to i and 0 to j.

→ we analyze the table and try to deduce the base cases.

① If the element above the LCS[i,j] is equal to the element preceeding LCS[i, j] i.e.
LCS[i-1, j] == LCS[i, j-1],
then LCS[i,j] = 1 + LCS[i-1, j-1]

② If the element above of and preceeding to LCS[i,j] are not equal
i.e. LCS[i-1, j] != LCS[i, j-1]

then LCS [i,j] = max[LCS(i-1,j), LCS(i,j-1)]

Constructing the table:

```
LCS(i,j) { for i = 0 to m
             LCS [i,0] = 0
           for j = 0 to n.
             LCS[0,j] = 0.
           for i = 0 to m
             for j = 0 to n.
               if(A[i] == B[j] && (i=0, j=0))
                 LCS[i,j] = 1.
               else if A[i] == B[j]
                 LCS[i,j] = LCS[i-1, j-1] + 1
               else
                 LCS[i,j] = max[LCS(i-1,j), LCS(i,j-1)]
           }.
```

Getting the length / LCS:

Let K be new array for LCS. and counter c=0

~~for~~ during iterating from back to first.

i.e. from bottom right of matrix,

```
i=m, j=n;
while (i>0 && j>0) {
    if(LCS[i-1,j] == LCS[i,j-1]) <
                        LCS[i,j])
        K[c] = LCS[i,j]; c++;
        i--; j--;
    if[ LCS[i-1,j] == LCS[i,j-1] &&
            LCS[i,j-1] == LCS[i,j]]
        i--; j--;
    if ( LCS[i-1,j] > LCS[i,j-1])
            i--;
    else j--;
}
```

Here, we printed the resultant matrix backward into K array.
we can get the correct values by iterating from the end in the array K.

Also, in the given (written) above code, we iterate the matrix from back to top and not for any change or decrease in values for LCS[i,j].

The running time for the problem would be m×n since we need to construct the LCS table.

$\therefore O(mn)$

e.g. AWESOME & ASSUME

LCS: ASME.

Q2. Algorithm for string of parentheses & brackets is balanced or not.

Let a string be. w = [( ) ( )] ( ).
Let us assume that k is such an element in w such that w[1 ... k] and w[k+1 ... n] are also balanced parenthesis.
provided n is the length of array w.

Moreover, let i be the starting index of balanced parentheses array and j be the ending index.

We need to make sure that i < j.

∴ Base Cases:
We can deduce the following cases based on the above details.

① If i ≥ j, there is no balanced subseq i.e. 0.

② We need to compute the maximum value between balanced seq (i, k) + balanced seq (k, j-1)
This is done when a[i] == '(' and a[j] == ')'.

→ Upon calculating the maximum of all the elements, we add 2 since we subtracted/removed the inital '(' at i and ')' at j.

(3) We can check for all (i,k) (k+1, n) for sub seq when i = '(' and j = ')' but it needs to have a minimum value of 2

Matrix :

	[	(	)	]	(	)
[	0					
(	0	0				
)	2	2	0			
]	4	2	0	0		
(	4	2	0	0	0	
)	6	4	2	2	2	0

starting at.

ending at.

If the final value in the bottom left corner is equal to the length of the input string then the input string is a balanced parantheses.

Pseudo code.

```
isBalSeq (A[1...n]):  //Initialize matrix
    for (i = n ; i>0 ; i++) {
        Mat[i,i] = 0
        for(j = i+1 ; j<=n ; j++)
            if((A[i] == '(' && A[j]==')')
                || A[i] == '[' && A[j]==']'))
                Mat[i,j] = Mat[ i+1, j-1 ] +2
            else Mat[i,j] = 0
```

```
for k = i to j-1. {                    // the inner max condition.
  Mat[i,j] =
      max(mat[i,j],
          mat[i,k] + Mat[k+1, j]
}
}
if (Mat[1,n] == n.)
    input is "Balanced string"
else "not".
```

The running time would be $O(n^3)$ since there are 3 loops

Q3. Sequence of cards game.

Let the sequence of cards be from S1 ... Sn. with the values Vi.

Let us first form the 2D array / Matrix for the problem.

