CS202 SOFTWARE TOOLS AND TECHNIQUES
LAB-5 : Code Coverage Analysis and Test Generation

Lab Topic: Code Coverage Analysis and Test Generation

Introduction:

This lab focuses on analyzing and improving code coverage for Python programs using automated testing tools. Code coverage metrics such as line, branch, and function coverage help assess how well a test suite exercises the code. We will gain insights into test effectiveness, automated test generation, and coverage analysis, which are crucial for software quality assurance.

SETUP:

After opening the set-iitgn-vm, I created a folder lab5 to do all the lab activities in that directory.

```
set-iitgn-vm@set-iitgn-vm:~$ cd lab5
set-iitgn-vm@set-iitgn-vm:~/lab5$
```

TOOLS:

- pytest (for running tests)
- pytest-cov (for line/branch coverage analysis)
- pytest-func-cov (for function coverage analysis)
- coverage (for detailed coverage metrics)
- pynguin (for automated unit test generation)1
- Some other tools (genhtml, lcov) introduced in Lecture 5

METHODOLOGY AND EXECUTION:

1. To clone the given github repository I used the command:
   git clone https://github.com/keon/algorithms.git
   Then navigate into the folder using *cd algorithms*

```
set-iitgn-vm@set-iitgn-vm:~/lab5$ git clone https://github.com/keon/algorithms.g
it
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 | 1.10 MiB/s, done.
Resolving deltas: 100% (3241/3241), done.
set-iitgn-vm@set-iitgn-vm:~/lab5$ cd algorithms
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$
```

   To get the current commit hash in the repository, I used the command: *git rev-parse HEAD.*
   It gave the commit: cad4754bc71742c2d6fcbd3b92ae74834d359844

```
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ git rev-parse HEAD
cad4754bc71742c2d6fcbd3b92ae74834d359844
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$
```

Next we need to set up a virtual environment for python 3.10 version in the algorithms directory.
To create a virtual environment to have python3.10 in it named ‘lab5env’, I used the command: *python3.10 -m venv lab5env.*
To activate the virtual environment, I used the command: *source lab5env/bin/activate*
Then I verified python version using : *python –version* which gave python 3.10.11

```
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ python3.10 -m venv lab5env
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ source lab5env/bin/activate
```

```
(lab5env) set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ python --version
Python 3.10.11
(lab5env) set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$
```

To setup the required dependencies, I used the command: *pip install -r test_requirements.txt*

