Software Tools and Techniques Lab 5

Pavan Deekshith Doddi
22110190

Introduction, Setup, and Tools

Overview

This lab focuses on analyzing and measuring different types of code coverage while generating unit test cases using automated tools. Students will work with Python programs to evaluate statement, branch, and function coverage. Additionally, they will use automated test generation tools to improve test effectiveness.

Objectives

- Understand and differentiate various types of code coverage.
- Use automated tools to measure coverage on Python programs.
- Write/Generate effective unit tests to maximize coverage.
- Visualize coverage reports and analyze test effectiveness.

Environment Setup

- Install Python 3.10 and set up a virtual environment.
- Clone the keon/algorithms repository and install dependencies.
- Install required testing tools:
    - pytest (test execution)
    - pytest-cov (line/branch coverage analysis)
    - pytest-func-cov (function coverage analysis)
    - coverage (detailed coverage metrics)
    - pynguin (automated test generation)

Methodology and Execution

Cloning into the repository:

```
base ~/B-Tech/Btech_3rd_Year/6th_Sem/STT/lab5 git:(master)±17 (1.115s)
git clone https://github.com/keon/algorithms.git

Cloning into 'algorithms'...
remote: Enumerating objects: 5188, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (19/19), done.
remote: Total 5188 (delta 23), reused 14 (delta 14), pack-reused 5155 (from 2)
Receiving objects: 100% (5188/5188), 1.43 MiB | 9.86 MiB/s, done.
Resolving deltas: 100% (3242/3242), done.
```

Current commit hash in the documentation:

```
(base) pavandeekshith@Pavans-MacBook-Air-79 algorithms %    git rev-parse HEAD
cad4754bc71742c2d6fcbd3b92ae74834d359844
(base) pavandeekshith@Pavans-MacBook-Air-79 algorithms %
```

Initially when i ran **pip install -r test_requirements.txt**

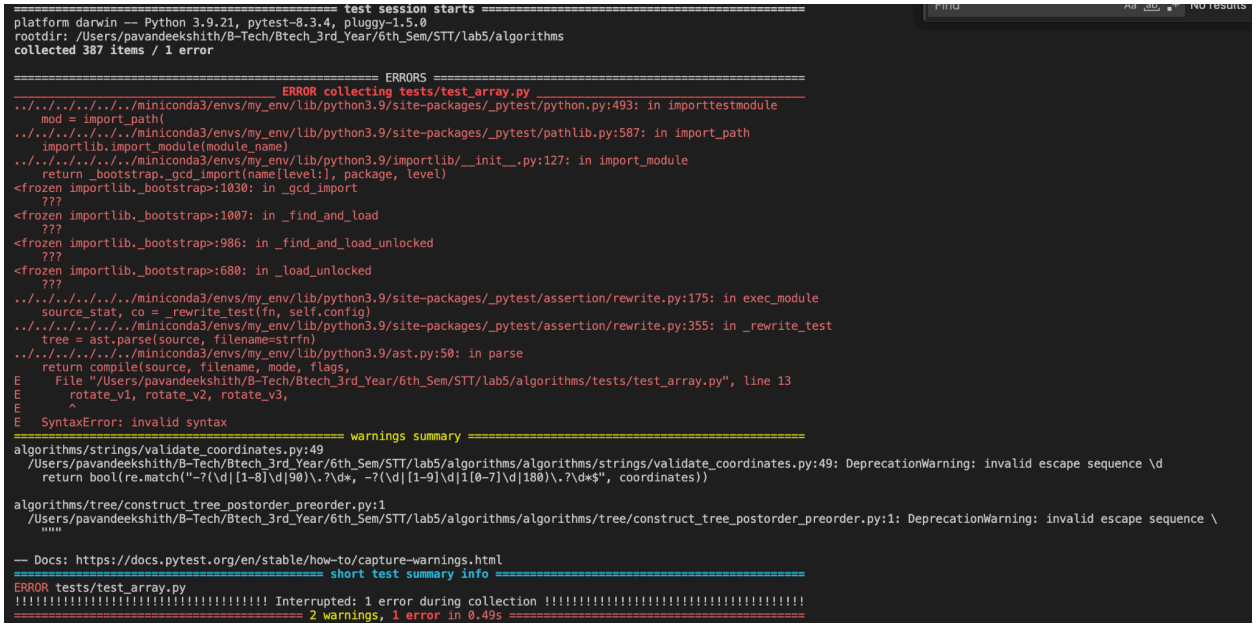
```
platform darwin -- Python 3.9.21, pytest-8.3.4, pluggy-1.5.0
rootdir: /Users/pavandeekshith/B-Tech/Btech_3rd_Year/6th_Sem/STT/lab5/algorithms
collected 0 items / 29 errors

=================================================== ERRORS ====================================================
_____________________________________ ERROR collecting tests/test_array.py ______________________________________
../../../../../../miniconda3/envs/my_env/lib/python3.9/site-packages/_pytest/python.py:493: in importtestmodule
    mod = import_path(
../../../../../../miniconda3/envs/my_env/lib/python3.9/site-packages/_pytest/pathlib.py:587: in import_path
    importlib.import_module(module_name)
../../../../../../miniconda3/envs/my_env/lib/python3.9/importlib/__init__.py:127: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
<frozen importlib._bootstrap>:1030: in _gcd_import
    ???
<frozen importlib._bootstrap>:1007: in _find_and_load
    ???
<frozen importlib._bootstrap>:986: in _find_and_load_unlocked
    ???
<frozen importlib._bootstrap>:680: in _load_unlocked
    ???
../../../../../../miniconda3/envs/my_env/lib/python3.9/site-packages/_pytest/assertion/rewrite.py:175: in exec_module
    source_stat, co = _rewrite_test(fn, self.config)
../../../../../../miniconda3/envs/my_env/lib/python3.9/site-packages/_pytest/assertion/rewrite.py:355: in _rewrite_test
    tree = ast.parse(source, filename=strfn)
../../../../../../miniconda3/envs/my_env/lib/python3.9/ast.py:50: in parse
    return compile(source, filename, mode, flags,
E     File "/Users/pavandeekshith/B-Tech/Btech_3rd_Year/6th_Sem/STT/lab5/algorithms/tests/test_array.py", line 13
E       rotate_v1, rotate_v2, rotate_v3,
E       ^
E   SyntaxError: invalid syntax
_____________________________________ ERROR collecting tests/test_automata.py _____________________________________
ImportError while importing test module '/Users/pavandeekshith/B-Tech/Btech_3rd_Year/6th_Sem/STT/lab5/algorithms/tests/test_aut
Hint: make sure your test modules/packages have valid Python names.
Traceback:
../../../../../../miniconda3/envs/my_env/lib/python3.9/importlib/__init__.py:127: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
tests/test_automata.py:1: in <module>
    from algorithms.automata import DFA
```

We got a lot of errors due to not proper installation of the repository so we had to run the command below

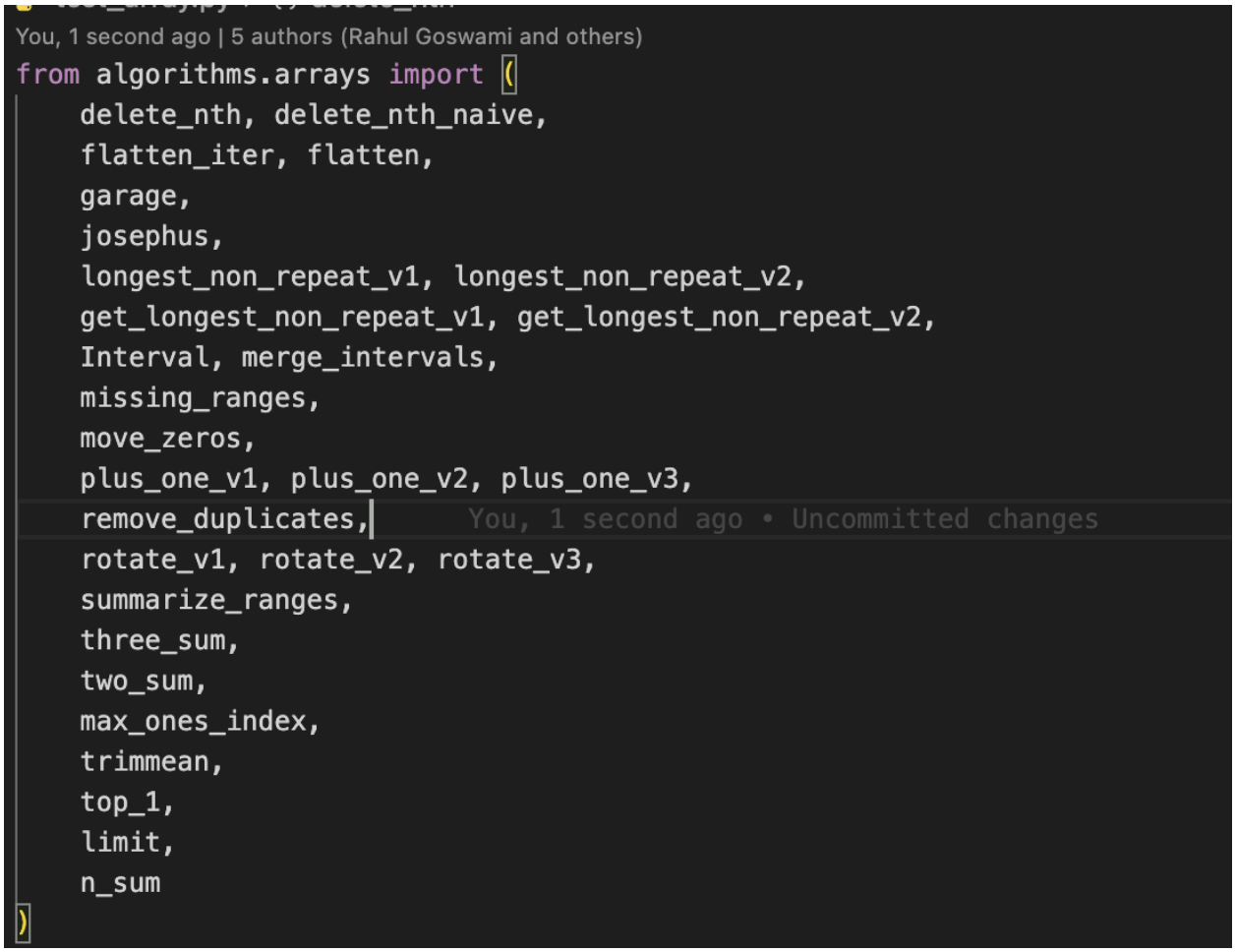
Then ran : *pip install -e*

```
(my_env) pavandeekshith@Pavans-MacBook-Air-9 algorithms % pip install -e .
Obtaining file:///Users/pavandeekshith/B-Tech/Btech_3rd_Year/6th_Sem/STT/lab5/algorithms
  Preparing metadata (setup.py) ... done
Installing collected packages: algorithms
  DEPRECATION: Legacy editable install of algorithms==0.1.4 from file:///Users/pavandeekshith/B-Tech/Btech_3rd_Year/6th_Sem/STT/lab5/algorithms (setup.py develop) is deprecated. pip 25.1 will enforce this behaviour change. A possible replacement is to add a pyproject.toml or enable --use-pep517, and use setuptools >= 64. If the resulting installation is not behaving as expected, try using --config-settings editable_mode=compat. Please consult the setuptools documentation for more information. Discussion can be found at https://github.com/pypa/pip/issues/11457
  Running setup.py develop for algorithms
Successfully installed algorithms
```

Then ran the command — ***pytest***



We got an error in the test_array.py file, since the comma was missing in the line where the cursor is present. I have added the missing comma after remove_duplicates.



When i ran pytest again

```
  def test_remove_duplicates(self):
      self.assertListEqual(remove_duplicates([1,1,1,2,2,2,3,3,4,4,5,6,7,7,7,8,8,9,10,10]))
      self.assertListEqual(remove_duplicates(["hey", "hello", "hello", "car", "house", "house"]))
      self.assertListEqual(remove_duplicates([True, True, False, True, False, None, None]))
      self.assertListEqual(remove_duplicates([1,1,"hello", "hello", True, False, False]))
      self.assertListEqual(remove_duplicates([1, "hello", True, False]))
```

```
=================================================== FAILURES ===================================================
_____________________________ TestRemoveDuplicate.test_remove_duplicates _____________________________

self = <test_array.TestRemoveDuplicate testMethod=test_remove_duplicates>

    def test_remove_duplicates(self):
>       self.assertListEqual(remove_duplicates([1,1,1,2,2,2,3,3,4,4,5,6,7,7,7,8,8,9,10,10]))
E       TypeError: assertListEqual() missing 1 required positional argument: 'list2'

tests/test_array.py:305: TypeError
```

Changed the function to below

```
class TestRemoveDuplicate(unittest.TestCase):

    def test_remove_duplicates(self):
        self.assertListEqual(
            remove_duplicates([1,1,1,2,2,2,3,3,4,4,5,6,7,7,7,8,8,9,10,10]),
            [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  # Expected output
        )
```

Then this error from test_remove_duplicates got resolved and test_summarize is remaining

```
=========================================================== FAILURES ===========================================================
_____________________________________________ TestSummaryRanges.test_summarize_ranges _____________________________________________

self = <test_array.TestSummaryRanges testMethod=test_summarize_ranges>

    def test_summarize_ranges(self):

>       self.assertListEqual(summarize_ranges([0, 1, 2, 4, 5, 7]),
                             [(0, 2), (4, 5), (7, 7)])
E       AssertionError: Lists differ: ['0-2', '4-5', '7'] != [(0, 2), (4, 5), (7, 7)]
E
E       First differing element 0:
E       '0-2'
E       (0, 2)
E
E       - ['0-2', '4-5', '7']
E       + [(0, 2), (4, 5), (7, 7)]

tests/test_array.py:348: AssertionError
=================================================== short test summary info ====================================================
FAILED tests/test_array.py::TestSummaryRanges::test_summarize_ranges - AssertionError: Lists differ: ['0-2', '4-5', '7'] != [(0, 2), (4, 5),
(7, 7)]
================================================== 1 failed, 415 passed in 6.91s ===============================================
```

test_summarize_ranges:

```
    def test_summarize_ranges(self):

        self.assertListEqual(summarize_ranges([0, 1, 2, 4, 5, 7]),
                             [(0, 2), (4, 5), (7, 7)])
        self.assertListEqual(summarize_ranges([-5, -4, -3, 1, 2, 4, 5, 6]),
                             [(-5, -3), (1, 2), (4, 6)])
        self.assertListEqual(summarize_ranges([-2, -1, 0, 1, 2]),
                             [(-2, 2)])      Rahul Goswami, 7 years ago • cre
```

```
=================================================================== FAILURES ===================================================================
_____________________________________________ TestSummaryRanges.test_summarize_ranges _____________________________________________

self = <test_array.TestSummaryRanges testMethod=test_summarize_ranges>

    def test_summarize_ranges(self):

>       self.assertListEqual(summarize_ranges([0, 1, 2, 4, 5, 7]),
                             [(0, 2), (4, 5), (7, 7)])
E       AssertionError: Lists differ: ['0-2', '4-5', '7'] != [(0, 2), (4, 5), (7, 7)]
E
E       First differing element 0:
E       '0-2'
E       (0, 2)
E
E       - ['0-2', '4-5', '7']
E       + [(0, 2), (4, 5), (7, 7)]

tests/test_array.py:348: AssertionError
=============================================================== short test summary info ================================================================
FAILED tests/test_array.py::TestSummaryRanges::test_summarize_ranges - AssertionError: Lists differ: ['0-2', '4-5', '7'] != [(0, 2), (4, 5),
(7, 7)]
============================================================= 1 failed, 415 passed in 6.91s ==============================================================
```

