1. **Can File comparison utilities like wc, grep, diff be used for comparing the programs, if yes how and if not what would be the limitation of these utilities?**

WC : wc, or "word count," prints a count of newlines, words, and bytes for each input file.

Using these utilities for detecting plagiarism is not advisable as plagiarised programs that have fewer or more words, bytes and characters can easily be created. Adding comments alone will substantially increase the word count and characters.

diff - find differences between two files

Simple changes like variable names, adding comments etc. can cause increase in amount of differences and even though program is plagiarised, we might wrongly assume that program is different and not plagiarised.

Grep: print lines matching a pattern

Searching for specific patterns in files is a very tedious task and is very inefficient.

To conclude, these utilities can be used for detecting plagiarised programs with very little modifications done to the original program but not for more complex types of plagiarism.

2. **What are the decision parameters to detect Level 0 plagiarism ? Write a algorithm for same**.

Level 0 plagiarism refers to the type of plagiarism where the original code is resubmitted without any modification. Detecting level 0 plagiarism is simple as we only need to compare each line in both the original file and plagiarised file. Linux utilities like grep, diff etc can be useful in such a case.

Algo :

```
Read contents of both files
While all lines of file 1 and file 2 have not been visited
        if lines in file 1 and file 2 are not perfect match
                print “found difference”
                break out of loop
```

if all lines are perfect match

print plagiarised

3. **What are the decision parameters to detect Level 1 plagiarism ? Write a algorithm for same**.

The decision parameters for detecting Level 1 plagiarism is that the that two files should match after ignoring the whitespace and the commets.

Algo :

Read contents of both files

While all lines of file 1 and file 2 have not been visited

While current lines in file 1 & file 2 are not code lines

If line starts with "//" or is empty

Skip line

If line starts with /*

Skip all lines until */ is detected

If line is code line

set iterator condition as fulfilled and end loop

if lines in file 1 and file 2 are not perfect match

print "found difference"

break out of loop

if all lines are perfect match

print plagiarised

4. **What are the decision parameters to detect Level 2 plagiarism ? Write a algorithm for same**.

Same as level 3 plagiarism

5. **What are the decision parameters to detect Level 3 plagiarism ? Write an algorithm for same**.

The decision parameters for detecting Level 2 plagiarism is that the two files should match after ignoring the whitespace and the comments

And then create hash map file for both programs that contains details of variable , variable count for specific data types and operator usage counts within the two programs.

Algo:
Read contents of both files
While all lines of file 1 and file 2 have not been visited
While current lines in file 1 & file 2 are not code lines
If line starts with "//" or is empty
Skip line
If line starts with /*
Skip all lines until */ is detected
If line is code line
Create hash map for both files
(contain identifiers like variable ,methods ,classes ,etc.)

Compare both hash map file
If not perfect match

print "found difference"

break out of loop

if perfect or above 90% match

print plagiarised

6. **What are the decision parameters to detect Level 4 plagiarism ? Write a algorithm for same.**

Pending

7. **What are the decision parameters to detect Level 5 plagiarism ? Write a algorithm for same**.

Pending

8. **Comparison between different Metric - Halstead metric , U.S.Airforce ACCUSE metric, Bowling Green State University Metric, Berghel and Sallach metric, University of Southampton metric, Brunel University metric, Jankowitz at Southampton University metric.**