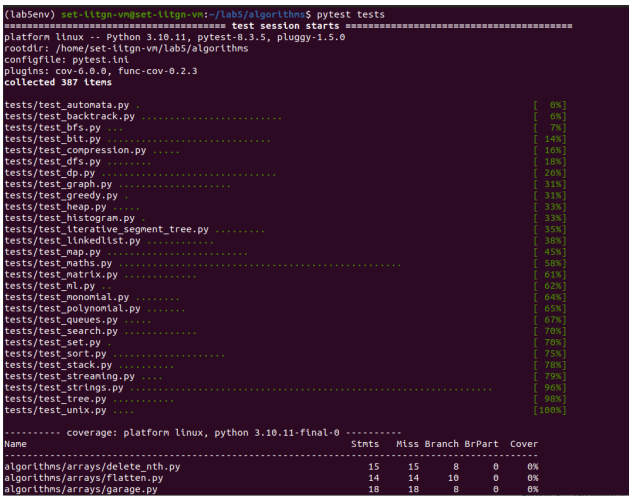
```
(lab5env) set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ python --version
Python 3.10.11
(lab5env) set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ pip install --upgrade pip
Requirement already satisfied: pip in ./lab5env/lib/python3.10/site-packages (23.0.1)
Collecting pip
  Downloading pip-25.0.1-py3-none-any.whl (1.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 8.4 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.0.1
    Uninstalling pip-23.0.1:
      Successfully uninstalled pip-23.0.1
Successfully installed pip-25.0.1
(lab5env) set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ pip install -r test_requirements.t
xt
Collecting flake8 (from -r test_requirements.txt (line 1))
  Downloading 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)
Collecting coverage (from -r test_requirements.txt (line 3))
  Downloading coverage-7.7.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux
_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.5 kB)
Collecting nose (from -r test_requirements.txt (line 4))
  Using cached nose-1.3.7-py3-none-any.whl.metadata (1.7 kB)
Collecting pytest (from -r test_requirements.txt (line 5))
  Downloading pytest-8.3.5-py3-none-any.whl.metadata (7.6 kB)
Collecting tox (from -r test_requirements.txt (line 6))
```

```
Using cached tomli-2.2.1-py3-none-any.whl (14 kB)
Using cached typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Downloading virtualenv-20.29.3-py3-none-any.whl (4.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.3/4.3 MB 9.3 MB/s eta 0:00:00
Using cached iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Using cached PyYAML-6.0.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (751
 kB)
Using cached requests-2.32.3-py3-none-any.whl (64 kB)
Using cached six-1.17.0-py2.py3-none-any.whl (11 kB)
Using cached certifi-2025.1.31-py3-none-any.whl (166 kB)
Using cached charset_normalizer-3.4.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (146 kB)
Using cached distlib-0.3.9-py2.py3-none-any.whl (468 kB)
Using cached idna-3.10-py3-none-any.whl (70 kB)
Using cached urllib3-2.3.0-py3-none-any.whl (128 kB)
Installing collected packages: nose, distlib, urllib3, typing-extensions, tomli, six, PyY
AML, pyflakes, pycodestyle, pluggy, platformdirs, pathspec, packaging, mypy-extensions, m
ccabe, iniconfig, idna, filelock, exceptiongroup, coverage, colorama, click, charset-norm
alizer, chardet, certifi, cachetools, virtualenv, requests, pytest, pyproject-api, flake8
, black, tox, python-coveralls
Successfully installed PyYAML-6.0.2 black-25.1.0 cachetools-5.5.2 certifi-2025.1.31 chard
et-5.2.0 charset-normalizer-3.4.1 click-8.1.8 colorama-0.4.6 coverage-7.7.0 distlib-0.3.9
 exceptiongroup-1.2.2 filelock-3.18.0 flake8-7.1.2 idna-3.10 iniconfig-2.0.0 mccabe-0.7.0
 mypy-extensions-1.0.0 nose-1.3.7 packaging-24.2 pathspec-0.12.1 platformdirs-4.3.6 plugg
y-1.5.0 pycodestyle-2.12.1 pyflakes-3.2.0 pyproject-api-1.9.0 pytest-8.3.5 python-coveral
ls-2.9.3 requests-2.32.3 six-1.17.0 tomli-2.2.1 tox-4.24.2 typing-extensions-4.12.2 urlli
b3-2.3.0 virtualenv-20.29.3
(lab5env) set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$
```

To check if the dependencies are installed I ran the command : *pip list* which gave:

```
(lab5env) set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ pip list
Package            Version
------------------ ---------
black              25.1.0
cachetools         5.5.2
certifi            2025.1.31
chardet            5.2.0
charset-normalizer 3.4.1
click              8.1.8
colorama           0.4.6
coverage           7.7.0
distlib            0.3.9
exceptiongroup     1.2.2
filelock           3.18.0
flake8             7.1.2
idna               3.10
iniconfig          2.0.0
mccabe             0.7.0
mypy-extensions    1.0.0
nose               1.3.7
packaging          24.2
pathspec           0.12.1
pip                25.0.1
platformdirs       4.3.6
pluggy             1.5.0
pycodestyle        2.12.1
pyflakes           3.2.0
pyproject-api      1.9.0
pytest             8.3.5
python-coveralls   2.9.3
PyYAML             6.0.2
requests           2.32.3
setuptools         65.5.0
six                1.17.0
tomli              2.2.1
tox                4.24.2
typing_extensions  4.12.2
urllib3            2.3.0
virtualenv         20.29.3
(lab5env) set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$
```