	2	1	5	8
2	2,0	2,1	6,2	10,6
1		1,0	5,1	9,5
5			5,0	8,5
8				8,0

Base Cases:

① The diagonal for the matrix is supposed to give Alice the maximum value V[i] ∴ S[i] == S[j] and there's only one card.

(2) Calculate the TotMax
TotMax = sum of array s from si to sj.

(3) There are 2 possible moves for Alice choose 1st card or the last.
let the 1st card be i and the last card be j
∴ if Alice chooses 1st, she'll gain V[i]
and if last, V[j].
simultaneously the Bob would choose the next card and the same process we'll recurse for him.
i.e subtracting the recursed value from Alice's turn.
∴ V[i] − fn(i+1, j)
or V[j] − fn(i, j−1)

→ For the function, the player would choose the max b/w first & last card
i.e:
fn(i, j) = max {Vi − fn(i+1, j), Vj − fn(i, j−1)}

∴ Pseudo Code:
```
getMax (int i, j) {
for (i=1; i<=n; i++) {
  for (j=i; j<=n; j++) {
    if (i==j)
      ∴ getMax(i,i] == V[i]
else
getMax[i,j] = Max [V[i] − getMax(i+1,j)
```

v[j] - getMax(i, j-1)]

}}

return getMax.

}

Time : $O(n^2)$
: 2 loops

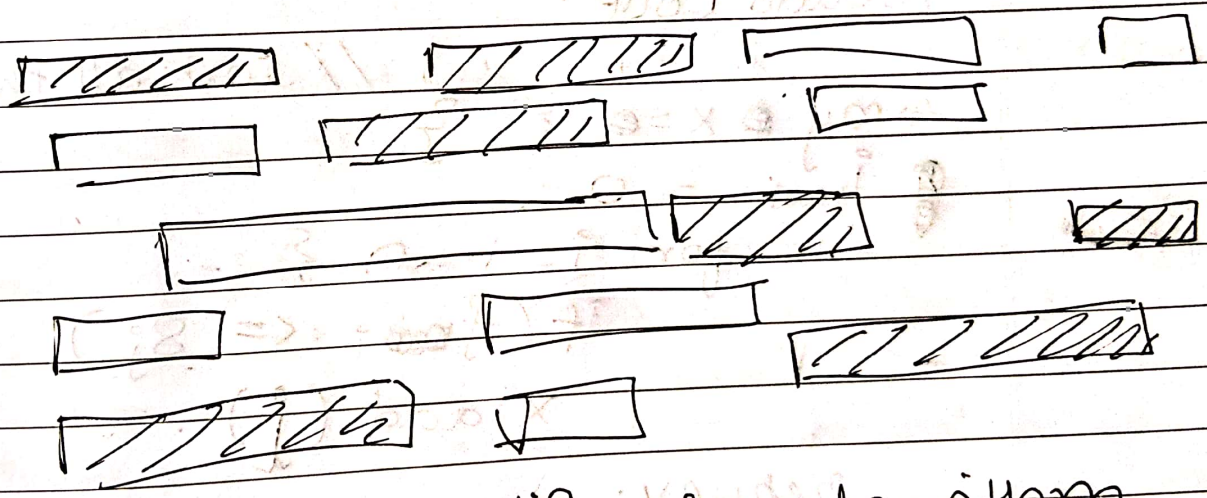
(#) If we want to get Max value of Bob, subtract getMax(i, j) from TotMax(i, j).

get value by getmax(i, j)

e.g. for 1, 8

Alice = 9

Q4. Smallest set of points that ~~stabs~~ cover X.



~~let~~ ~~as~~ To simplify the algorithm,

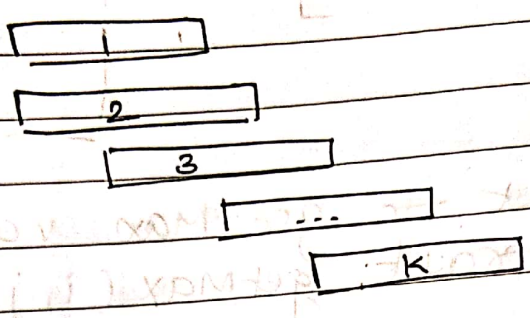
(1) let us first sort all the intervals on the basis of their ~~start pos~~ finish time i.e. in increasing order.