Program Measures	Halstead metric	U.S. Airforce ACCUSE metric	Bowling Green State University metric	Berghel and Sallach metric	University of southampton metric	Brunel University metric	Jankowitz at Southampton University metric
Number of characters per line	- -	- -	- -	- -	- -	m 1	z 1
Number of comment lines	- -	a 3	- -	f 2	- -	m 2	- -
Number of indented lines	- -	a 2 0	- -	- -	- -	m 3	- -
Number of blank lines	- -	- -	- -	- -	- -	m 4	- -
Average procedure/functional length	- -	- -	- -	- -	- -	m 5	- -
Number of reserved words	- -	- -	- -	f 4	c 3	m 6	y 3 , z 2
Average identifier length	- -	- -	- -	- -	- -	m 7	- -
Average space percentage per line	- -	- -	- -	- -	c 2	m 8	- -
Number of labels and gotos	- -	a 1 5	- -	- -	- -	m 9	- -
Unique Operands	n 2	a17,a6,a9,a10,a11	b 1	f5,f6,f10,f11,f15	c 4	m 1 0	y 2 , z 3
Number of program intervals	- -	- -	- -	- -	- -	m 1 1	- -
Number of colors used in coloring the control graphs	- -	- -	- -	- -	- -	m 1 2	- -
Number of vertices colored with color 3 in coloring the control graphs	- -	- -	- -	- -	- -	m 1 3	- -
Number of vertices colored with color 4 in coloring the control graphs	- -	- -	- -	- -	- -	m 1 4	- -
Total operands	N 2	a 1 9	b 6	f 7 , f 1 3	- -	m 1 5	y 4
Unique operators	n 1	a 1 6 , a 5	- -	f 8 , f 9 , f 1 4	- -	m 1 6	y 4
Total operators	N 1	a 1 8	- -	f 1 2	- -	m 1 7	- -
Program factor structure percentage	- -	- -	- -	- -	- -	m 1 8	- -
Program impurity percentage	- -	a 7	- -	- -	- -	m 1 9	- -
Module contribution percentage	- -	a 8	b 7	- -	- -	m 2 0	y 1 0
Numbers of modules	- -	- -	b 2	- -	c 6	m 2 1	- -
Conditional statement percentage	- -	- -	b 4	- -	- -	m 2 2	y5,y8,z4,z7
Repetitive statement percentage	- -	a12,a13,a14	b 5	f 3	- -	m 2 3	y6,y7,z5,z6
Number of program statements	- -	a 1 , a 2	b 8	f 1	c 5 , c 1	m 2 4	y 1
Reference sequence order	- -	- -	- -	- -	- -	- -	z 9
Multiple statement lines	- -	a 4	- -	- -	- -	- -	- -

9. **Most of these plagiarism metrics are tested on FORTRAN and PASCAL, since we are working on Java find if there are any metric .**

Ans : The plagarism metrics for Java :

1. identifier renaming.

2. Whitespace padding.

3. Comment alteration.

4. Algebric expressions.

5. Structure of loops(for loop, while loop, do while loop, for each loop).

6. Code reordering.

7. Variable position.

10. **Is there any tools available in to find plagiarism in Java code?**

Ans : 1. JPlag : JPlag is a system that finds similarities among multiple sets of source code files. JPlag does not merely compare bytes of text, but is aware of programming language syntax and program structure and hence is robust against many kinds of attempts to disguise similarities between plagiarized files. JPlag currently supports Java, C#, C, C++, Scheme and natural language text. JPlag is typically used to detect and thus discourage the unallowed copying of student exercise programs in programming education.

2. MOSS : Moss (for a Measure Of Software Similarity) is an automatic system for determining the similarity of programs. Since its development in 1994, Moss has been very effective in this role. The main application of Moss has been in detecting plagiarism in programming classes. Moss can currently analyze code written in the following languages:

C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, a8086 assembly, MIPS assembly, HCL2.

3. Anti-Plagiarism : This software is designed to effectively detect and thereby prevent plagiarism. It is a versatile tool to deal with World Wide Web copy-pasting information from the assignment of authorship.

The goal of this program is to help reduce the impact of plagiarism on education and educational institutions. At present, it distributes free software to detect plagiarism. It checks documents in a format *.rtf, *.doc, *.docx, *.pdf. It also checks the source code of C, C++, C#, Java, etc.

4. SIM : SIM is used to detect plagiarism of code written in Java, C, Pascal, Modula-2, Lisp, Miranda. SIM is also used to check similarity between plain text files. SIM converts the source code into strings of token and then compare these strings by using dynamic programming string alignment technique. This technique is also used in DNA string matching. The alignment is very expensive and exhaustive computationally for all applications because for large code repositories SIM is not scalable. The source code of SIM is available publically but it is no more actively supported.