**Correctness and Complexity Analysis of Pseudo code**

// In the main function we get text files from user. If there are more than two files, we compare first file with second and their result with third so on.

**Convert to string**

We have a file F and we need to convert it to string. Let A be the resultant array.

//Function to convert text file to string

Convert\_to\_String(File)

1. Aj=File.length
2. Let A[0….j] be a string
3. for i=0 to j
4. A[i]= character of file
5. return A

**Remove Sensitivity**

We have an array A and we need all the characters of A be in lowercase. Let B be the resultant array.

Let A = <SOFIA HASSAN>

Then B = <sofia hassan>

Our **induction hypothesis** is: B(n) = A(n) + 32

**Induction Base**: let n = 1 . Now we will prove our hypothesis on this value.

B(1) = A(1) + 32

**Induction Step:** Now we will prove it for n+1 . As the

B(n+1)=A(n+1) + 32 ------- 1

B(n+1) = B(n)+ n+1 // as it is recursive

= A(n) +32 + (n+1)

= A(n+1) + 32 // which is equal to 1

So this algorithm is correct.

Clear\_Sensitivity(A)

1. for i =0 to A.length
2. if A[i] >= 65 and A[i] <= 92
3. A[i] = A[i] +32
4. return A

**Remove Spaces**

We have an array A and we want to remove all of its spaces. Let C be the resultant array.

Let A = <sofia hassan>

Then C = <sofiahassan>

Our **induction hypothesis** is: C(n) = A(n+1)

**Induction Base**: let n = 1 . Now we will prove our hypothesis on this value.

C(1) = A(1+1)

**Induction Step:** Now we will prove it for n+1 . As the

C(n+1)=A(n+1+1) =A(n+2)------- 1

C(n+1) = C(n)+ n+1 // as it is recursive

= A(n+1) + (n+1)

= A(n+2) // which is equal to 1

So this algorithm is correct.

Remove\_Spaces(A)

1. i = 1
2. size = A.length
3. while i <= A.length
4. if A[i] = “ ”
5. A[i] = A[i+1]
6. size = size- 1
7. return A, size

**Longest Common Subsequence**

Let given sequences are X and Y. Z is a common subsequence to both X and Y. If Z is the longest possible subsequence of both X and Y then it is the longest common subsequence.

X = < B, D, C, A, B, A >

Y = < A, B, C, B, D, A, B >

Common subsequence of X and Y is Z = < B, D, A, B >

**Finding the LCS:**

Could enumerate all subsequences of X (length m) -> 2mand Y (length n) -> 2n must search for hits and sort by length. This is an exponential algorithm. LCS has an optimal substructure property based on prefixes. If X = <x1, ...,xm> then Xi = <x1, ...,xi> is the ith prefix of X and X0 is empty.

**Theorem of Optimal substructure for LCS**

If X = <x1, ...,xm> and if Y = <y1, ...,y­n> are sequences, let Z = <z1, ...,zk> be some LCS of X and Y.

If xm = yn then zk = xm and Zk-1 is an LCS of Xm-1 and Yn-1

If xm ≠ yn then zk ≠ xm => Z is an LCS of Xm-1 and Y

If xm ≠ yn then zk ≠ ym => Z is an LCS of X and Yn-1

**Proof:**

1. If zk ≠ xm then we could add xm = yn to Z to get an LCS of length k + 1. By contradiction it must be that zk = xm = yn . |zk-1 | = k - 1 and is an LCS of Xm-1 and Yn-1 . It is an LCS, if not then for all W CS of Xm-1 and Yn-1 with | W | > k - 1 and so by appending xm = yn we get a CS of X and Y of length greater than k, a contradiction.
2. If zk ≠ xm then z is a CS of Xm-1 and Y. If for all W a CS with | W | > k, then W would be a CS of X and Y, a contradiction.
3. Proof by reversing x and y

This means that to find the LCS of X and Y:

if xm = yn find LCS of Xm-1 and Yn-1

If xm ≠ y

1. find LCS of Xm-1 and Y
2. find LCS of X and Yn-1

and take the larger of 'a.' or 'b.'. Thus we start with small problem, find LCS and grow our solution:

Let c[i,j] = | W |, W is LCS of Xi and Yj

c[i,j] = 0 if i\*j = 0

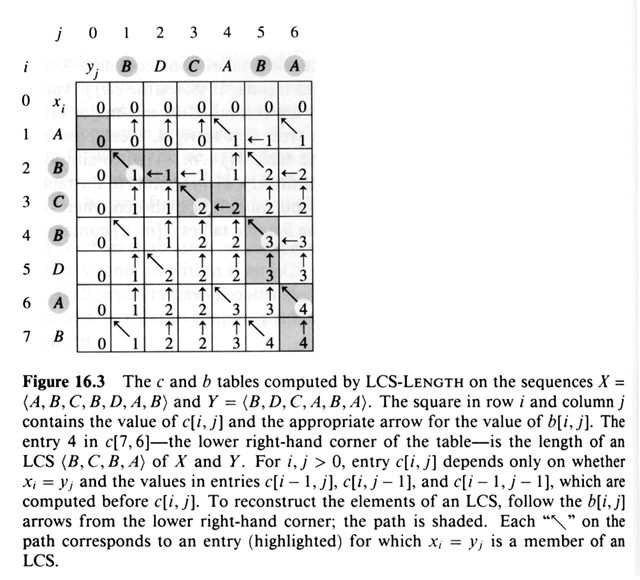
= c[i-1,j-1] + 1, if i\*j > 0 and xi = yj

= max (c[i, j-1]. c[i-1,j]) if i,j > 0 and xi ≠ yj

Could use this to write an exponential algorithm via recursion. However, there are only O (m n) sub-problems, so we use DP. Store c[i,j] and b[i,j] which points to the optimal sub-problem chosen. c[m,n] contains the length of the LCS and b[m,n] directions used to build it.

LCS\_length( X, Y)

1. m = X.length
2. n = Y.length
3. let b[1…m,1…n] and c[0…m,0...n] be new tables
4. for i = 0 to m
5. c[i,0] = 0
6. for j = 0 to n
7. c[0][j] = 0
8. for i = 1 to m
9. for j = 1 to n
10. if x[ i] == y[ j]
11. c[i][j] = c[i - 1][j - 1] + 1
12. b[i][j] = '/'
13. else if c[i - 1][j] >= c[i][j - 1]
14. c[i][j] = c[i - 1][j]
15. b[i][j] = '|'
16. else c[i - 1][j] = c[i][j - 1]
17. c[i][j] = c[i][j - 1]
18. b[i][j] = '-'
19. return c and b



Running Time = O (m n) since each table entry takes O (1) time.

//Getting common characters and their index

Print\_LCS( b, X, i, j)

1. int q=0
2. let D[0…i] be the array to keep index
3. if i == 0 or j == 0
4. return
5. if b[i][j] == '/'
6. Print\_LCS(b, x, i - 1, j - 1)
7. print x[i]
8. D[q] = i
9. q = q+1
10. Q return D
11. else if b[i][j] == '|'
12. Print\_LCS(b, x, i - 1, j)
13. else Print\_LCS(b, x, i, j - 1)

This takes O ( m + n ) since either i or j is decremented in the recursion

// Getting percentage of Plagiarism

**Percentage of Plagiarism**

We have to calculate percentage using i and x variables and p be the result.

Plagiarism\_Percentage (D,A)

// A is given file here

1. i=A.length
2. x=D.length
3. p=x/i \*100
4. print p