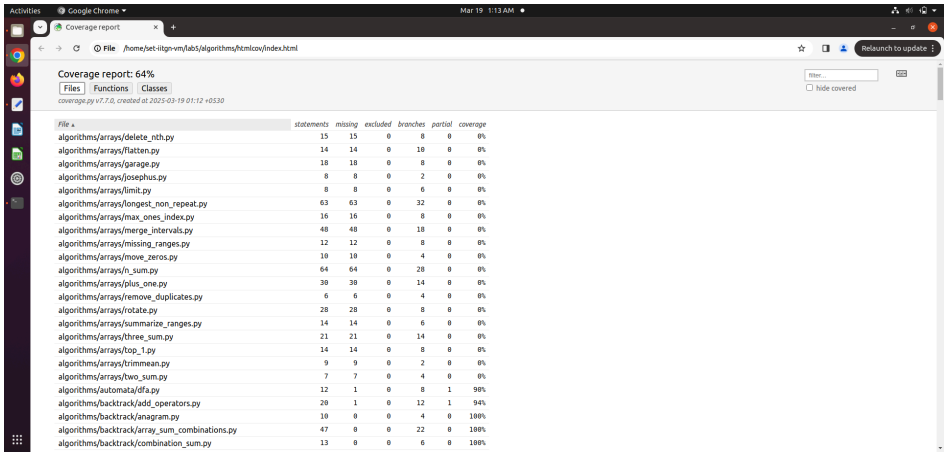
2. Configuring pytest and pytest-cov, coverage,
   First I ran the command *pip install -e .*
   It installed the other required dependencies. Then ran the command *pytest tests*
   to execute the existing tests. It gave an error in test_array.py. After fixing that I got the output:



```
algorithms/backtrack/add_operators.py                    20      1     12      1    94%
algorithms/backtrack/anagram.py                          10      0      4      0   100%
algorithms/backtrack/array_sum_combinations.py           47      0     22      0   100%
algorithms/backtrack/combination_sum.py                  13      0      6      0   100%
algorithms/backtrack/factor_combinations.py              19      0     10      0   100%
algorithms/backtrack/find_words.py                       27      0     16      0   100%
algorithms/backtrack/generate_abbreviations.py           14      0      6      0   100%
algorithms/backtrack/generate_parenthesis.py             23      0     12      0   100%
algorithms/backtrack/letter_combination.py               12      1      8      1    90%
algorithms/backtrack/palindrome_partitioning.py          20      8     16      0    56%
algorithms/backtrack/pattern_match.py                    17      1     14      2    90%
algorithms/backtrack/permute.py                          24      0     16      1    98%
algorithms/backtrack/permute_unique.py                   11      0      8      0   100%
algorithms/backtrack/subsets.py                          17      0      4      0   100%
algorithms/backtrack/subsets_unique.py                   11      0      2      0   100%
algorithms/bfs/count_islands.py                          23      0     14      0   100%
algorithms/bfs/maze_search.py                            27      1     12      1    95%
algorithms/bfs/shortest_distance_from_all_buildings.py   27     23     20      0     9%
algorithms/bfs/word_ladder.py                            32      1     24      1    96%
algorithms/bit/add_bitwise_operator.py                    6      0      2      0   100%
algorithms/bit/binary_gap.py                             28      0     12      0   100%
algorithms/bit/bit_operation.py                          14      0      0      0   100%
algorithms/bit/bytes_int_conversion.py                   26      0      8      0   100%
algorithms/bit/count_flips_to_convert.py                  7      0      2      0   100%
algorithms/bit/count_ones.py                             10      0      4      0   100%
algorithms/bit/find_difference.py                         6      0      2      0   100%
algorithms/bit/find_missing_number.py                    12      0      2      0   100%
algorithms/bit/flip_bit_longest_sequence.py              15      0      8      1    96%
algorithms/bit/has_alternative_bit.py                    16      1      6      1    91%
algorithms/bit/insert_bit.py                             13      0      0      0   100%
algorithms/bit/power_of_two.py                            2      0      0      0   100%
algorithms/bit/remove_bit.py                              5      0      0      0   100%
algorithms/bit/reverse_bits.py                            8      0      2      0   100%
algorithms/bit/single_number2.py                          6      0      2      0   100%
algorithms/bit/single_number3.py                         11      0      6      0   100%
algorithms/bit/single_number.py                           5      0      2      0   100%
algorithms/bit/subsets.py                                 9      0      2      0   100%
algorithms/bit/swap_pair.py                               5      0      0      0   100%
algorithms/compression/elias.py                          18      0      2      0   100%
algorithms/compression/huffman_coding.py                217     20     58      4    91%
algorithms/compression/rle_compression.py                22      0     12      1    97%
algorithms/dfs/all_factors.py                            33      0     16      0   100%
algorithms/dfs/count_islands.py                          18      0     10      0   100%
```