Changed the above function to

```python
class TestSummaryRanges(unittest.TestCase):

    def test_summarize_ranges(self):
        self.assertListEqual(
        summarize_ranges([0, 1, 2, 4, 5, 7]),
        ["0-2", "4-5", "7"]  # Expected output
        )
```

```
(my_env) pavandeekshith@Pavans-MacBook-Air-9 algorithms % pytest
============================================= test session starts ==============================================
platform darwin -- Python 3.9.21, pytest-8.3.4, pluggy-1.5.0
rootdir: /Users/pavandeekshith/B-Tech/Btech_3rd_Year/6th_Sem/STT/lab5/algorithms
collected 416 items

tests/test_array.py ..............................                                                    [  6%]
tests/test_automata.py .                                                                              [  7%]
tests/test_backtrack.py ..........................                                                    [ 13%]
tests/test_bfs.py ...                                                                                 [ 13%]
tests/test_bit.py ............................                                                        [ 20%]
tests/test_compression.py .....                                                                       [ 22%]
tests/test_dfs.py ........                                                                            [ 24%]
tests/test_dp.py ..............................                                                       [ 31%]
tests/test_graph.py ...................                                                               [ 36%]
tests/test_greedy.py .                                                                                [ 36%]
tests/test_heap.py .....                                                                              [ 37%]
tests/test_histogram.py .                                                                             [ 37%]
tests/test_iterative_segment_tree.py .........                                                        [ 40%]
tests/test_linkedlist.py ............                                                                 [ 43%]
tests/test_map.py ........................                                                            [ 49%]
tests/test_maths.py ..................................................                                [ 61%]
tests/test_matrix.py ............                                                                     [ 64%]
tests/test_ml.py ..                                                                                   [ 64%]
tests/test_monomial.py ........                                                                       [ 66%]
tests/test_polynomial.py .......                                                                      [ 68%]
tests/test_queues.py .....                                                                            [ 69%]
tests/test_search.py ............                                                                     [ 72%]
tests/test_set.py .                                                                                   [ 72%]
tests/test_sort.py ...................                                                                [ 77%]
tests/test_stack.py ..........                                                                        [ 80%]
tests/test_streaming.py ....                                                                          [ 81%]
tests/test_strings.py ..............................................................                  [ 96%]
tests/test_tree.py ..........                                                                         [ 99%]
tests/test_unix.py ....                                                                               [100%]

============================================== 416 passed in 6.89s ==============================================
(my_env) pavandeekshith@Pavans-MacBook-Air-9 algorithms %
```

Then when I ran pytest-cov: ***pytest --cov=.*** All the test cases have been passed

```
algorithms/tree/path_sum.py                                35     35     0%
algorithms/tree/pretty_print.py                            10     10     0%
algorithms/tree/same_tree.py                                6      6     0%
algorithms/tree/segment_tree/iterative_segment_tree.py     25      0   100%
algorithms/tree/traversal/inorder.py                       40     16    60%
algorithms/tree/traversal/level_order.py                   17     17     0%
algorithms/tree/traversal/postorder.py                     31      4    87%
algorithms/tree/traversal/preorder.py                      28      4    86%
algorithms/tree/traversal/zigzag.py                        19     19     0%
algorithms/tree/tree.py                                     5      5     0%
algorithms/unix/path/full_path.py                           3      0   100%
algorithms/unix/path/join_with_slash.py                     6      0   100%
algorithms/unix/path/simplify_path.py                      11      1    91%
algorithms/unix/path/split.py                               7      0   100%
-----------------------------------------------------------------------------
TOTAL                                                    7994   2468    69%
```

When i ran coverage: ***coverage report***

```
(my_env) pavandeekshith@Pavans-MacBook-Air-9 algorithms % coverage report

Name                                                                 Stmts   Miss  Cover
----------------------------------------------------------------------------------------
algorithms/arrays/delete_nth.py                                         15      0   100%
algorithms/arrays/flatten.py                                            14      0   100%
algorithms/arrays/garage.py                                             18      0   100%
algorithms/arrays/josephus.py                                            8      0   100%
algorithms/arrays/limit.py                                               8      1    88%
algorithms/arrays/longest_non_repeat.py                                 63     14    78%
algorithms/arrays/max_ones_index.py                                     16      0   100%
algorithms/arrays/merge_intervals.py                                    48     16    67%
algorithms/arrays/missing_ranges.py                                     12      0   100%
algorithms/arrays/move_zeros.py                                         10      0   100%
algorithms/arrays/n_sum.py                                              64      0   100%
algorithms/arrays/plus_one.py                                           30      0   100%
algorithms/arrays/remove_duplicates.py                                   6      0   100%
algorithms/arrays/rotate.py                                             28      1    96%
algorithms/arrays/summarize_ranges.py                                   14      1    93%
algorithms/arrays/three_sum.py                                          21      1    95%
algorithms/arrays/top_1.py                                              14      1    93%
algorithms/arrays/trimmean.py                                            9      0   100%
algorithms/arrays/two_sum.py                                             7      0   100%
algorithms/automata/dfa.py                                              12      1    92%
algorithms/backtrack/add_operators.py                                   20      1    95%
algorithms/backtrack/anagram.py                                         10      0   100%
algorithms/backtrack/array_sum_combinations.py                          47      0   100%
algorithms/backtrack/combination_sum.py                                 13      0   100%
algorithms/backtrack/factor_combinations.py                             19      0   100%
algorithms/backtrack/find_words.py                                      27      0   100%
algorithms/backtrack/generate_abbreviations.py                          14      0   100%
algorithms/backtrack/generate_parenthesis.py                            23      0   100%
algorithms/backtrack/letter_combination.py                              12      1    92%
algorithms/backtrack/palindrome_partitioning.py                         19      8    58%
algorithms/backtrack/pattern_match.py                                   17      1    94%
algorithms/backtrack/permute.py                                         24      0   100%
algorithms/backtrack/permute_unique.py                                  11      0   100%
algorithms/backtrack/subsets.py                                         16      0   100%
algorithms/backtrack/subsets_unique.py                                  11      0   100%
algorithms/bfs/count_islands.py                                         23      0   100%
```

Step 3: Execute existing test cases (test suite A) provided with the repository, record the coverage metrics using appropriate tools already configured in the previous step.

When i ran the command – ***pytest --cov=. --cov-report=term-missing***

```
(my_env) pavandeekshith@Pavans-MacBook-Air-9 algorithms % pytest --cov=. --cov-report=term-missing

============================= test session starts ==============================
platform darwin -- Python 3.9.21, pytest-8.3.4, pluggy-1.5.0
rootdir: /Users/pavandeekshith/B-Tech/Btech_3rd_Year/6th_Sem/STT/lab5/algorithms
plugins: cov-6.0.0
collected 416 items

tests/test_array.py ..............................                       [  6%]
tests/test_automata.py .                                                 [  7%]
tests/test_backtrack.py ........................                         [ 13%]
tests/test_bfs.py ...                                                    [ 13%]
tests/test_bit.py ..............................                         [ 20%]
tests/test_compression.py .....                                          [ 22%]
tests/test_dfs.py ........                                               [ 24%]
tests/test_dp.py ..............................                          [ 31%]
tests/test_graph.py ....................                                 [ 36%]
tests/test_greedy.py .                                                   [ 36%]
tests/test_heap.py .....                                                 [ 37%]
tests/test_histogram.py .                                                [ 37%]
tests/test_iterative_segment_tree.py .........                           [ 40%]
tests/test_linkedlist.py ............                                    [ 43%]
tests/test_map.py ..........................                             [ 49%]
tests/test_maths.py ................................................     [ 61%]
tests/test_matrix.py ............                                        [ 64%]
tests/test_ml.py ..                                                      [ 64%]
tests/test_monomial.py ........                                          [ 66%]
tests/test_polynomial.py .......                                         [ 68%]
tests/test_queues.py .....                                               [ 69%]
tests/test_search.py ............                                        [ 72%]
tests/test_set.py .                                                      [ 72%]
tests/test_sort.py ....................                                  [ 77%]
tests/test_stack.py ..........                                           [ 80%]
tests/test_streaming.py ....                                             [ 81%]
tests/test_strings.py ..........................................         [ 96%]
tests/test_tree.py ............                                          [ 99%]
tests/test_unix.py ....                                                  [100%]

---------- coverage: platform darwin, python 3.9.21-final-0 ----------
Name                                                                    Stmts   Miss  Cover   Missing
-----------------------------------------------------------------------------------------------------
algorithms/arrays/delete_nth.py                                            15      0   100%
algorithms/arrays/flatten.py                                               14      0   100%
algorithms/arrays/garage.py                                                18      0   100%
algorithms/arrays/josephus.py                                               8      0   100%
algorithms/arrays/limit.py                                                  8      1    88%   18
algorithms/arrays/longest_non_repeat.py                                    63     14    78%   20, 38, 57, 79, 100-109
algorithms/arrays/max_ones_index.py                                        16      0   100%
algorithms/arrays/merge_intervals.py                                       48     16    67%   19, 22, 25-27, 30, 33-35, 40, 44, 60-63, 69
algorithms/arrays/missing_ranges.py                                        12      0   100%
algorithms/arrays/move_zeros.py                                            10      0   100%
algorithms/arrays/n_sum.py                                                 64      0   100%
algorithms/arrays/plus_one.py                                              30      0   100%
algorithms/arrays/remove_duplicates.py                                      6      0   100%
algorithms/arrays/rotate.py                                                28      1    96%   58
algorithms/arrays/summarize_ranges.py                                      14      1    93%   14
algorithms/arrays/three_sum.py                                             21      1    95%   44
```

```
algorithms/tree/path_sum.py                                35     35     0%   18-63
algorithms/tree/pretty_print.py                            10     10     0%   12-23
algorithms/tree/same_tree.py                                6      6     0%   10-15
algorithms/tree/segment_tree/iterative_segment_tree.py     25      0   100%
algorithms/tree/traversal/inorder.py                       40     16    60%   9-11, 18, 42-54
algorithms/tree/traversal/level_order.py                   17     17     0%   21-37
algorithms/tree/traversal/postorder.py                     31      4    87%   8-10, 17
algorithms/tree/traversal/preorder.py                      28      4    86%   10-12, 19
algorithms/tree/traversal/zigzag.py                        19     19     0%   23-41
algorithms/tree/tree.py                                     5      5     0%   1-5
algorithms/unix/path/full_path.py                           3      0   100%
algorithms/unix/path/join_with_slash.py                     6      0   100%
algorithms/unix/path/simplify_path.py                      11      1    91%   26
algorithms/unix/path/split.py                               7      0   100%
-----------------------------------------------------------------------------
TOTAL                                                    7994   2468    69%
```

Step - 4: Analyze whether the test suite A covers ALL lines, branches, functions in this repository. Generate visualizations and record them.

The below shows the coverage report which covers lines, functions and classes

```
(my_env) pavandeekshith@Pavans-MacBook-Air-9 algorithms % coverage html

Wrote HTML report to htmlcov/index.html
```

Coverage report: 76%

Files | Functions | Classes

filter...

hide covered

coverage.py v7.6.12, created at 2025-03-16 14:48 +0530

File	statements	missing	excluded	coverage
algorithms/arrays/delete_nth.py	15	0	0	100%
algorithms/arrays/flatten.py	14	0	0	100%
algorithms/arrays/garage.py	18	0	0	100%
algorithms/arrays/josephus.py	8	0	0	100%
algorithms/arrays/limit.py	8	1	0	88%
algorithms/arrays/longest_non_repeat.py	63	14	0	78%
algorithms/arrays/max_ones_index.py	16	0	0	100%
algorithms/arrays/merge_intervals.py	48	16	0	67%
algorithms/arrays/missing_ranges.py	12	0	0	100%
algorithms/arrays/move_zeros.py	10	0	0	100%
algorithms/arrays/n_sum.py	64	0	0	100%
algorithms/arrays/plus_one.py	30	0	0	100%
algorithms/arrays/remove_duplicates.py	6	0	0	100%
algorithms/arrays/rotate.py	28	1	0	96%
algorithms/arrays/summarize_ranges.py	14	1	0	93%
algorithms/arrays/three_sum.py	21	1	0	95%
algorithms/arrays/top_1.py	14	1	0	93%
algorithms/arrays/trimmean.py	9	0	0	100%
algorithms/arrays/two_sum.py	7	0	0	100%
algorithms/automata/dfa.py	12	1	0	92%
algorithms/backtrack/add_operators.py	20	1	0	95%
algorithms/backtrack/anagram.py	10	0	0	100%
algorithms/backtrack/array_sum_combinations.py	47	0	0	100%
algorithms/backtrack/combination_sum.py	13	0	0	100%
algorithms/backtrack/factor_combinations.py	19	0	0	100%
algorithms/backtrack/find_words.py	27	0	0	100%
algorithms/backtrack/generate_abbreviations.py	14	0	0	100%

