**Project Plagiarism checker**

**Time Complexity Analysis**

**Here we have two types of checker:**

1. English script checker
2. C++ code checker

First we will discuss the time complexity of English script checker algorithm.

1. English script checker code:

Here we have some assumptions first:

1. “f1” & “f2” be the two lists such that f1 contain all words of fist file and f2 having second file.
2. “matching” be the list where all the common words of both files will be stored.
3. “matching\_count1” & “matching\_count2” are lists they contains the count value of a particular common word occurs in particular file. such that there is a word which is common in both first and second file that is located at “first index” in “matching” list, then the “fist index” of “matching\_count1” will have the count of that word occurred in first file and “first index” of “matching\_count2” will have the count value of that common word in second file.

**Code for the checker:**

1. for p in f1:
2. count=1
3. if p.lower() in matching:
4. ind=matching.index(p.lower())
5. matching\_count1[ind]=matching\_count1[ind]+1
6. else:
7. for q in f2:
8. if p.lower()==q.lower():
9. if count==1:
10. matching.append(p.lower())
11. matching\_count1.append(1)
12. matching\_count2.append(1)
13. count=2
14. else:
15. ind=matching.index(p.lower())
16. matching\_count2[ind]=matching\_count2[ind]+1

Let’s assume that the size of “f1” is ‘n’ and “f2” is ‘m’. We have three cases:

1. Best Case
2. Average Case
3. Worst Case

**Best Case:**

The best case in our algorithm is that we assume all same words in first file then the situation will be first it has to take that word and compare in second file and then the all words in the first file will be same then it do not need to search again that word in second file. This is the best case that the code will just run ‘n’ times, in this ‘n’ time just ‘one time’ “inner loop” will run and n-1 times will be “constant time operation”.

1. **n-times ( for loop will run ‘n’ times )**
2. c1 (single operation)
3. n-1 times
4. c2 (single operation) **if** statement will run **‘n-1+c1+c2+c3’** times.
5. c3 (single operation)
6. 1-times
7. **m-times (for loop will run ‘m’ times)**

**Best of Best Case Average of Best case Worst of Best Case**

1. 1-times times m times
2. 1-times 1 times 1 times
3. c4(single operation) c4 (single operation) c4 (single operation)
4. c5(single operation) c5 (single operation) c5 (single operation)
5. c6(single operation) c6 (single operation) c6 (single operation)
6. c7(single operation) c7 (single operation) c7 (single operation)
7. 0-times - 1 times m - 1 times
8. 0-times c8 (single operation) c8 (single operation)
9. 0-times c9 (single operation) c9 (single operation)

**We conclude that we got three cases again now making result of all these:**

**Best of Best:** (n-1) + c1 + c2 + c3 + 1 + 1 + c4 + c5 + c6 + c7 **= n+1+ = O (n)**

**Average of Best:** (n-1) + c1 + c2 + c3 + + 1 + c4 + c5 + c6 + c7 + ( - 1) + c8 + c9 = **n+m-2+ = O (n+m)**

**Worst of Best:** (n-1) + c1 + c2 + c3 + m + 1 + c4 + c5 + c6 + c7 + (m-1) + c8 + c9 = **n+2m+1+ = O (n+2m)**

**Average of Best Case Time Complexity: O (n+m)**

**Average Case:**

The best case in our algorithm is that we assume half of words are same in first file then the situation will be that half of the time it will find same words in the file and half time the words will be different. It means the inner loop will work n/2 times. Let’s see how!

1. **n-times ( for loop will run ‘n’ times )**
2. c1 (single operation)
3. times
4. c2 (single operation) **if** statement run **‘ -1+c1+c2+c3’** times.
5. c3 (single operation)
6. times
7. **m-times (for loop will run ‘m’ times)**

**Best of Average Case Average of Average case Worst of Average Case**

1. **\*** 1 times **\***  times **\*** m times
2. **\*** 1 times **\*** 1 times **\*** 1 times
3. c4(single operation) c4 (single operation) c4 (single operation)
4. c5(single operation) c5 (single operation) c5 (single operation)
5. c6(single operation) c6 (single operation) c6 (single operation)
6. c7(single operation) c7 (single operation) c7 (single operation)
7. 0-times **\* (**  – 1) times **\*(** m – 1) times
8. 0-times c8 (single operation) c8 (single operation)
9. 0-times c9 (single operation) c9 (single operation)

**We conclude that we got three cases again now making result of all these:**

**Best of Average:**  - 1+ c1 + c2 + c3 + + + c4 + c5 + c6 + c7 **= 3 () + 1 + = O ()**

**Average of Average:**  - 1+ c1 + c2 + c3 + ( **\***  ) + +c4+c5+c6+ c7 + ( -1)+ c8+ c9

= **n + + - 2 + = O ()**

**Worst of Average:**  - 1 + c1 + c2 + c3 + ( **\*** m) + + c4 + c5 + c6 + c7 + ( \*m - 1)+ c8 + c9

**= n + (n \* m) + 1 + = O (n + (n\*m))**

**Average of Average Case Time Complexity: O ()**

**Worst Case:**

The best case in our algorithm is that we assume all words in first file are different then the situation will be that it always has to go through the inner loop then the first if condition will not be execute. Let’s see how!