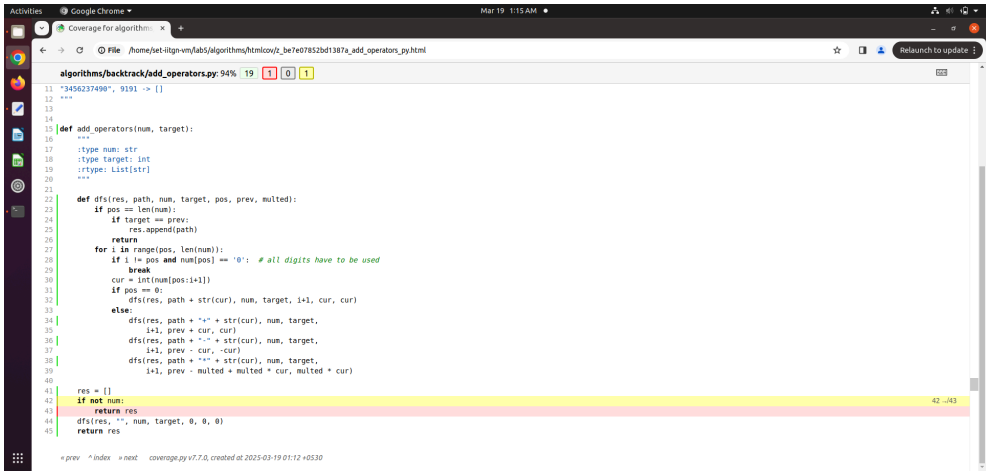
```
algorithms/strings/strong_password.py                    11      2      8      2    79%
algorithms/strings/text_justification.py                 45      1     22      1    97%
algorithms/strings/unique_morse.py                       14      0      6      0   100%
algorithms/strings/validate_coordinates.py               22      7      6      1    71%
algorithms/strings/word_squares.py                       20      0     12      0   100%
algorithms/tree/avl/avl.py                               77     77     34      0     0%
algorithms/tree/b_tree.py                               151     18     54      2    87%
algorithms/tree/bin_tree_to_list.py                      28     28     16      0     0%
algorithms/tree/binary_tree_paths.py                     13     13      8      0     0%
algorithms/tree/construct_tree_postorder_preorder.py     42      7     18      3    83%
algorithms/tree/deepest_left.py                          25     25      8      0     0%
algorithms/tree/fenwick_tree/fenwick_tree.py             21      0      6      0   100%
algorithms/tree/invert_tree.py                            8      8      6      0     0%
algorithms/tree/is_balanced.py                           12     12      4      0     0%
algorithms/tree/is_subtree.py                            19     19      8      0     0%
algorithms/tree/is_symmetric.py                          25     25     16      0     0%
algorithms/tree/longest_consecutive.py                   15     15      6      0     0%
algorithms/tree/lowest_common_ancestor.py                 8      8      4      0     0%
algorithms/tree/max_height.py                            33     33     14      0     0%
algorithms/tree/max_path_sum.py                          11     11      2      0     0%
algorithms/tree/min_height.py                            40     40     20      0     0%
algorithms/tree/path_sum2.py                             42     42     28      0     0%
algorithms/tree/path_sum.py                              35     35     28      0     0%
algorithms/tree/pretty_print.py                          10     10      6      0     0%
algorithms/tree/same_tree.py                              6      6      4      0     0%
algorithms/tree/segment_tree/iterative_segment_tree.py   25      0     10      0   100%
algorithms/tree/traversal/inorder.py                     40     16     12      2    65%
algorithms/tree/traversal/level_order.py                 17     17     10      0     0%
algorithms/tree/traversal/postorder.py                   31      4     14      1    89%
algorithms/tree/traversal/preorder.py                    28      4     12      1    88%
algorithms/tree/traversal/zigzag.py                      19     19     10      0     0%
algorithms/tree/tree.py                                   5      5      0      0     0%
algorithms/unix/path/full_path.py                         3      0      0      0   100%
algorithms/unix/path/join_with_slash.py                   6      0      0      0   100%
algorithms/unix/path/simplify_path.py                    11      1      6      1    88%
algorithms/unix/path/split.py                             7      0      0      0   100%
----------------------------------------------------------------------------------------
TOTAL                                                  7994   2839   3780    237    64%
Coverage HTML written to dir htmlcov


====================================== 387 passed in 9.75s =======================================
(lab5env) set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$
```

For html visualization I used the command: *coverage html*
The output:



File	statements	missing	excluded	branches	partial	coverage
algorithms/arrays/delete_nth.py	15	15	0	8	0	0%
algorithms/arrays/flatten.py	14	14	0	10	0	0%
algorithms/arrays/garage.py	18	18	0	8	0	0%
algorithms/arrays/josephus.py	8	8	0	2	0	0%
algorithms/arrays/limit.py	8	8	0	6	0	0%
algorithms/arrays/longest_non_repeat.py	63	63	0	32	0	0%
algorithms/arrays/max_ones_index.py	16	16	0	8	0	0%
algorithms/arrays/merge_intervals.py	48	48	0	18	0	0%
algorithms/arrays/missing_ranges.py	12	12	0	8	0	0%
algorithms/arrays/move_zeros.py	10	10	0	4	0	0%
algorithms/arrays/n_sum.py	64	64	0	28	0	0%
algorithms/arrays/plus_one.py	30	30	0	14	0	0%
algorithms/arrays/remove_duplicates.py	6	6	0	4	0	0%
algorithms/arrays/rotate.py	28	28	0	8	0	0%
algorithms/arrays/summarize_ranges.py	14	14	0	6	0	0%
algorithms/arrays/three_sum.py	21	21	0	14	0	0%
algorithms/arrays/top_1.py	14	14	0	8	0	0%
algorithms/arrays/trimmean.py	9	9	0	2	0	0%
algorithms/arrays/two_sum.py	7	7	0	4	0	0%
algorithms/automata/dfa.py	12	1	0	8	1	90%
algorithms/backtrack/add_operators.py	20	1	0	12	1	94%
algorithms/backtrack/anagram.py	10	0	0	4	0	100%
algorithms/backtrack/array_sum_combinations.py	47	0	0	22	0	100%
algorithms/backtrack/combination_sum.py	13	0	0	6	0	100%

Here the columns, statements - total statements
Missing - Lines not covered by tests
Branches - total branches
Partial - partially covered branches
Coverage - overall coverage percentage.
In the report the uncovered code can be highlighted in red and partial one in yellow.
Example:



I ran this command to also get missing lines:

```
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ pytest --cov=algorithms --cov-report=term-missing --cov-branch
============================= test session starts ==============================
platform linux -- Python 3.10.11, pytest-7.4.4, pluggy-1.3.0
rootdir: /home/set-iitgn-vm/lab5/algorithms
configfile: pytest.ini
testpaths: tests
plugins: cov-4.1.0
collected 393 items

tests/test_automata.py .                                                 [  0%]
tests/test_backtrack.py ......................                           [  6%]
tests/test_bfs.py ...                                                    [  7%]
tests/test_bit.py ..........................                             [ 14%]
tests/test_compression.py .....                                          [ 16%]
tests/test_dfs.py ........                                               [ 18%]
tests/test_dp.py ............................                            [ 25%]
tests/test_graph.py ....................                                 [ 31%]
tests/test_greedy.py .                                                   [ 31%]
tests/test_heap.py .....                                                 [ 32%]
tests/test_histogram.py .                                                [ 32%]
tests/test_iterative_segment_tree.py .........                           [ 35%]
tests/test_linkedlist.py ............                                    [ 38%]
tests/test_map.py ..........................                             [ 44%]
```

```
---------- coverage: platform linux, python 3.10.11-final-0 ----------
Name                                                                                                Stmts   Miss Branch BrPart  Cover   Missing
-----------------------------------------------------------------------------------------------------------------------------------------
algorithms/arrays/delete_nth.py                                                                        15     12      8      0    13%   14-18, 23-32
algorithms/arrays/flatten.py                                                                           14     11     10      0    12%   11-18, 27-31
algorithms/arrays/garage.py                                                                            18     16      8      0     8%   37-54
algorithms/arrays/josephus.py                                                                           8      7      2      0    10%   13-19
algorithms/arrays/limit.py                                                                              8      7      8      0     6%   17-25
algorithms/arrays/longest_non_repeat.py                                                                63     58     32      0     5%   19-29, 37-47, 56-69, 78-91, 10
0-109
algorithms/arrays/max_ones_index.py                                                                    16     15      8      0     4%   21-43
algorithms/arrays/merge_intervals.py                                                                   48     34     21      0    20%   15-16, 19, 22, 25-27, 30, 33-3
5, 38-40, 44, 49-55, 60-63, 68-77
algorithms/arrays/missing_ranges.py                                                                    12     11      8      0     5%   8-22
algorithms/arrays/move_zeros.py                                                                        10      0      4      0   100%
algorithms/arrays/n_sum.py                                                                             64     63     28      0     1%   52-140
algorithms/arrays/plus_one.py                                                                          30     27     14      0     7%   15-28, 32-39, 44-48
algorithms/arrays/remove_duplicates.py                                                                  6      5      4      0    10%   12-18
algorithms/arrays/rotate.py                                                                            28     25      8      0     8%   22-29, 40-53, 57-61
algorithms/arrays/summarize_ranges.py                                                                  14     12      8      0     9%   12-24
algorithms/arrays/three_sum.py                                                                         21     20     14      0     3%   23-48
algorithms/arrays/top_1.py                                                                             14     13      8      0     5%   16-36
```