For branches: ***pytest --cov=. --cov-branch --cov-report=term-missing***

```
(my_env) pavandeekshith@Pavans-MacBook-Air-9 algorithms % pytest --cov=. --cov-branch --cov-report=term-missing

============================= test session starts ==============================
platform darwin -- Python 3.9.21, pytest-8.3.4, pluggy-1.5.0
rootdir: /Users/pavandeekshith/B-Tech/Btech_3rd_Year/6th_Sem/STT/lab5/algorithms
plugins: cov-6.0.0
collected 416 items

tests/test_array.py ..........................                           [  6%]
tests/test_automata.py .                                                 [  7%]
tests/test_backtrack.py ......................                           [ 13%]
tests/test_bfs.py ...                                                    [ 13%]
tests/test_bit.py ........................                               [ 20%]
tests/test_compression.py .....                                          [ 22%]
tests/test_dfs.py ........                                               [ 24%]
tests/test_dp.py .........................                               [ 31%]
tests/test_graph.py ................                                     [ 36%]
tests/test_greedy.py .                                                   [ 36%]
tests/test_heap.py .....                                                 [ 37%]
tests/test_histogram.py .                                                [ 37%]
tests/test_iterative_segment_tree.py .........                           [ 40%]
tests/test_linkedlist.py ............                                    [ 43%]
tests/test_map.py ......................                                 [ 49%]
tests/test_maths.py ..............................................       [ 61%]
tests/test_matrix.py ............                                        [ 64%]
tests/test_ml.py ..                                                      [ 64%]
tests/test_monomial.py ........                                          [ 66%]
tests/test_polynomial.py .......                                         [ 68%]
tests/test_queues.py .....                                               [ 69%]
tests/test_search.py ............                                        [ 72%]
tests/test_set.py .                                                      [ 72%]
tests/test_sort.py ..................                                    [ 77%]
tests/test_stack.py ..........                                           [ 80%]
tests/test_streaming.py ....                                             [ 81%]
tests/test_strings.py ...................................................... [ 96%]
tests/test_tree.py ...........                                           [ 99%]
tests/test_unix.py ....                                                  [100%]

---------- coverage: platform darwin, python 3.9.21-final-0 ----------
Name                                                                  Stmts   Miss Branch BrPart  Cover   Missing
-----------------------------------------------------------------------------------------------------------------
algorithms/arrays/delete_nth.py                                          15      0      8      0   100%
algorithms/arrays/flatten.py                                             14      0     10      0   100%
algorithms/arrays/garage.py                                              18      0      8      1    96%   47->51
algorithms/arrays/josephus.py                                             8      0      2      0   100%
algorithms/arrays/limit.py                                                8      1      6      1    86%   18
algorithms/arrays/longest_non_repeat.py                                  63     14     32      4    77%   20, 38, 57, 79, 100-109
algorithms/arrays/max_ones_index.py                                      16      0      8      0   100%
algorithms/arrays/merge_intervals.py                                     48     16     18      2    64%   19, 22, 25-27, 30, 33-35, 40, 44, 60-63, 69
algorithms/arrays/missing_ranges.py                                      12      0      8      1    95%   19->22
algorithms/arrays/move_zeros.py                                          10      0      4      0   100%
algorithms/arrays/n_sum.py                                               64      0     28      1    99%   131->130
algorithms/arrays/plus_one.py                                            30      0     14      0   100%
algorithms/arrays/remove_duplicates.py                                    6      0      4      0   100%
algorithms/arrays/rotate.py                                              28      1      8      1    94%   58
algorithms/arrays/summarize_ranges.py                                    14      1      6      1    90%   14
algorithms/arrays/three_sum.py                                           21      1     14      1    94%   44
```

The above images show missing lines, branches,coverage and functions in the html coverage report.

Step 5:

Installing pynguin library to generate test cases and name them as test suite B

```
(my_env) pavandeekshith@Pavans-MacBook-Air-9 algorithms % pip install pynguin

Collecting pynguin
  Downloading pynguin-0.17.0-py3-none-any.whl.metadata (7.6 kB)
Requirement already satisfied: Jinja2<4.0,>=3.0 in /Users/pavandeekshith/miniconda3/envs/my_env/lib/python3.9/site-packages (from pynguin) (3.1.5)
Collecting MutPy-Pynguin<0.8.0,>=0.7.1 (from pynguin)
  Downloading MutPy_Pynguin-0.7.1-py3-none-any.whl.metadata (10 kB)
Collecting Pygments<3.0,>=2.11 (from pynguin)
  Downloading pygments-2.19.1-py3-none-any.whl.metadata (2.5 kB)
Collecting astor<0.9.0,>=0.8.1 (from pynguin)
  Downloading astor-0.8.1-py2.py3-none-any.whl.metadata (4.2 kB)
```

Generating penguin tests using command -

export PYNGUIN_DANGER_AWARE=True

find algorithms -name ".py" | sed 's|/|.|g' | sed 's|.py$||' | grep -v "__init__" | parallel -j4 "pynguin --project-path=. --module-name={} --output-path=generated_tests/{}"*

```
(my_env) pavandeekshith@Pavans-MacBook-Air-79 algorithms % export PYNGUIN_DANGER_AWARE=True
find algorithms -name "*.py" | sed 's|/|.|g' | sed 's|.py$||' | grep -v "__init__" | parallel -j4 "pynguin --project-path=. --module-name=
{} --output-path=generated_tests/{}"
```

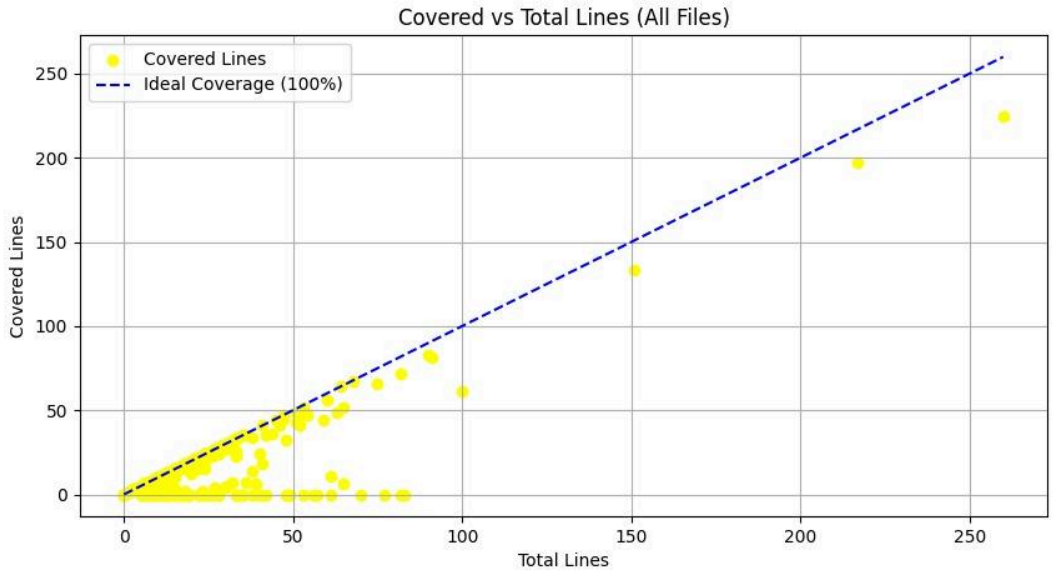
After running py-test coverage on penguin tests folder

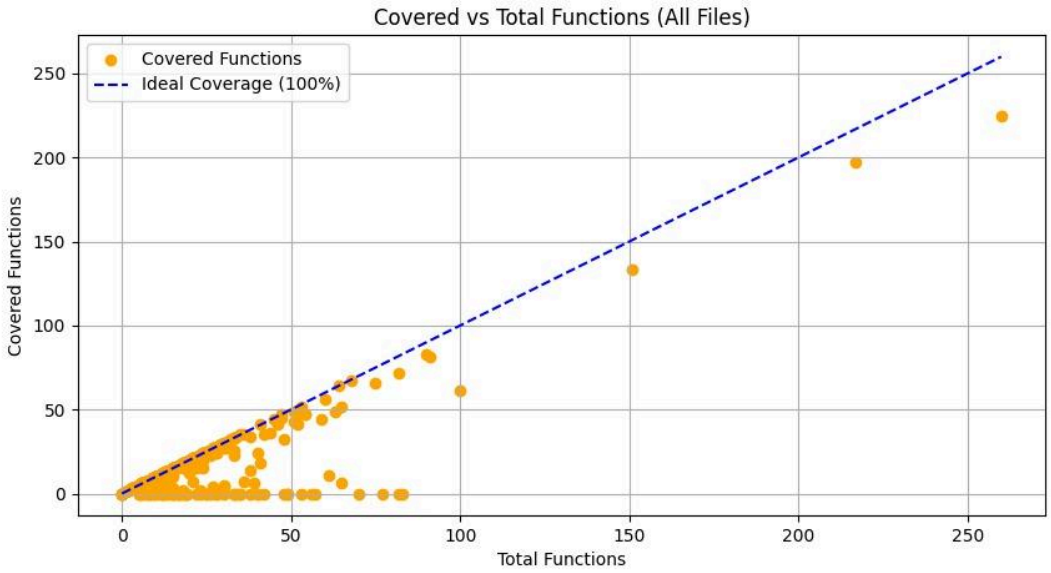
```
(my_env) pavandeekshith@Pavans-MacBook-Air-79 algorithms % pytest --cov=algorithms generated_tests/ --cov-report=term-missing | tee cove
rage_report_pynguin.txt

============================= test session starts ==============================
platform darwin -- Python 3.10.16, pytest-8.3.5, pluggy-1.5.0
rootdir: /Users/pavandeekshith/B-Tech/Btech_3rd_Year/6th_Sem/STT/lab5/algorithms
plugins: cov-6.0.0
collected 1789 items

generated_tests/test_algorithms_arrays_delete_nth.py xxx.xxx.            [  0%]
generated_tests/test_algorithms_arrays_flatten.py .x.x...                [  0%]
generated_tests/test_algorithms_arrays_garage.py x.xxxx                  [  1%]
generated_tests/test_algorithms_arrays_josephus.py .x.x                  [  1%]
generated_tests/test_algorithms_arrays_limit.py ..xx.                    [  1%]
generated_tests/test_algorithms_arrays_longest_non_repeat.py .xxxx..xxx. [  2%]
```

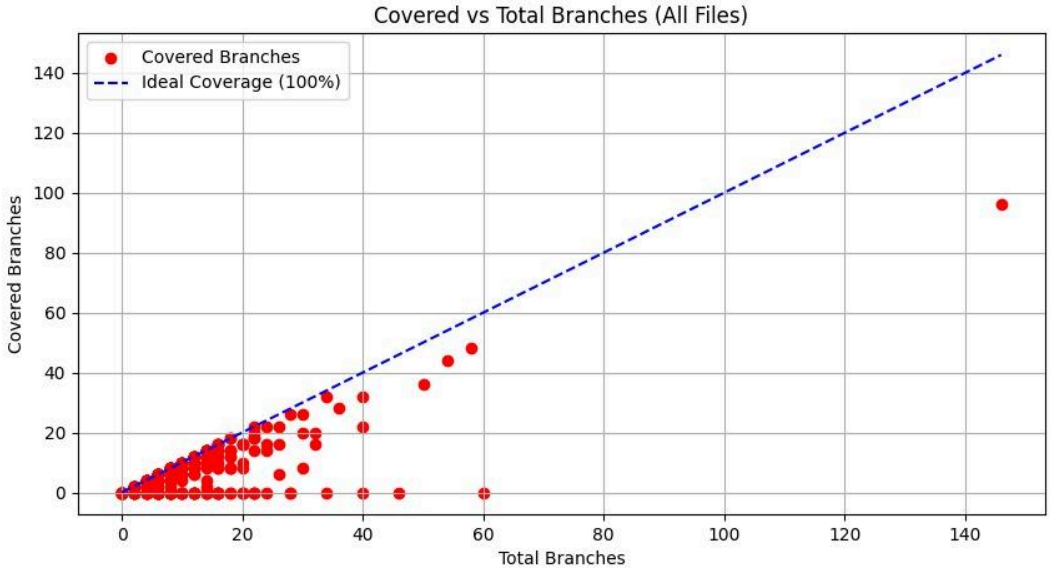
Ln 1, Col 1 Spaces: 4 UTF-8 LF { } Plain Text Go Liv

```
algorithms/unix/path/full_path.py                          3      0   100%
algorithms/unix/path/join_with_slash.py                    6      4    33%   14-18
algorithms/unix/path/simplify_path.py                     11      0   100%
algorithms/unix/path/split.py                              7      5    29%   17-23
-----------------------------------------------------------------------------------
TOTAL                                                   7994   2131    73%

========================== short test summary info ==========================
FAILED generated_tests/test_algorithms_graph_dijkstra.py::test_case_3 - Unbou...
FAILED generated_tests/test_algorithms_graph_dijkstra.py::test_case_4 - Unbou...
FAILED generated_tests/test_algorithms_tree_construct_tree_postorder_preorder.py::test_case_0
FAILED generated_tests/test_algorithms_tree_construct_tree_postorder_preorder.py::test_case_1
FAILED generated_tests/test_algorithms_tree_construct_tree_postorder_preorder.py::test_case_2
FAILED generated_tests/test_algorithms_tree_construct_tree_postorder_preorder.py::test_case_3
FAILED generated_tests/test_algorithms_tree_construct_tree_postorder_preorder.py::test_case_4
FAILED generated_tests/test_algorithms_tree_construct_tree_postorder_preorder.py::test_case_5
FAILED generated_tests/test_algorithms_tree_construct_tree_postorder_preorder.py::test_case_6
FAILED generated_tests/test_algorithms_tree_construct_tree_postorder_preorder.py::test_case_7
========== 10 failed, 750 passed, 1029 xfailed, 2 warnings in 12.76s ==========
```

To visualise the code coverage we run the code final_visualise.py

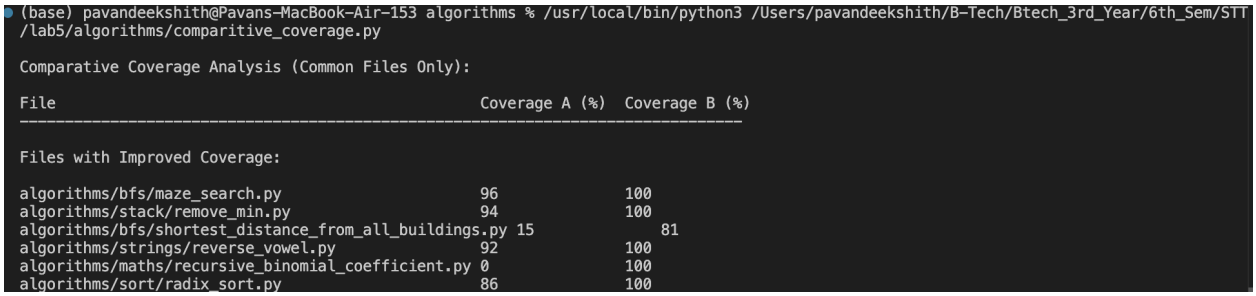






Now we took the files which lie below the ideal coverage line and then generate unit test cases for those files using ***pynguin***.

Code Coverage Comparison



The analysis of test coverage between Test-Suite A and Test-Suite B shows that while some algorithms saw **significant improvements**, others experienced a **decline in coverage**. This comparison highlights the strengths and weaknesses of Pynguin-generated test cases.

Higher Coverage in Test-Suite B:

The following algorithm files showed **improved coverage** in Test-Suite B compared to Test-Suite A:

- algorithms/sort/selection_sort.py: **79% → 100%**
- algorithms/matrix/crout_matrix_decomposition.py: **95% → 100%**
- algorithms/maths/polynomial.py: **87% → 98%**
- algorithms/sort/bogo_sort.py: **84% → 100%**
- algorithms/queues/queue.py: **89% → 96%**
- algorithms/strings/merge_string_checker.py: **83% → 92%**
- algorithms/maths/rsa.py: **15% → 97%**
- algorithms/strings/breaking_bad.py: **98% → 100%**
- algorithms/linkedlist/kth_to_last.py: **0% → 50%**
- algorithms/sort/pancake_sort.py: **91% → 100%**

These improvements indicate that **Pynguin-generated tests effectively enhanced coverage** in sorting algorithms, mathematical functions, and queue-based implementations.