1. **n-times ( for loop will run ‘n’ times )**
2. c1 (single operation)
3. 0 times
4. 0 times **if** statement will not run
5. 0 times
6. n times
7. **m-times (for loop will run ‘m’ times)**

**Best of Worst Case Average of Worst case Worst of Worst Case**

1. n **\*** 1 times n \* times n \* m times
2. n \* 1 times n \* 1 times n \* 1 times
3. c2(single operation) c2 (single operation) c2 (single operation)
4. c3(single operation) c3 (single operation) c3 (single operation)
5. c4(single operation) c4 (single operation) c4 (single operation)
6. c5(single operation) c5 (single operation) c5 (single operation)
7. 0-times n \*( – 1) times n \* (m – 1) times
8. 0-times c6 (single operation) c6 (single operation)
9. 0-times c7 (single operation) c7 (single operation)

**We conclude that we got three cases again now making result of all these:**

**Best of worst:** c1 + n + n \* 1 + n \* 1 + c2 + c3 + c4 + c5 **= 3n + 1 + = O (3n)**

**Average of worst:**  c1 + n + (n \* ) + (n \* 1) + c2 + c3 + c4 + c5 + n \*( – 1) + c6 + c7

= **n +** **+ n +**   **- 1 = O (n(m + 2))**

**Worst of worst:** c1 + (n \* m) + (n \* 1) + c2 + c3 + c4 + c5 + (n \* (m-1)) + c6 + c7

= (**n \* m) + n + (n \* m) + n + = O (2(n \* m))**

**Average of Worst Case Time Complexity: O (n(m + 2))**

1. English script checker code:

Here we have some assumptions first:

1. “X” & “Y” be the two variables such that X contains a line of fist file and Y having second file’s.
2. Now we will give one by one two lines to the “checker” function it will check them.

**Code for the C++ Checker:**

def **checker**(X,Y): # **Driver Code**

1. global s
2. s=""
3. if len(X)>len(Y):
4. t=(len(X)\*90)/100
5. else:
6. t=(len(Y)\*90)/100
7. if X==Y:
8. return X
9. else:
10. b,c=**LCS(X,Y) #calling main code**
11. Print\_LCS(b,X,len(Y),len(X))
12. if s != "" and len(s)>t:
13. return s

def **LCS**(X,Y): # **Main Code**

1. m=len(X)
2. n=len(Y)
3. c=[[0 for i in range(m+1)] for j in range(n+1)]
4. b=[[0 for i in range(m+1)] for j in range(n+1)]
5. if X[0].lower()==Y[0].lower():
6. if m==0 or n==0:
7. return
8. for i in range(1,n+1):
9. for j in range(1,m+1):
10. if Y[i-1].upper()==X[j-1].upper():
11. c[i][j]=1+c[i-1][j-1]
12. b[i][j]="D"
13. else:
14. c[i][j]=max(c[i-1][j],c[i][j-1])
15. if c[i-1][j]>c[i][j-1]:
16. b[i][j]="U”
17. else:
18. b[i][j]="L"
19. return b,c

def **Print\_LCS**(b, X, i, j): **# Result generating**

1. if i == 0 or j == 0:
2. return
3. if b[i][j] == "D":
4. global s
5. s=X[j-1]+s
6. Print\_LCS(b, X, i-1, j-1)
7. elif b[i][j] == "U":
8. Print\_LCS(b, X, i-1, j)
9. elif b[i][j] == "L":
10. Print\_LCS(b, X, i, j-1)

As we see that the main code is LCS so we will discuss that in our time complexity:

We have three cases

1. Best Case
2. Average Case
3. Worst Case

In order to discuss every case that will be waste of time because in each there is nothing much difference.

1. c1
2. c2
3. c3
4. c4
5. and 6. The code will be returned so a rare chance.

7 **n times**

8 **m times**

9 to 19 will all give us not much difference margin.

We see that the **n \* m** both loop will always run in every condition except rare case of 0 length of any line. It means that the **n \* m**  times the code will run is confirmed.

**Best case : O (n \* m)**

**Average case : O (n \* m)**

**Worst case : O (n \* m)**