To get the coverage in lcov format,

File /home/set-iitgn-vm/lab5/algorithms/lcov-report/index.html

LCOV - code coverage report

rrent view: top level
Test: coverage.lcov
Date: 2025-03-23 15:21:42

Hit
Lines: 546
Functions:

Directory	Line Coverage	
arrays	17.6 %	75 / 425
automata	91.7 %	11 / 12
backtrack	96.3 %	289 / 300
bfs	77.5 %	86 / 111
bit	99.5 %	221 / 222
compression	92.3 %	241 / 261
dfs	91.2 %	165 / 181
distribution	100.0 %	6 / 6
dp	63.9 %	296 / 463
graph	56.2 %	453 / 806
greedy	92.3 %	12 / 13
heap	78.5 %	102 / 130
linkedlist	32.3 %	194 / 601
map	69.0 %	176 / 255
maths	74.0 %	732 / 989
matrix	70.2 %	369 / 526
ml	100.0 %	20 / 20
queues	73.9 %	130 / 176
search	95.0 %	209 / 220
set	8.7 %	9 / 103
sort	79.4 %	370 / 466
stack	83.7 %	237 / 283
streaming	100.0 %	53 / 53
strings	84.9 %	681 / 802
tree	32.3 %	172 / 532
tree/avl	0.0 %	0 / 77
tree/fenwick_tree	100.0 %	22 / 22
tree/segment_tree	100.0 %	26 / 26
tree/traversal	56.8 %	79 / 139

3. Now I generated unit tests using pynguin
   First installed it using the command: *pip install pynguin*
   I ran the command : *pynguin --project-path . --module_name algorithms.arrays.trimmean --output-path pynguin_tests/ --maximum_search_time=60*
   I ran it for several modules which didn't have 100% coverage.

```
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ pynguin --project-path . --module_name algorithms.arrays.top_1 --output-path pyngu
in_tests/ --maximum_search_time=60
[False, 1, 1, 2, 1, 3, 'a', 0, 0]
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ pynguin --project-path . --module_name algorithms.arrays.trimmean --output-path py
nguin_tests/ --maximum_search_time=60
[False, 1, 1, 2, 1, 3, 'a', 0, 0]
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$
```