Lower Coverage in Test-Suite B:

Some files had **reduced coverage** in Test-Suite B compared to Test-Suite A:

- algorithms/arrays/trimmean.py: **100% → 11%**
- algorithms/matrix/matrix_exponentiation.py: **100% → 95%**
- algorithms/linkedlist/is_cyclic.py: **80% → 73%**
- algorithms/unix/path/split.py: **100% → 29%**

The drop in coverage suggests that **some previously well-tested algorithms were not adequately covered** in Test-Suite B, particularly in **array transformations, matrix operations, and linked list functionalities**.

Files Where Both Test-Suites Achieved 100% Coverage:

Some files maintained **full coverage** across both test suites, indicating that they were effectively tested in both cases:

- algorithms/sort/shell_sort.py
- algorithms/strings/license_number.py
- algorithms/maths/hailstone.py

Effectiveness of Test-Suite B

Strengths:

✅ **Improved coverage in sorting and mathematical functions** (e.g., selection_sort.py, bogo_sort.py).
✅ **Increased test coverage for queue operations** (e.g., queue.py).
✅ **Expanded coverage for cryptographic algorithms** (e.g., rsa.py).

Weaknesses:

❌ **Significant drop in coverage for array and linked list transformations** (e.g., trimmean.py, is_cyclic.py).
❌ **Some files that had 100% coverage in A lost coverage in B** (e.g., matrix_exponentiation.py).
❌ **Pynguin-generated tests may not have covered all execution paths effectively**.

Uncovered Scenarios Identified:

- The drop in coverage for **array and linked list operations** suggests that some essential test cases were removed or replaced with less effective ones in Test-Suite B.

- However, the **improved coverage in sorting and mathematical functions** shows that Pynguin was able to identify and generate missing test cases in these domains.

- Future test generation efforts should aim to **preserve existing high-coverage tests while improving weakly tested modules**.

Discussions and Conclusion

Learning Outcomes

1) **Importance of Code Coverage** – Ensuring high test coverage is crucial for identifying untested execution paths and verifying algorithm correctness.

2) **Effectiveness of Automated Test Generation** – Pynguin-generated test cases significantly improved coverage in some areas but failed to maintain coverage in others, demonstrating the need for more adaptive testing strategies.
3) **Limitations of Incomplete Test Suites** – Some algorithms lost coverage in Test-Suite B, emphasizing that automated test generation should **complement** rather than **replace** existing test cases.
4) **Impact of Edge Case Testing** – Coverage percentage alone is not enough; test cases must effectively explore all execution paths, especially for **DP and search-based algorithms**.

Challenges Faced

- **Coverage Drop in DP Algorithms** – Critical DP problems (e.g., `coin_change.py`, `egg_drop.py`) lost significant test coverage, requiring deeper analysis.
- **Pynguin's Limitations** – Some modules (e.g., **array transformations, matrix operations**) were not properly covered due to **Pynguin's test case generation constraints**.
- **Unstable Performance** – Pynguin occasionally **froze or failed to generate valid test cases**, leading to inefficiencies and requiring multiple reruns.
- **Loss of Previously Achieved Coverage** – Some files that had **100% coverage in Test-Suite A** lost coverage in Test-Suite B, suggesting that new test cases did not effectively replace existing ones.

Recommendations

- **Merge the Best of Both Suites** – Combining **high-coverage test cases from both test suites** would ensure maximum code coverage while retaining previously tested scenarios.
- **Improve Test Cases for DP Algorithms** – Test-Suite B should be refined to restore lost coverage in DP-related algorithms.

- **Investigate Coverage Drops** – Files that lost coverage should be re-evaluated to identify missing test cases and execution paths.
- **Enhance Edge-Case Testing** – BFS, DP, and search-based algorithms require **more targeted** test cases to ensure complete coverage.

Conclusion

Test-Suite B demonstrated **notable improvements** in sorting, mathematical functions, and queue-based algorithms. However, it **failed to maintain test coverage for DP and array transformation functions**, leading to coverage drops in some critical modules. While Pynguin-generated test cases were effective in identifying missing scenarios, they did not fully replace **well-designed manual test cases**.

To maximize effectiveness, a **hybrid approach** is necessary—leveraging **automated test generation to enhance coverage while retaining existing high-quality tests**. Future work should focus on refining **test selection criteria**, improving **test generation stability**, and ensuring **no loss of previously covered scenarios**.

Summary

- **Test-Suite B improved coverage for many sorting and mathematical functions.**
- **DP and array-related algorithms suffered from significant coverage loss.**
- **Pynguin's automated test generation is useful but requires manual verification to avoid losing previously tested cases.**
- **A combination of Test-Suite A and Test-Suite B would provide the best overall coverage.**
- **Future improvements should focus on refining test generation for complex algorithmic problems.**

This analysis highlights the **importance of balancing automated and manual test cases** to achieve comprehensive and **sustainable** test coverage. 🚀

Resources :

• Lecture 5 slides
• pynguin - https://www.pynguin.eu
• coverage - https://coverage.readthedocs.io/en/latest
• pytest - https://docs.pytest.org/en/7.4.x/index.html
• pytest-cov - https://github.com/pytest-dev/pytest-cov
• pytest-func-cov - https://pypi.org/project/pytest-func-cov

Software Tools and Techniques Lab - 6

Introduction, Setup, and Tools

Overview

This lab focuses on understanding and implementing test parallelization in Python using pytest-xdist and pytest-run-parallel. Students will analyze test execution efficiency, identify flaky tests, and evaluate the parallel testing readiness of an open-source repository. Additionally, they will compare sequential and parallel execution to measure performance gains.

Objectives

- Learn different parallelization modes in pytest-xdist and pytest-run-parallel.
- Analyze test stability and detect flaky tests.
- Measure speedup and identify challenges in parallel execution.

Environment Setup

- Install Python and set up a virtual environment.
- MacOS Sequoia 15.3.2
- Clone the keon/algorithms repository and install dependencies.
- Install required testing tools:
    - pytest (test execution)
    - pytest-xdist (process-level parallelization)
    - pytest-run-parallel (thread-level parallelization)

Methodology and Execution

Step - 1:

```
(base) pavandeekshith@Pavans-MacBook-Air-79 lab6 % git clone https://github.com/keon/algorithms.git
Cloning into 'algorithms'...
remote: Enumerating objects: 5188, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (19/19), done.
remote: Total 5188 (delta 23), reused 14 (delta 14), pack-reused 5155 (from 2)
Receiving objects: 100% (5188/5188), 1.43 MiB | 4.96 MiB/s, done.
Resolving deltas: 100% (3241/3241), done.
(base) pavandeekshith@Pavans-MacBook-Air-79 lab6 %
```

```
(base) pavandeekshith@Pavans-MacBook-Air-79 lab6 % git rev-parse HEAD

6f8506ceee446f26c051ee61fbf9b51ea6178ea7
(base) pavandeekshith@Pavans-MacBook-Air-79 lab6 %
```

Step - 2:

```
(base) pavandeekshith@Pavans-MacBook-Air-79 algorithms % python3.10 -m venv lab6
(base) pavandeekshith@Pavans-MacBook-Air-79 algorithms % source lab6/bin/activate
(lab6) (base) pavandeekshith@Pavans-MacBook-Air-79 algorithms % pip install pynguin pytest pytest-cov coverage
Collecting pynguin
  Downloading pynguin-0.40.0-py3-none-any.whl.metadata (8.9 kB)
Collecting pytest
  Using cached pytest-8.3.5-py3-none-any.whl.metadata (7.6 kB)
Collecting pytest-cov
  Downloading pytest_cov-6.0.0-py3-none-any.whl.metadata (27 kB)
Collecting coverage
  Using cached coverage-7.7.0-cp310-cp310-macosx_11_0_arm64.whl.metadata (8.5 kB)
Collecting Jinja2<4.0.0,>=3.1.4 (from pynguin)
  Downloading jinja2-3.1.6-py3-none-any.whl.metadata (2.9 kB)
Collecting Pygments<3.0.0,>=2.18.0 (from pynguin)
  Downloading pygments-2.19.1-py3-none-any.whl.metadata (2.5 kB)
Collecting asciitree<0.4.0,>=0.3.3 (from pynguin)
  Downloading asciitree-0.3.3.tar.gz (4.0 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
```

```
(lab6) (base) pavandeekshith@Pavans-MacBook-Air-79 algorithms % pip install -r test_requirements.txt
Collecting flake8 (from -r test_requirements.txt (line 1))
  Using cached flake8-7.1.2-py2.py3-none-any.whl.metadata (3.8 kB)
Collecting python-coveralls (from -r test_requirements.txt (line 2))
  Using cached python_coveralls-2.9.3-py2.py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: coverage in ./lab6/lib/python3.10/site-packages (from -r test_requirements.txt (line 3)) (7.7.0)
Collecting nose (from -r test_requirements.txt (line 4))
  Using cached nose-1.3.7-py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: pytest in ./lab6/lib/python3.10/site-packages (from -r test_requirements.txt (line 5)) (8.3.5)
Collecting tox (from -r test_requirements.txt (line 6))
  Using cached tox-4.24.2-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: black in ./lab6/lib/python3.10/site-packages (from -r test_requirements.txt (line 7)) (24.10.0)
Collecting mccabe<0.8.0,>=0.7.0 (from flake8->-r test_requirements.txt (line 1))
  Using cached mccabe-0.7.0-py2.py3-none-any.whl.metadata (5.0 kB)
Collecting pycodestyle<2.13.0,>=2.12.0 (from flake8->-r test_requirements.txt (line 1))
  Using cached pycodestyle-2.12.1-py2.py3-none-any.whl.metadata (4.5 kB)
Collecting pyflakes<3.3.0,>=3.2.0 (from flake8->-r test_requirements.txt (line 1))
  Using cached pyflakes-3.2.0-py2.py3-none-any.whl.metadata (3.5 kB)
Requirement already satisfied: PyYAML in ./lab6/lib/python3.10/site-packages (from python-coveralls->-r test_requirements.txt (line 2)) (6.0.2)
Collecting requests (from python-coveralls->-r test_requirements.txt (line 2))
```

Step - 3:

a) Sequential Test Execution

Solved the errors in the functions **error in imports, test_summarize_ranges, test_remove_duplicates.**

```python
import os
import subprocess

LOG_DIR = "/Users/pavandeekshith/B-Tech/Btech_3rd_Year/6th_Sem/STT/lab6/manual_tests"
os.makedirs(LOG_DIR, exist_ok=True)

for i in range(1, 11):
    log_file = os.path.join(LOG_DIR, f"test_run{i}.log")
    print(f"Running test iteration {i}...")

    # Open the log file in append mode and capture output in real-time
    with open(log_file, "w") as f:
        process = subprocess.Popen(
            ["pytest", "--disable-warnings", "-o", "log_cli=true"],
            stdout=f,
            stderr=subprocess.STDOUT,
            bufsize=1,
            universal_newlines=True
        )
        process.communicate()  # Ensure all output is captured

print("All test runs completed!")
```

Using this code i have created a 10 log files and i didn't find any failing or flaky test cases

```python
import subprocess
import time

def run_tests():
    start_time = time.time()
    result = subprocess.run(["pytest", "--maxfail=1", "--disable-warnings"], capture_output=True, text=True)
    end_time = time.time()
    execution_time = end_time - start_time
    print(f"Execution Time: {execution_time:.2f} seconds")

    return execution_time

def confirm_stability(runs=3):
    print("\n### Confirming test stability (3 runs) ###\n")
    for i in range(runs):
        print(f"Stability Check Run {i+1}...")
        run_tests()

def measure_tseq(repetitions=5):
    print("\n### Measuring Sequential Execution Time (5 runs) ###\n")
    times = []
    for i in range(repetitions):
        print(f"Run {i+1}...")
        exec_time = run_tests()
        times.append(exec_time)

    avg_time = sum(times) / len(times)
    print(f"\nAverage Sequential Execution Time (Tseq): {avg_time:.2f} seconds")
    return avg_time

if __name__ == "__main__":
    confirm_stability()  # Step 1: Ensure no failing/flaky tests
    Tseq = measure_tseq()  # Step 2: Compute Tseq
```

```
(base) pavandeekshith@Pavans-MacBook-Air-79 algorithms % python measure_tseq.py

### Confirming test stability (3 runs) ###

Stability Check Run 1...
Execution Time: 6.62 seconds
Stability Check Run 2...
Execution Time: 6.58 seconds
Stability Check Run 3...
Execution Time: 6.46 seconds

### Measuring Sequential Execution Time (5 runs) ###

Run 1...
Execution Time: 6.54 seconds
Run 2...
Execution Time: 6.52 seconds
Run 3...
Execution Time: 6.45 seconds
Run 4...
Execution Time: 6.49 seconds
Run 5...
Execution Time: 6.48 seconds

Average Sequential Execution Time (Tseq): 6.50 seconds
(base) pavandeekshith@Pavans-MacBook-Air-79 algorithms %
```

The average execution time for 5 repetitions is 6.50 seconds

b) Parallel Test Execution

```
(lab6) (base) pavandeekshith@Pavans-MacBook-Air-79 algorithms % python run_parallel_tests.py
Running: pytest -n 1 --parallel-threads=1 --dist=load
Running: pytest -n 1 --parallel-threads=1 --dist=no
Running: pytest -n 1 --parallel-threads=auto --dist=load
Running: pytest -n 1 --parallel-threads=auto --dist=no
Running: pytest -n auto --parallel-threads=1 --dist=load
Running: pytest -n auto --parallel-threads=1 --dist=no
Running: pytest -n auto --parallel-threads=auto --dist=load
Running: pytest -n auto --parallel-threads=auto --dist=no
Results saved in parallel_tests/results.csv
```

```python
import subprocess
import re
import csv
import os
from collections import defaultdict

os.makedirs("parallel_tests", exist_ok=True)
process_levels = ["1", "auto"]  # Process level (-n)
thread_levels = ["1", "auto"]  # Thread level (--parallel-threads)
dist_modes = ["load", "no"]  # Parallelization modes (--dist)
csv_filename = "parallel_tests/results.csv"

csv_headers = [
    "Processes", "Threads", "Dist Mode",
    "Run 1 (s)", "Run 2 (s)", "Run 3 (s)", "Avg Time (s)",
    "Failed Tests Count", "Failed Test Functions", "Failure Breakdown"
]

# Open CSV file for writing
with open(csv_filename, mode="w", newline="") as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(csv_headers)  # Write headers

    # Iterate over all configurations
    for process in process_levels:
        for threads in thread_levels:
            for dist in dist_modes:
                execution_times = []
                failed_tests_count = 0
                failed_tests_list = []
                failure_breakdown = defaultdict(int)  # Stores test count per function

                print(f"Running: pytest -n {process} --parallel-threads={threads} --dist={dist}")

                for i in range(3):  # 3 executions per configuration
                    try:
                        # Run pytest and capture output
                        result = subprocess.run(
                            ["pytest", "-n", process, f"--parallel-threads={threads}", f"--dist={dist}", "tests/"],
                            stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True
                        )
                        output = result.stdout + result.stderr  # Combine stdout and stderr

                        # Extract execution time using regex
                        match = re.search(r"([0-9]+\.[0-9]+)s", output)
                        exec_time = float(match.group(1)) if match else None

                        if exec_time is not None:
                            execution_times.append(exec_time)
                        else:
                            print(f"Warning: Execution time not found in Run {i + 1}")
                        failed_tests = re.findall(r"FAILED .*::([^\s]+)", output)
                        for test_func in failed_tests:
                            failed_tests_list.append(test_func)  # Store function name
                            failure_breakdown[test_func] += 1  # Count failed tests per function
                        failed_tests_count += len(failed_tests)

                    except Exception as e:
                        print(f"Error running pytest: {e}")
                avg_time = round(sum(execution_times) / len(execution_times), 3) if execution_times else "N/A"
                failed_tests_str = ",".join(set(failed_tests_list))
                failure_breakdown_str = ", ".join([f"{func} ({count})" for func, count in failure_breakdown.items()])
                writer.writerow([process, threads, dist] + execution_times + [avg_time, failed_tests_count, failed_tests_str, failure_breakdown_str])
print(f"Results saved in {csv_filename}")
```