~~R[i] > R[i]~~

R[i] > R[i-1]

R[i] > R[i-1]

∴ The new seq be:



② for each i to n jobs, determine max schedule jobs / time from 0 to i including job i and excluding job i

~~and taking the~~

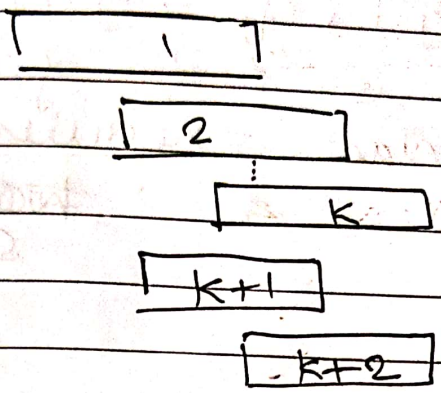
Array x = 0 // Selected optimal jobs

Pseudo Code:

```
                                    // sort all the intervals
Array x = { }
i_last = 0
        for i = 1 : n {
            if (f_i <= S_j)
                x.add (i)
        }
return x;
```

Greedy Algorithm is an algo where the shortest interval is used first for the next or each step.

Let us say that 1, 2, 3 ... k is not the optimal soln in below intervals.



If we pick the next scheduling class, k+1, we will need k+1 and k+2 to cover the same interval covered by k.

But this increases the no. of interval classes compared to getting only k.

∴ This is false and k is the optimal soln.

Proof for Greedy

Let $G(g_1, g_2 \cdots g_l)$ be the classes.

Suppose G is not optimal for all j

$g_1, g_2 \cdots g_{j-1}, g_j$

where $g_1 g_2 \cdots g_{j-1}$ is prefix to an optimal schedule.

but $g_1 = o_1$, $g_2 = o_2 \cdots$; $g_{j-1} = o_{j-1}$ but $g_j \neq o_{j-1}$

∴ NOT Optimal Soln.

After O_{j-1}, the algo will then pick g_j instead of O_j

$\therefore O_1, O_2 \ldots O_{j-1}, g_j \ldots O_{j+1}, O_{j+2} \ldots$

but this is prefix of an optimal class scheduler.

$\therefore$ Contradiction. $\therefore$ Assumed stmt is incorrect.

Qs Maj Element with String.

Here, we can directly state that, the problems can be formed in pairs of 2 and there must be atleast 1 ~~element~~ pair where the elments in it are same if the element is the majority elements.

S = [Ap, Ap, Ba, Ap, Ba, Pe, Ba, Ba]

Here if we divide the array in blocks of 2.

→ (Ap Ap) (Ap, Ba) (Ba, Ap), (Ap Ba), (Ba Pe), (Pe Ba), (Ba Ba)
↓ ↓
Ap Ba.

No common in Ap, Ba
∴ No maj element.

→ If S = [Ap, Ap, Ba, Ap, Ap, Ap, Ba, Ba]

(Ap Ap) (Ap Ba) (Ba Ap) (Ap Ap) (Ap Ap) (Ap Ba) (Ba Ba)
↓ ↓ ↓ ↓ ↓
Ap Ap Ap Ba.

B: [(Ap Ap) (Ap Ap) (Ap Ba)]
↓
B [Ap → Maj element]

Pseudo Code

```
majele (a[])
    if n=2 :
    if (a[1] == a[2])
        return a[1].
    else return "No ele";
    for i = 1 to n {
        if (a[i] == a[i+1])
            [j] = a[i]
        i = i+1 ; j = j+1 ;
    return majele (B[])  ;
}
// Main                    // a = {apple, ba ...}
    m = majele (a[]) ;  counter = 0 ;
    for (i=0 ; i < size(a) ; i++) {
        if (a[i] == m)
            counter ++ ;
    if ( counter > n/2 )
        print (" Maj element is " + m) ;
    else " no maj ele " ;
```

Recursion to simplify each time

entire code runs in O(n)
since each call for
main
and majele run independent
of each other.