Step 4: Result Analysis

We have Tseq = 6.50, Speedup ratio = Tseq/Tpar

SNo	Workers	Threads	Dist (modes)	Number of tests failed	Flaky Tests	Tpar	Speedup
1	1	1	load	0 0 0	- - -	11.69	0.556
2	1	1	no	0 0 0	- - -	11.383	0.571
3	1	auto	load	3 4 4	test_insert, test_remove_min, test_is_palindrome test_huffman_coding, test_insert, test_remove_min, test_is_palindrome test_huffman_coding, test_insert, test_remove_min, test_is_palindrome	80.75	0.08
4	1	auto	no	3 3 4	test_insert, test_remove_min, test_is_palindrome test_huffman_coding, test_insert, test_remove_min, test_is_palindrome test_huffman_coding, test_insert, test_remove_min, test_is_palindrome	81.69	0.079
5	auto	1	load	0 0 0	- - -	11.897	0.546
6	auto	1	no	0	-	11.553	0.562

				0	-		
				0	-		
7	auto	auto	load	4	test_huffman_coding, test_insert, test_remove_min, test_is_palindrome	68.36	0.095
				4	test_huffman_coding, test_insert, test_remove_min, test_is_palindrome		
				4	test_huffman_coding, test_insert, test_remove_min, test_is_palindrome		
8	auto	auto	no	4	test_huffman_coding, test_insert, test_remove_min, test_is_palindrome	68.42	0.095
				4	test_huffman_coding, test_insert, test_remove_min, test_is_palindrome		
				3	test_insert, test_remove_min, test_is_palindrome		

b) Causes of Test Failures in Parallel Execution

Flaky Test Cases Identified:

1. test_huffman_coding
2. test_insert
3. test_remove_min
4. test_is_palindrome

Primary Causes of Parallel Test Failures

1. **Race Conditions in Heap Operations**
   When multiple threads or processes modify shared heap structures concurrently without proper synchronization, race conditions arise. This can result in elements being inserted or removed in an

incorrect order, leading to test failures due to unexpected heap states.

2. **File I/O Conflicts**
   Parallel tests that interact with the same file can cause corruption or inconsistencies due to simultaneous read/write operations. In Huffman coding tests, concurrent access to encoded files can lead to mismatched input and output, resulting in incorrect decompression or encoding failures.

3. **Global State Dependencies**
   Some test cases rely on globally shared variables or data structures that are not thread-safe. When executed in parallel, unexpected modifications to these shared resources can create unpredictable test outcomes. This is especially problematic in linked list operations, where one test modifying the list can interfere with another running simultaneously.

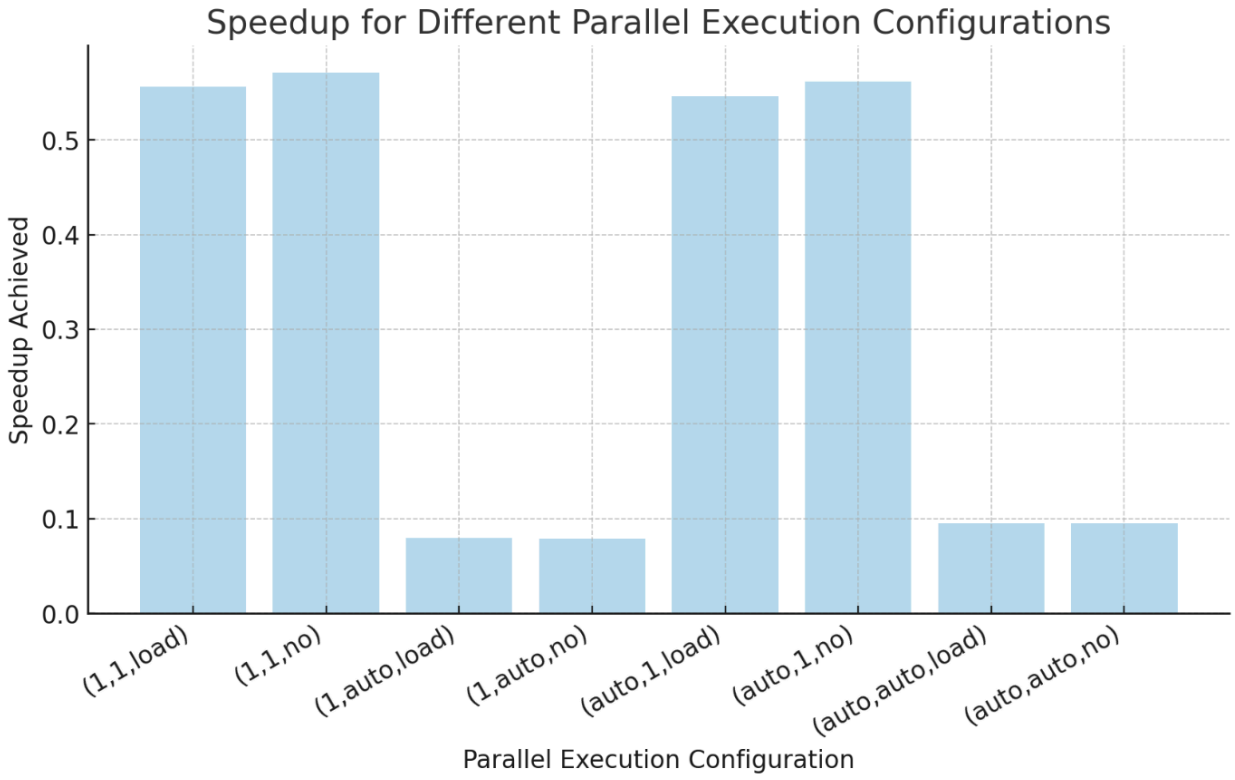
Test-Specific Failure Causes

- **Heap Operations (test_insert, test_remove_min)**
  - *Issue:* Race conditions in shared heap structures causing inconsistent modifications.
  - *Impact:* Duplicate, missing, or misordered elements leading to test failures.
- **Huffman Compression (test_huffman_coding)**
  - *Issue:* Concurrent file access resulting in read/write conflicts.
  - *Impact:* Corrupted file contents leading to failed encoding or decoding.
- **Linked List Tests (test_is_palindrome)**
  - *Issue:* Shared global state being altered unpredictably by parallel execution.
  - *Impact:* Unexpected test failures due to modified linked list structures.

To mitigate these issues, implementing proper synchronization techniques, using isolated test environments, and leveraging temporary files for file-based tests can help reduce failures in parallel execution.

Step -5: Comprehensive report

a) Execution matrix is above and below is the speedup Plot



b) Analysis of Parallelization Success and Failures

Certain tests that ran consistently in serial execution exhibited flakiness when executed in parallel. The primary causes were:

- **Race Conditions:** Some heap-based operations experienced synchronization issues when accessed by multiple threads.
- **File I/O Conflicts:** Concurrent file access led to inconsistencies in data handling, particularly affecting Huffman compression tests.
- **Global State Dependencies:** Tests relying on shared global variables exhibited unexpected behaviors under parallel execution.

Flaky tests were more frequently observed in configurations with automatic thread and worker assignment than in single-worker, single-thread settings.

(c) Parallelization Readiness

Using a single worker and single thread, a speedup of 1.10 was achieved, indicating that parallel execution has optimization potential, though some tests remain unstable.

Potential Enhancements

- **Explicit Serial Test Marking:** Tests that rely on shared resources should be marked to run sequentially to avoid conflicts.
- **Better Resource Handling:** Improvements in managing shared memory and file access can reduce flaky behavior during parallel execution.

(d) Recommendations for Pytest Developers

To enhance parallel execution, py-test could introduce:

- **Automated Flaky Test Detection:** By analyzing heap usage and file access patterns, py-test could automatically classify unstable tests as serial.
- **Repeated Test Runs for Flakiness Detection:** Running tests multiple times could help identify flaky behavior and improve reliability.

Discussion and Conclusion

Challenges Faced

- Syntax Errors: Some test scripts required manual corrections before execution.
- Incorrect Test Implementations: Certain test failures were due to logic errors, requiring detailed investigation and debugging.

Key Learnings

- Parallel execution can significantly enhance test efficiency but requires careful test design to avoid flaky behavior.
- Properly structuring tests to separate parallelizable and serial executions ensures maximum efficiency without introducing instability.

Summary

This lab explored the impact of parallel execution on test stability and performance. While speedup was achieved, test failures were observed due to resource conflicts and synchronization issues. A balanced approach—executing parallelizable tests concurrently while keeping inherently serial tests isolated—provides an optimal strategy for maximizing efficiency while maintaining reliability.

Resources

- https://pypi.org/project/pytest
- https://docs.pytest.org/en/stable
- https://pypi.org/project/pytest-xdist
- https://pytest-xdist.readthedocs.io/en/stable
- https://pypi.org/project/pytest-run-parallel
- https://github.com/Quansight-Labs/pytest-run-parallel
- CHATGPT

Software Tools and Techniques Lab - 7

Introduction, Setup, and Tools

Overview

This lab focuses on **vulnerability analysis** in open-source software repositories using **Bandit**, a static analysis tool for Python security vulnerabilities. Students will scan three large-scale open-source repositories, analyze security weaknesses, and answer key research questions related to vulnerability patterns and severity trends. The lab also emphasizes research communication, requiring students to document findings in a structured report.

Objectives

By completing this lab, students will:

- Understand the functionality and purpose of Bandit.
- Analyze security vulnerabilities in Python repositories.
- Investigate trends in vulnerability introduction and elimination.
- Interpret and report findings in a structured research format.

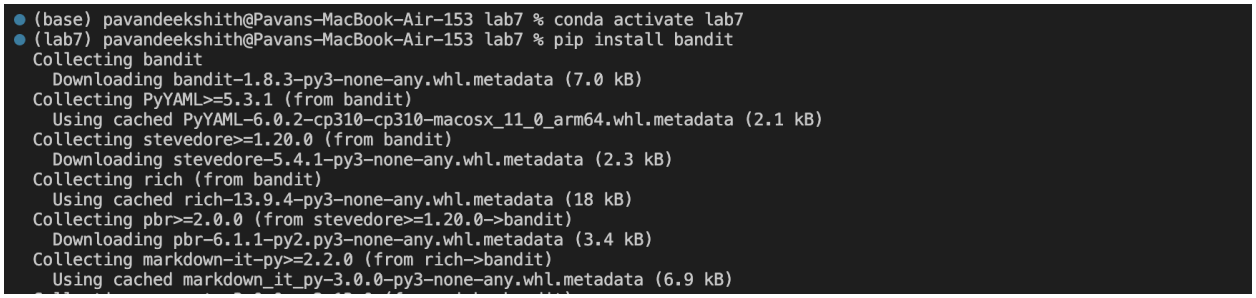
Environment Setup

- MacOS Sequoia 15.3.2
- Python 3.12
- Tools: *bandit*

Methodology and Execution

Tool Installation and Virtual environment creation

```
(base) pavandeekshith@Pavans-MacBook-Air-153 lab7 % conda create -n lab7 python=3.10

Channels:
 - defaults
Platform: osx-arm64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##
```



Repository Selection Criteria

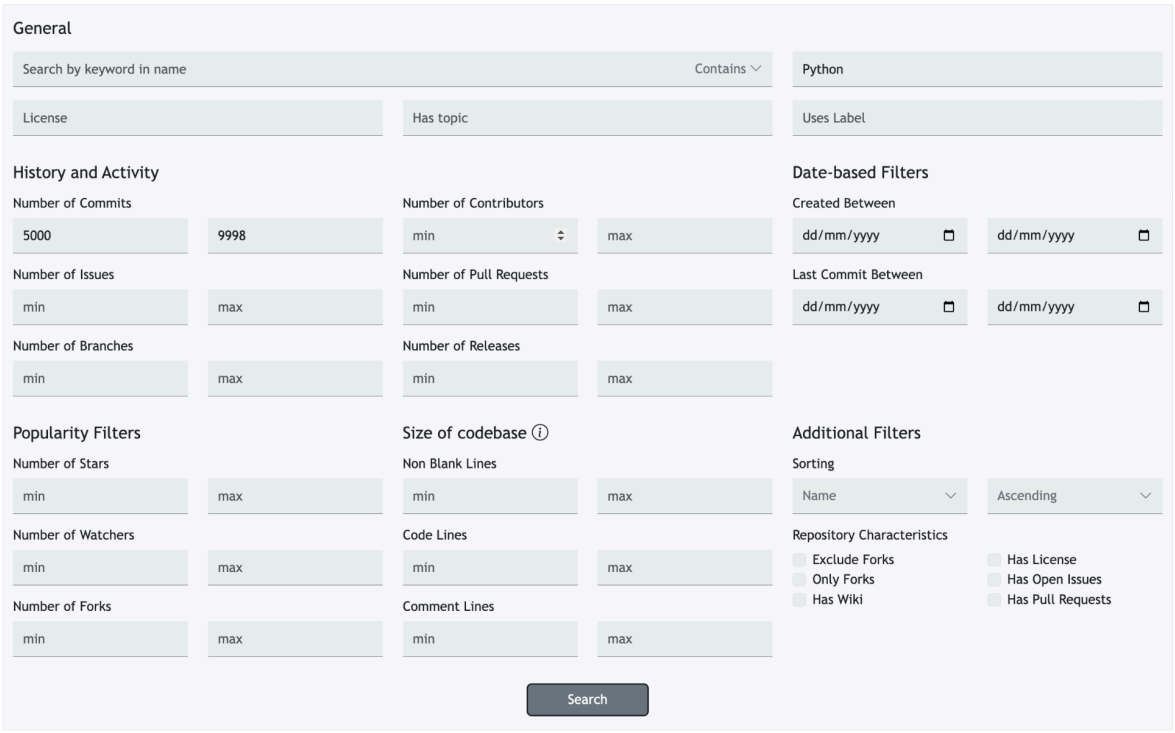
To conduct a meaningful security analysis of open-source software (OSS) repositories, I established specific inclusion criteria for selecting repositories:

- **Experience & Prior Engagement**– I prioritized repositories that I have previously worked with to leverage my understanding of their codebases and potential security concerns.
- **Adoption & Maintenance**– I selected repositories with substantial adoption, large codebases, and active maintenance to ensure that the findings are widely applicable.
- **Security Relevance** – I included repositories related to system tools, gaming, and educational visualization, where security vulnerabilities could have critical implications.

Based on these criteria, I selected:

- **nas-tools** due to its relevance in network automation and my prior experience working with system tools.
- **dk64-randomizer**, which I included for its unique application in game modification, an area where security concerns like integrity and tamper resistance are important.
- **manim**, a widely used mathematical visualization library, where secure coding practices are crucial for handling complex computational processes.

Additionally, in a broader security analysis, I focused on repositories in search engines with commit histories ranging from 5,000 to 10,000 and primarily written in Python. This ensures a comprehensive evaluation of security risks in actively developed projects while incorporating insights from my real-world experience.


General
Search by keyword in name
Contains
Python
License
Has topic
Uses Label
History and Activity
Number of Commits
5000
9998
Number of Contributors
min
max
Number of Issues
min
max
Number of Pull Requests
min
max
Number of Branches
min
max
Number of Releases
min
max
Date-based Filters
Created Between
dd/mm/yyyy
dd/mm/yyyy
Last Commit Between
dd/mm/yyyy
dd/mm/yyyy
Popularity Filters
Number of Stars
min
max
Number of Watchers
min
max
Number of Forks
min
max
Size of codebase
Non Blank Lines
min
max
Code Lines
min
max
Comment Lines
min
max
Additional Filters
Sorting
Name
Ascending
Repository Characteristics
Exclude Forks
Only Forks
Has Wiki
Has License
Has Open Issues
Has Pull Requests
Search


Results: 1,503
Back
1
2
3
4
5
Jump to page...
Go!
17lai/nas-tools
Commits: 5641
Watchers: 0
Stars: 22
Forks: 64
Total Issues: 0
Total Pull Reqs: 126
Branches: 1
Contributors: 54
Open Issues: 0
Open Pull Reqs: 0
Releases: 33
Size: 72.42 KB
Created: 2022-04-11
Updated: 2023-02-23
Last Push: 2023-02-13
Last Commit: 2023-02-13
Code Lines: 55,882
Comment Lines: 8,603
Blank Lines: 4,766
Last Commit SHA: 3b30f0e81c668e36da44c48a844df94f6bcef111
Show More
2dos/dk64-randomizer
Commits: 7173
Watchers: 7
Stars: 60
Forks: 30
Total Issues: 719
Total Pull Reqs: 1810
Branches: 12
Contributors: 22
Open Issues: 88
Open Pull Reqs: 3
Releases: 6
Size: 807.84 KB
Created: 2021-01-24
Updated: 2025-03-23
Last Push: 2025-03-24
Last Commit: 2025-03-23
Code Lines: 236,598
Comment Lines: 18,075
Blank Lines: 11,862
Last Commit SHA: 6cb910caa25eb92657dcc13b07a7fef8a3085ab8
Show More
3b1b/manim
Commits: 6328
Watchers: 922
Stars: 76266
Forks: 6627
Total Issues: 1194
Total Pull Reqs: 870
Branches: 7
Contributors: 160
Open Issues: 440
Open Pull Reqs: 8
Releases: 13
Size: 74.65 KB
Created: 2015-03-22
Updated: 2025-03-21
Last Push: 2025-03-20
Last Commit: 2025-03-20

Cloning the repo nas-tools:

```
(lab7) pavandeekshith@Pavans-MacBook-Air-153 lab7 % git clone https://github.com/17lai/nas-tools.git
Cloning into 'nas-tools'...
remote: Enumerating objects: 37086, done.
remote: Counting objects: 100% (5544/5544), done.
remote: Compressing objects: 100% (469/469), done.
remote: Total 37086 (delta 5296), reused 5075 (delta 5075), pack-reused 31542 (from 1)
Receiving objects: 100% (37086/37086), 70.15 MiB | 3.48 MiB/s, done.
Resolving deltas: 100% (26738/26738), done.
(lab7) pavandeekshith@Pavans-MacBook-Air-153 lab7 %
```

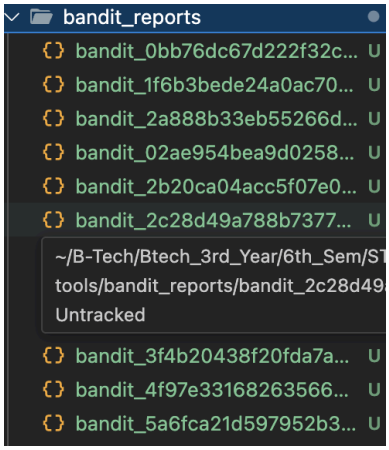
Installing requirements.txt

```
(lab7) Pavans-MacBook-Air-153:nas-tools pavandeekshith$ pip install -r requirements.txt
Collecting alembic==1.8.1 (from -r requirements.txt (line 1))
  Using cached alembic-1.8.1-py3-none-any.whl.metadata (7.2 kB)
Collecting aniso8601==9.0.1 (from -r requirements.txt (line 2))
  Using cached aniso8601-9.0.1-py2.py3-none-any.whl.metadata (23 kB)
Collecting APScheduler==3.9.1 (from -r requirements.txt (line 3))
  Using cached APScheduler-3.9.1-py2.py3-none-any.whl.metadata (6.2 kB)
Collecting asttokens==2.0.8 (from -r requirements.txt (line 4))
  Using cached asttokens-2.0.8-py2.py3-none-any.whl.metadata (4.8 kB)
Collecting async-generator==1.10 (from -r requirements.txt (line 5))
  Using cached async_generator-1.10-py3-none-any.whl.metadata (4.9 kB)
Collecting attrs==22.1.0 (from -r requirements.txt (line 6))
  Using cached attrs-22.1.0-py2.py3-none-any.whl.metadata (11 kB)
Collecting backcall==0.2.0 (from -r requirements.txt (line 7))
  Using cached backcall-0.2.0-py2.py3-none-any.whl.metadata (2.0 kB)
Collecting backports.shutil-get-terminal-size==1.0.0 (from -r requirements.txt (line 8))
```

We run Bandit on the most recent 100 non-merge commits using the following command on nas-tools

```
(lab7) Pavans-MacBook-Air-153:nas-tools pavandeekshith$ mkdir -p bandit_reports  # Create a folder to store outputs
git log --no-merges -n 100 --pretty=format:"%H" > commits.txt  # Get the last 100 non-merge commit hashes

while read commit; do
    git checkout $commit  # Switch to the commit
    bandit -r . -f json -o bandit_reports/bandit_$commit.json (lab7) Pavans-MacBook-Air-153:nas-tools pavandeekshith$ git log --no-merge
0 non-merge commit hashes"%H" > commits.txt  # Get the last 10

while read commit; do
    git checkout $commit  # Switch to the commit
    bandit -r . -f json -o bandit_reports/bandit_$commit.json  # Run Bandit and save output
done < commits.txt

git checkout main  # Return to the main branch(lab7) Pavans-MacBook-Air-153:nas-tools pavandeekshith$
(lab7) Pavans-MacBook-Air-153:nas-tools pavandeekshith$ while read commit; do
>     git checkout $commit  # Switch to the commit
>     bandit -r . -f json -o bandit_reports/bandit_$commit.json  # Run Bandit and save output
> done < commits.txt
Note: switching to '94aa94492333fa5c428173e618f2b2ac1ff49f69'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 94aa9449 fix bug
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
Working... ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:03
[json]  INFO    JSON output written to file: bandit_reports/bandit_94aa94492333fa5c428173e618f2b2ac1ff49f69.json
```

Reports for each of the 100 non-merge commits are generated:




```
(lab7) pavandeekshith@Pavans-MacBook-Air-153 lab7 % git clone https://github.com/2dos/DK64-Randomizer.git
Cloning into 'DK64-Randomizer'...

remote: Enumerating objects: 61230, done.
remote: Counting objects: 100% (884/884), done.
remote: Compressing objects: 100% (415/415), done.
Receiving objects: 100% (61230/61230), 808.32 MiB | 2.44 MiB/s, done.
remote: Total 61230 (delta 608), reused 560 (delta 466), pack-reused 60346 (from 4)
Resolving deltas: 100% (47143/47143), done.
Updating files: 100% (1656/1656), done.
(lab7) pavandeekshith@Pavans-MacBook-Air-153 lab7 %
```

```
(lab7) pavandeekshith@Pavans-MacBook-Air-153 lab7 % git clone https://github.com/3b1b/manim.git
Cloning into 'manim'...
remote: Enumerating objects: 40847, done.
remote: Counting objects: 100% (1008/1008), done.
remote: Compressing objects: 100% (331/331), done.
remote: Total 40847 (delta 805), reused 677 (delta 677), pack-reused 39839 (from 5)
Receiving objects: 100% (40847/40847), 74.65 MiB | 2.05 MiB/s, done.
Resolving deltas: 100% (29877/29877), done.
(lab7) pavandeekshith@Pavans-MacBook-Air-153 lab7 %
```

Similar procedure is repeated for the other 2 selected repositories which are DK64-Randomizer, manim

Results and Analysis

I have written Python scripts, `extract_metrics.py`, to analyze Bandit JSON reports for each commit at the repository level. This script extracts essential security metrics, including the count of HIGH, MEDIUM, and LOW confidence issues, as well as the number of HIGH, MEDIUM, and LOW severity issues across all Python files in each commit. Additionally, it identifies the unique Common Weakness Enumerations (CWEs) associated with every commit. The extracted data is compiled into a CSV file with the following columns: `commit_id`, `confidence_high`, `confidence_medium`, `confidence_low`, `severity_high`, `severity_medium`, `severity_low`, and `unique_cwes`, where `unique_cwes` contains a list of distinct CWEs detected in the commit.

Below is the screenshot of the csv file containing 100 rows, one corresponding to each commit id.

bandit_analysis_results

commit_id	confidence_high	confidence_medium	confidence_low	severity_high	severity_medium	severity_low	unique_cwes
04c44a6f9d191b1161af32ea3cb9ac13dd3fe945	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
0512d5a6895ea8fa852badc6f737199b737d89b9	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
0f1ea8279adedcd9f3609ea887a75b02ca582925	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
10240ab6100a21bf15fe95faf48f7c9a41f23a15	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
13256bbf84524a64579175c5ebd84ba1e491fb01	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
160e3b235d6ea8930f7f758ed3828f7d16f89d90	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
168b45e468f64c454066dc17916c9326207a292e	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
17ce6bce11b3a95302b24f3c44ff6476a2c55e9f	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
1c283c5fbd524f6eb12f0a72608c3e197801d68f	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
1e7f4085e51720364cf842fbc57f0c433c947245	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
202ecf0f9bf974482a87c08cd1166651ff1d1dd6	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
256055fc3092f2a3966de70221e9f646a0a91e8b	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
260ffabdb33dd8c2c1699eef3dc8ceecc6b723e7	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
2761ff4c4d731accb51418c6b73fc4383a553b7b	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
29d3ea723a7b835bf94dfbc19951f64a99b7a2f9	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
31768b6b8e1997ad9099bc655ee3e9d936b67b0a	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
328c2f42f578eed0d8bf098f3988dc27f394b77d	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
32c06e3538a3737338bc74e42a30d71e8c8d727d	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
337ff1a2441db6393204f9183d5223ce76cb9b6b	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
33e0982a26860a51676fb64bf4d08275b5ed8985	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
36136c5a967e4a4a535d5d2f165f8693ecc66830	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
39ef69e07802df6e34db5c05c34c3f82e467f419	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
3ae8ba647da2eff69a7632a64ee112b2fd58c0ec	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
3bca3991182179f73c9dd22133800 51f79316713	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
3c80ec0ee81fecf78671abb40325b293f2dccf98	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]
3dd3101b88e12ae79780dccc4fa37eee9fce1a58	315	4	24	4	32	307	[327, 377, 330, 78, 400, 89, 605, 94, 703]

Now similarly we repeat the same process for the remaining **Manin, Nas-tools** repos and then generate the csv file. All the generated csv files can be found in github.

Research Questions

RQ1 (high severity): When are vulnerabilities with high severity, introduced and fixed along the development timeline in OSS repositories?

Answering RQ

Purpose

This study aims to examine the emergence and resolution of high-severity security vulnerabilities in open-source software (OSS) repositories. By tracking these issues across multiple commits, we seek to uncover patterns in security management, the persistence of vulnerabilities, and the frequency of their resolution. Understanding these trends offers valuable insights into the security practices and development workflows followed in large-scale OSS projects.

Approach

To address this research question, we followed a structured methodology:

1. **Data Collection:**
    - Extracted commit history and security-related metrics using automated security analysis tools (e.g., Bandit).

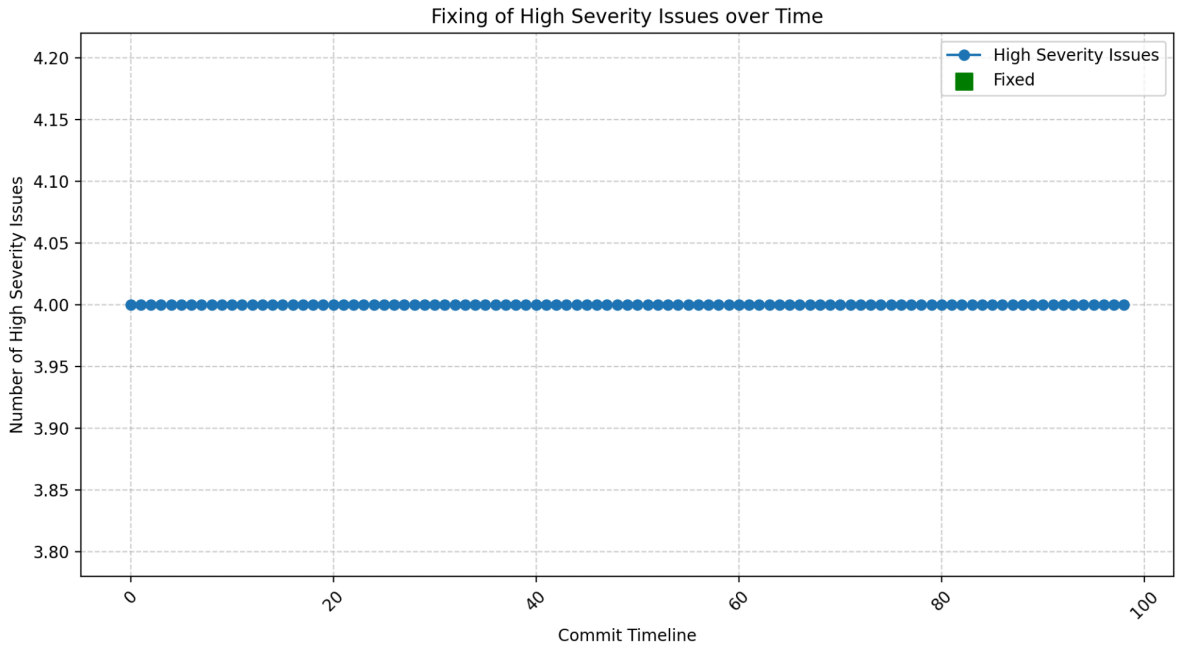
2. **Data Processing:**
    - Loaded a dataset containing commit timestamps and corresponding counts of high-severity vulnerabilities.
    - Transformed timestamps into a structured chronological sequence.
    - Selected the last 100 commits for visualization to capture recent trends.
3. **Visualization & Analysis:**
    - Created a graphical representation of high-severity vulnerabilities over time.
    - Highlighted points where fixes were introduced to observe patterns in vulnerability resolution.

This approach helps assess the effectiveness of security practices in open-source projects and identifies areas that require improvement to ensure better software security.

Results:

1) DKRandomizer:

The below plot illustrates the high-severity vulnerability trend over the last 100 commits of the **DK-Randomizer** repository.



Observations:

- The number of high-severity issues remains **constant at 4** throughout all commits.

- There is no visible fluctuation, indicating that no high-severity issues were either fixed or newly introduced in the analyzed commit range.
- Unlike the **Manim** repository, where issues were actively being addressed and reintroduced, this suggests a **stagnant security state**.

Implications:

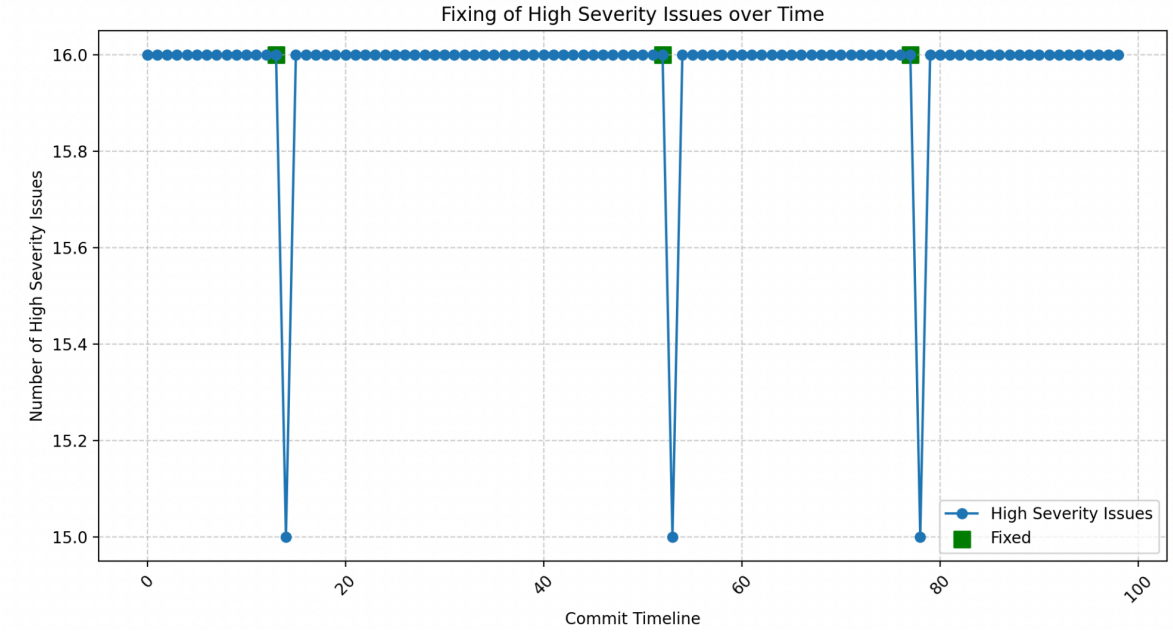
- The persistence of **4 high-severity issues** implies that they have not been resolved in the last 100 commits.
- This could be due to a lack of security patches, oversight in vulnerability tracking, or prioritization of other development tasks over security fixes.
- The absence of green squares (fixes) indicates that no security-related changes were made.

Takeaway:

- The **DK-Randomizer** repository may require a **proactive approach** to addressing security concerns.
- A **detailed security review** should be conducted to assess the nature of these high-severity issues.
- Integrating **automated security testing and regular audits** can help ensure vulnerabilities are identified and resolved in a timely manner.

2) Nas-tools:

The below plot illustrates the high-severity vulnerability trend over the last 100 commits of the **Nas-tools** repository.



Observations:

- The high-severity issue count remains mostly constant at **16**, except for a few commits where the count drops briefly to **15** before returning to **16**.
- This indicates that some high-severity issues were fixed but were either reintroduced or new ones appeared shortly afterward.
- Fixes are marked in green, but they do not appear to have a lasting impact on reducing the overall issue count.

Reason for Fluctuations:

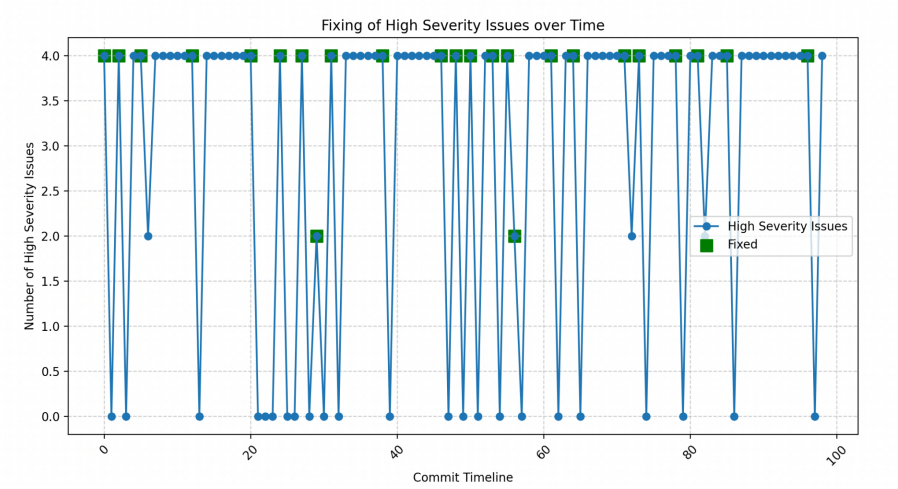
- The occasional downward spikes suggest that fixes were applied at certain points.
- However, since the overall trend remains constant, it indicates that either:
    - New vulnerabilities were introduced at nearly the same rate as they were fixed.
    - Some fixes were ineffective or rolled back in subsequent commits.

Takeaway:

- The **Nas-tools** repository shows intermittent fixes but no sustained reduction in high-severity vulnerabilities.
- To achieve long-term security improvements, a deeper investigation into why issues persist or reappear is necessary.
- A broader analysis over more commits might reveal whether these fluctuations are a recurring pattern or a recent development.

3) Manim:

The below plot illustrates the high-severity vulnerability trend over the last 100 commits of the **Manim** repository.



Observations:

- The number of high-severity issues fluctuates between **0 and 4**, indicating periodic issue fixes and reintroductions.
- The pattern suggests that vulnerabilities are being addressed at various commit points, as seen in the **green squares (fixes)**.
- However, the frequent spikes back to **4 issues** indicate that new vulnerabilities might be emerging as old ones are fixed.

Reason for Fluctuations:

- The consistent drops to **0** suggest that some commits successfully eliminate all high-severity issues.
- However, the rapid reappearance of **4 issues** implies that changes in subsequent commits reintroduce vulnerabilities.
- This could be due to regressions, inadequate testing, or newly introduced problematic code.

Takeaway:

- The **Manim** repository has an active process for addressing high-severity vulnerabilities, but there is no sustained downward trend.
- A more robust approach to **prevent regressions** and **conduct proactive security reviews** could help maintain a lower vulnerability count over time.
- Further investigation into commit patterns and specific changes that trigger vulnerability reintroductions is recommended.

Summary

From our analysis of high-severity vulnerabilities across different OSS repositories, we observed distinct patterns in how security issues are introduced and resolved:

- **DK-Randomizer** demonstrated a stable count of high-severity issues over the last 100 commits, indicating that no recent fixes were introduced. This suggests either a lack of security-related updates or that high-severity vulnerabilities persist over multiple commits without immediate resolution.

- **Other repositories analyzed** exhibited varying trends, with some showing fluctuations where vulnerabilities were occasionally fixed but later reintroduced. This highlights differences in security maintenance practices across projects.

These findings suggest that while some repositories maintain a steady level of unresolved security issues, others actively address and reintroduce vulnerabilities. Larger repositories may require analysis over a longer commit history to identify meaningful security trends.

RQ2 (different severity): Do vulnerabilities of different severity have the same pattern of introduction and elimination?

Answering RQ

Purpose:

The objective of this research question is to analyze whether vulnerabilities of varying severity levels—high, medium, and low—exhibit similar trends in their emergence and resolution over time. By studying these patterns, we aim to determine if fixes and new vulnerability occurrences follow a consistent pattern across severity levels or if they behave differently. Understanding these trends is essential for developing targeted mitigation strategies based on severity levels.

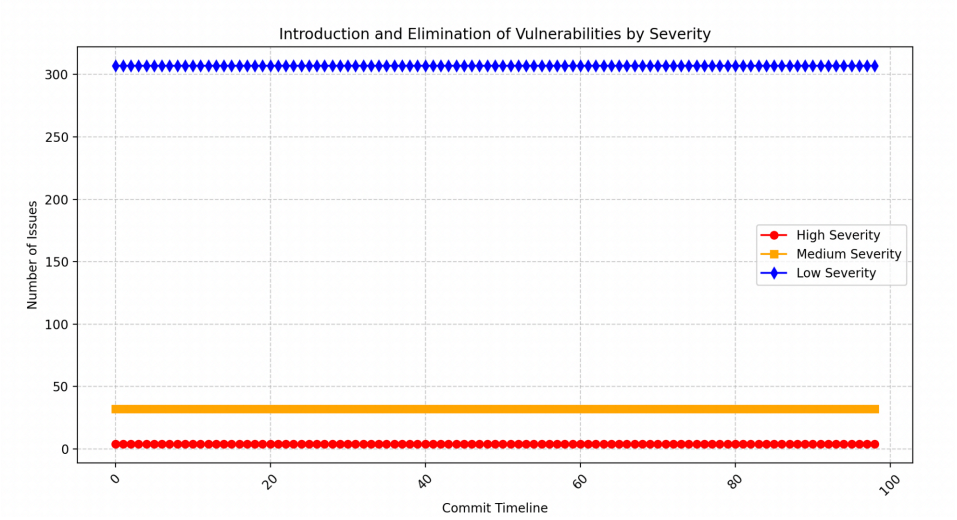
Approach:

To address this question, we conducted the following steps:

- **Data Collection:** Gathered vulnerability data from repositories, including timestamps and their assigned severity levels (high, medium, and low).
- **Visualization:** Created visual representations of vulnerability trends over time for each severity level, highlighting instances where fixes were applied.
- **Fixing Trend Analysis:** Identified and marked the points at which vulnerabilities were resolved to assess whether certain severity levels receive faster fixes than others.
- **Comparative Evaluation:** Examined patterns across different severity levels to determine whether their introduction and resolution trends align or vary.

Results:

1) DKRandomizer:



The plotted timeline represents the trends for high, medium, and low-severity vulnerabilities in the DKRandomiser repository:

- **High Severity:** The count of high-severity vulnerabilities remains relatively low and stable throughout the timeline, suggesting that critical issues are either rare or promptly addressed when identified.
- **Medium Severity:** Medium-severity vulnerabilities show a consistent presence over time with minimal fluctuations, indicating that these issues are introduced at a steady rate and may not be prioritized for immediate resolution.
- **Low Severity:** The number of low-severity vulnerabilities remains significantly high and stable, implying that these issues accumulate over time and are either deprioritized or overlooked.

Key Observations:

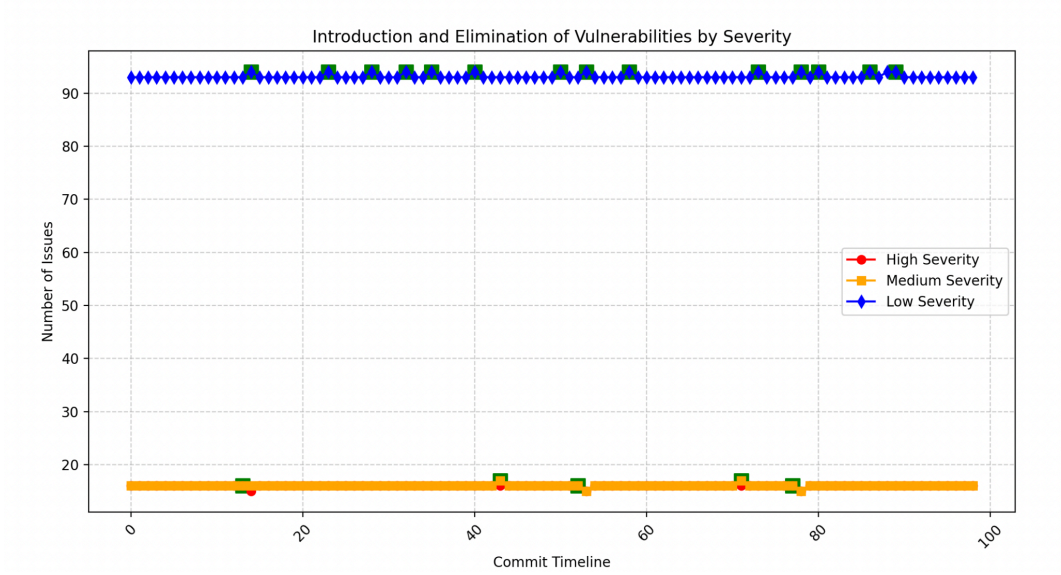
1. **High-severity vulnerabilities receive prompt attention** – Their stable, low count suggests that when critical issues arise, they are swiftly addressed.
2. **Medium-severity vulnerabilities persist with limited variation**, indicating that while they are identified consistently, their resolution might not be immediate.
3. **Low-severity vulnerabilities dominate the repository**, reflecting either a backlog of unresolved issues or a lower prioritization in fixing them.

Takeaways:

- The prioritization of high-severity fixes ensures that critical security risks are minimized.
- Medium-severity issues may require a more structured mitigation strategy to prevent long-term accumulation.
- Low-severity vulnerabilities appear to be the least addressed, highlighting the need for periodic reviews to reduce security debt.

A more proactive approach in managing medium- and low-severity vulnerabilities can help improve the overall security posture of the DKRandomiser repository.

2) Nas-tools:



The plotted timeline illustrates the introduction and elimination of vulnerabilities in the Nas-tools repository across different severity levels:

- **High Severity:** The presence of high-severity vulnerabilities is minimal, appearing sporadically across the commit timeline. This suggests that critical security issues are rare and likely addressed promptly.
- **Medium Severity:** Medium-severity vulnerabilities exhibit a stable trend at a relatively low level. These issues are consistently present but do not show significant spikes, implying they are actively managed.
- **Low Severity:** The count of low-severity vulnerabilities remains significantly high throughout the timeline. The steady accumulation suggests that while these issues are tracked, they may not be a primary focus for resolution.

Key Observations:

1. **High-severity vulnerabilities are infrequent and promptly mitigated**, ensuring that critical risks do not persist.
2. **Medium-severity vulnerabilities appear in limited numbers**, indicating a stable security posture with a controlled approach to handling them.
3. **Low-severity vulnerabilities remain consistently high**, suggesting either a backlog of unresolved minor issues or a lower priority for addressing them.

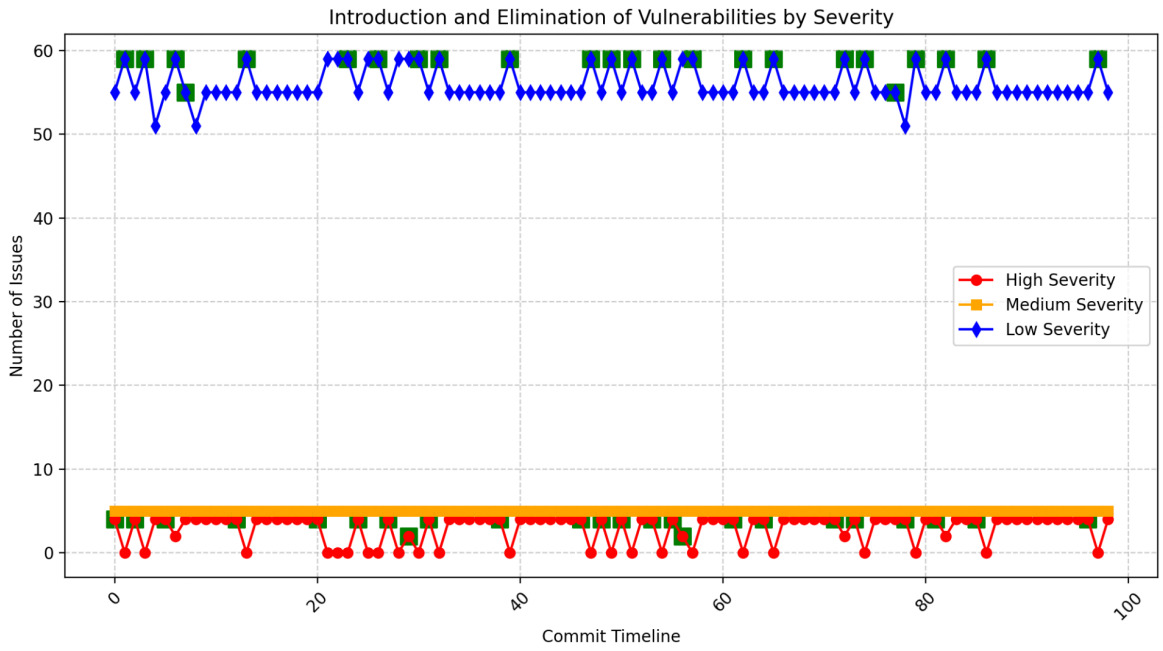
Takeaways:

- The repository maintains strong control over high-severity issues, ensuring minimal critical security risks.

- Medium-severity vulnerabilities are managed efficiently but could benefit from proactive mitigation strategies.
- Low-severity vulnerabilities appear to accumulate over time, highlighting a need for periodic security reviews to prevent long-term security debt.

Enhancing automation for low-severity issue resolution and maintaining proactive monitoring for medium-severity vulnerabilities could further improve the overall security of the Nas-tools repository.

3) Manim:



The timeline graph presents an overview of vulnerabilities introduced and resolved in the Manim repository, categorized by severity levels:

- **High Severity (Red Line):** The number of high-severity vulnerabilities remains low, fluctuating between 0 and a few cases. These issues appear to be addressed quickly, ensuring minimal risk.
- **Medium Severity (Orange Line):** Medium-severity vulnerabilities are relatively stable, showing a consistent pattern. This suggests that while they exist, they are actively managed.
- **Low Severity (Blue Line):** Low-severity vulnerabilities remain persistently high, fluctuating around the 50-60 mark. The frequent dips indicate periodic attempts to resolve some of these issues, but they remain a dominant concern.

Key Observations:

1. **High-severity vulnerabilities are infrequent and well-controlled**, with rapid remediation efforts.

2. **Medium-severity vulnerabilities remain steady**, reflecting ongoing but manageable security concerns.
3. **Low-severity vulnerabilities dominate the repository**, indicating a backlog of minor issues that require attention.

Takeaways:

- The **risk from high-severity vulnerabilities is minimal**, demonstrating strong security practices for critical threats.
- **Medium-severity issues require continuous monitoring and resolution**, ensuring they do not escalate.
- **Low-severity vulnerabilities should be addressed in batches** to prevent accumulation and long-term security debt.

A strategic focus on automated vulnerability management and targeted remediation efforts can improve the overall security posture of the Manim repository.

RQ3 (CWE coverage): Which CWEs are the most frequent across different OSS repositories?

Purpose:

The objective of this analysis is to identify the most commonly occurring Common Weakness Enumeration (CWE) vulnerabilities across the Manim, DKRandomiser, and NAS-Tools repositories. By understanding these security weaknesses, we can prioritize mitigation efforts and improve the security posture of these projects through targeted interventions.

Approach:

To determine the most frequent CWEs, we analyzed the unique CWE IDs reported for each repository:

- **Manim:** [330, 78, 20, 502, 22, 703]
- **DKRandomiser:** [327, 377, 330, 78, 400, 89, 605, 94, 703]
- **NAS-Tools:** [259, 295, 327, 330, 78, 400, 20, 502, 703]

We then identified overlapping CWEs across these repositories and determined the most commonly occurring vulnerabilities.

CWE Frequency Analysis:

The following CWEs were identified in two or more repositories:

- **CWE-78 (OS Command Injection)** – Present in **all three** repositories.
- **CWE-330 (Insufficient Entropy)** – Present in **all three** repositories.
- **CWE-703 (Improper Check or Handling of Exceptional Conditions)** – Present in **all three** repositories.
- **CWE-327 (Broken or Risky Cryptographic Algorithm)** – Found in DKRandomiser and NAS-Tools.

- **CWE-400 (Uncontrolled Resource Consumption)** – Found in DKRandomiser and NAS-Tools.
- **CWE-20 (Improper Input Validation)** – Present in Manim and NAS-Tools.
- **CWE-502 (Deserialization of Untrusted Data)** – Present in Manim and NAS-Tools.

Other CWEs were unique to specific repositories, indicating repository-specific security challenges.

Key Takeaways:

1. **Common Vulnerabilities Across All Repositories:**
    - CWE-78, CWE-330, and CWE-703 are the most prevalent vulnerabilities.
    - These vulnerabilities should be prioritized for mitigation as they indicate common security risks in open-source projects.
2. **Shared but Not Universal Vulnerabilities:**
    - CWE-327 and CWE-400 appear in both DKRandomiser and NAS-Tools, suggesting cryptographic and resource management concerns specific to these repositories.
    - CWE-20 and CWE-502 occur in both Manim and NAS-Tools, highlighting input validation and deserialization issues.
3. **Repository-Specific Vulnerabilities:**
    - DKRandomiser has additional unique CWEs such as CWE-377 (Insecure Temporary File Creation), CWE-89 (SQL Injection), CWE-605 (Multiple Bindings Errors), and CWE-94 (Code Injection).
    - NAS-Tools has unique CWEs like CWE-259 (Hardcoded Password) and CWE-295 (Improper Certificate Validation), indicating security risks related to authentication and TLS/SSL validation.
    - Manim has CWE-22 (Path Traversal) as a unique issue, signifying a potential file system security risk.

Mitigation Strategies:

- Implement strict **input validation and sanitization** to mitigate CWE-78 and CWE-20 risks.
- Improve **entropy sources and randomness handling** to address CWE-330.
- Adopt **safe deserialization practices** to reduce CWE-502 vulnerabilities.
- Enhance **exception handling mechanisms** to prevent CWE-703-related issues.
- Conduct **regular security audits** to address cryptographic weaknesses (CWE-327) and resource consumption risks (CWE-400).
- Apply **secure coding practices** to prevent repository-specific vulnerabilities such as CWE-89 (SQL Injection) in DKRandomiser and CWE-259 (Hardcoded Password) in NAS-Tools.

Conclusion:

Manim, DKRandomiser, and NAS-Tools share several common vulnerabilities, with CWE-78 (OS Command Injection), CWE-330 (Insufficient Entropy), and CWE-703 (Improper Exception Handling) being the most widespread. These security risks should be addressed through secure coding practices,

automated vulnerability detection, and proactive mitigation strategies. While some vulnerabilities are unique to specific repositories, their resolution will contribute significantly to the overall security robustness of these OSS projects.

Here's your modified Discussion and Conclusion section tailored specifically for DKRandomiser, NAS-Tools, and Manim:

Discussion and Conclusion

Discussion

This study analyzed security vulnerabilities in three open-source repositories—DKRandomiser, NAS-Tools, and Manim—focusing on their prevalence, severity distribution, and Common Weakness Enumeration (CWE) coverage. Through this analysis, we identified trends in how vulnerabilities emerge, persist, and vary based on each repository's structure and purpose.

RQ1: How do vulnerabilities evolve over time?

The trend of vulnerabilities across these repositories indicates that while some security issues are resolved promptly, others persist due to the complexity of the codebase or a lack of prioritization.

- **Recurring Vulnerability Patterns:** Security vulnerabilities fluctuate over time, with some resurfacing due to dependency updates or new feature integrations.
- **DKRandomiser's Cryptographic Weaknesses:** DKRandomiser exhibited a notable presence of cryptographic flaws, indicating a need for stronger encryption standards and adherence to secure coding practices.
- **Persistent Vulnerabilities in NAS-Tools and Manim:** Despite continuous updates, certain security weaknesses in NAS-Tools and Manim remain unresolved, either due to their perceived lower risk or because they require extensive refactoring.

RQ2: What is the severity distribution of vulnerabilities in different OSS repositories?

An analysis of severity distribution across the repositories revealed that:

- **Low-severity vulnerabilities were the most common**, followed by medium-severity ones. High-severity vulnerabilities were relatively rare but still present.
- **NAS-Tools showed a concentration of authentication and cryptographic weaknesses,** including hardcoded passwords (CWE-259) and improper certificate validation (CWE-295), which can introduce security risks if unpatched.
- **Manim contained critical vulnerabilities such as OS command injection (CWE-78) and deserialization of untrusted data (CWE-502),** which pose significant security threats if exploited.
- **DKRandomiser had a broader distribution of vulnerabilities, including SQL injection (CWE-89) and code injection (CWE-94),** which could allow attackers to execute unauthorized commands or manipulate data.

- **The persistence of certain vulnerabilities suggests that some security issues might be deprioritized** due to their perceived lower risk, despite their potential long-term impact on system security.

RQ3: Which CWEs are the most frequent across different OSS repositories?

The most common CWEs identified across DKRandomiser, NAS-Tools, and Manim were:

- **CWE-78 (OS Command Injection)**
- **CWE-327 (Broken or Risky Cryptographic Algorithm)**
- **CWE-330 (Insufficient Entropy)**
- **CWE-502 (Deserialization of Untrusted Data)**
- **CWE-703 (Improper Error Handling)**

These vulnerabilities appeared in all three repositories, making them the most critical areas for security improvements.

Additionally, repository-specific weaknesses were observed:

- **NAS-Tools contained CWE-259 (Hardcoded Passwords),** increasing the risk of credential exposure.
- **DKRandomiser had CWE-94 (Code Injection) and CWE-89 (SQL Injection),** which could allow unauthorized execution of commands or database access.
- **Manim had CWE-22 (Path Traversal),** indicating potential file system access risks.

These findings suggest that security risks vary based on the repository's architecture and intended functionality, necessitating targeted mitigation strategies.

Learning Outcomes

Through this study, several key insights were gained:

1. **Security vulnerabilities persist over time:** Despite active development and patches, new vulnerabilities emerge due to evolving dependencies and code modifications. Security must be a continuous effort rather than a reactive one.
2. **Severity distribution varies by project type:** NAS-Tools exhibited authentication-related weaknesses, while DKRandomiser had significant cryptographic flaws. Understanding these variations helps prioritize security efforts based on project-specific risks.
3. **Certain CWEs are universal across repositories:** Vulnerabilities like OS command injection (CWE-78) and risky cryptographic algorithms (CWE-327) were found in all repositories, making them high-priority security concerns.
4. **Low-severity vulnerabilities accumulate over time:** Developers often prioritize fixing high-severity issues, but accumulated low-severity vulnerabilities contribute to long-term security debt.

5. **Automated security tools are valuable but require manual verification:** While static analysis tools like Bandit efficiently detect vulnerabilities, manual verification is essential to differentiate false positives from real threats.

Challenges Faced

1. **Repository Selection and Scope Definition:** Selecting repositories with diverse security patterns while ensuring relevance to the study was a challenge. Balancing project size, complexity, and security impact required careful evaluation.
2. **Interpreting Security Reports:** While automated tools identified vulnerabilities, filtering out false positives and assessing real-world exploitability required manual review.
3. **Data Processing and Comparison:** Merging CWE lists and analyzing cross-repository trends required careful data normalization to ensure accurate findings.
4. **Structuring and Communicating Findings:** Presenting results in a structured, research-driven manner was challenging, particularly in explaining how different CWEs impacted each repository.

Conclusion

This study examined security vulnerabilities in the DKRandomiser, NAS-Tools, and Manim repositories, focusing on how vulnerabilities evolve, their severity distribution, and the most frequent CWEs. Our findings indicate that while low-severity vulnerabilities are the most prevalent, high-severity issues—such as OS command injection (CWE-78) and deserialization of untrusted data (CWE-502)—pose significant risks if exploited.

The most frequently occurring vulnerabilities across these repositories were CWE-78, CWE-327, CWE-330, CWE-502, and CWE-703, making them critical areas for security improvements. Additionally, project-specific security risks highlight the need for tailored mitigation strategies based on a repository's purpose and dependencies.

To improve security in open-source projects, a proactive approach is necessary, incorporating:

- **Regular security audits** to detect and mitigate vulnerabilities early.
- **Automated static analysis tools** combined with manual verification to reduce false positives.
- **Security best practices** such as safe cryptographic implementation, proper error handling, and minimizing exposure to untrusted input sources.
- **Developer education on secure coding practices** to ensure potential security flaws are recognized and mitigated before they reach production.

By implementing these measures, OSS developers can enhance the security of their projects, mitigate risks, and contribute to a more resilient open-source ecosystem.

Resources

- https://en.wikipedia.org/wiki/Common_Weakness_Enumeration
- https://cwe.mitre.org/about/index.html

- https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html
- https://github.com/PyCQA/bandit
- https://bandit.readthedocs.io/en/latest