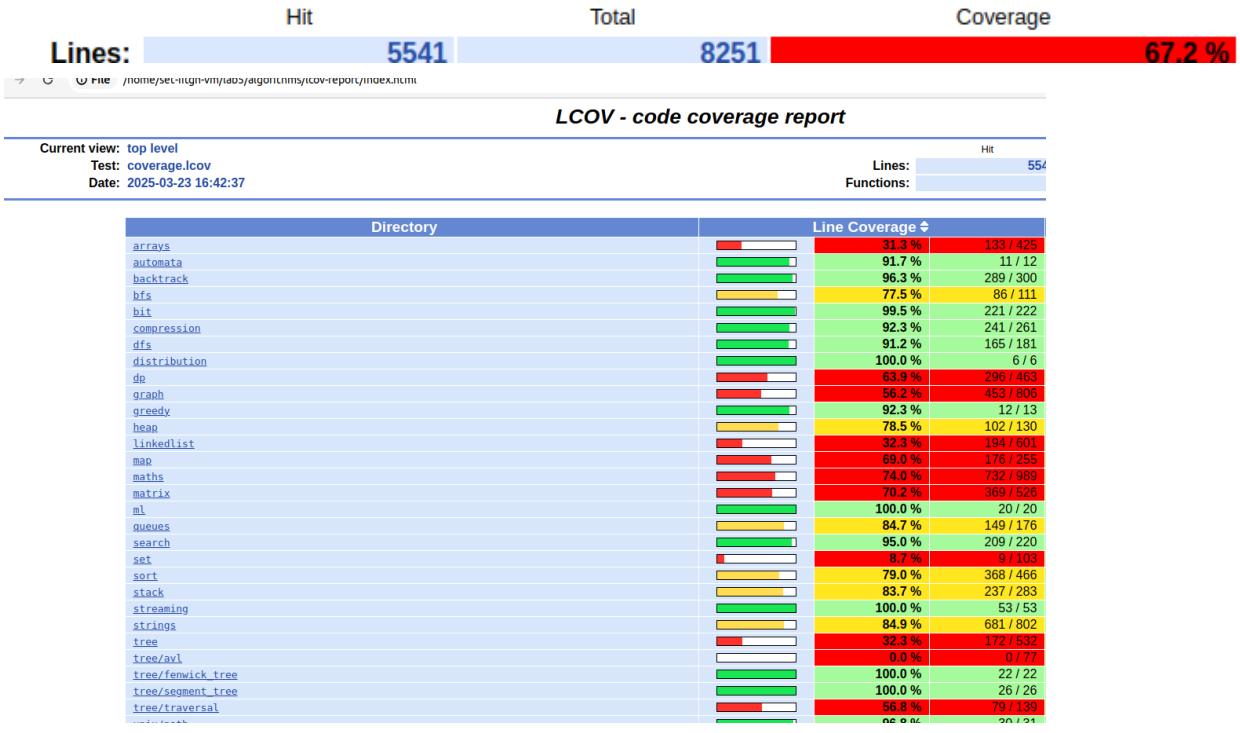
```
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ pynguin --project-path . --module_name algorithms.arrays.delete_nth --output-path
pynguin_tests/ --maximum_search_time=60
[False, 1, 1, 2, 1, 3, 'a', 0, 0]
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ pynguin --project-path . --module_name algorithms.arrays.flatten --output-path pyn
guin_tests/ --maximum_search_time=60
[False, 1, 1, 2, 1, 3, 'a', 0, 0]
set-iitgn-vm@set-iitgn-vm:~/lab5/algorithms$ pynguin --project-path . --module_name algorithms.arrays.garage --output-path pyng
uin_tests/ --maximum_search_time=60
[False, 1, 1, 2, 1, 3, 'a', 0, 0]
```

Then I ran the tests and generated tests together using command: *pytest tests/ pynguin_tests/ --cov=algorithms --cov-branch --cov-report=xml --cov-report=html --cov-report=term-missing* and the coverage increased:

```
algorithms/unix/path/join_with_slash.py                    6      0      0      0   100%
algorithms/unix/path/simplify_path.py                     11      1      6      1    88%   26
algorithms/unix/path/split.py                              7      0      0      0   100%
-----------------------------------------------------------------------------------------------
TOTAL                                                   7999   2710   4057    241    65%
Coverage HTML written to dir htmlcov
Coverage XML written to file coverage.xml
```

I ran the command:
coverage lcov -o coverage.lcov to convert coverage file to lcov file
Then the command:
genhtml coverage.lcov --output-directory lcov-report
To generate Lcov html report and to open in browser:
xdg-open lcov-report/index.html

Updated coverage report:



The coverage increased from 64% to 67.2% after penguin tests were generated.

Analysis:

- The original test suite (Test Suite A) provided reasonable coverage (64%) but left several lines, branches, and functions untested.
- Pynguin's automated test generation increased coverage from **64% to 67.2%**.
- The additional tests primarily improved branch and function coverage by exploring rare code paths. However, some complex logic and error-handling scenarios still remained uncovered.
- The coverage html and genhtml visualizations effectively highlighted problematic areas (e.g., red lines for missed code and yellow for partial coverage)

Challenges:

- When I ran the generated tests and existing tests, it gave less coverage at first but then after running pynguin on less coverage files, the coverage improved after several takes.
- Initial test failures like in test_array.py required manual correction of them.

Summary and Takeaways:

Initial coverage with the given tests in the repo is 64%, after using penguin it increased to 67.2%. HTML and LCov visualizations helped identify uncovered code sections. Automated testing tools like Pynguin are effective for improving coverage but require strategic use for optimal results. If we run it more on uncovered files, the coverage will be increased even more.

CS202 SOFTWARE TOOLS AND TECHNIQUES
LAB-6 : Python Test Parallelization

Lab Topic: Python Test Parallelization

Introduction:
Parallel test execution improves testing efficiency by running tests concurrently instead of sequentially. This lab explores the challenges of test parallelization using pytest-xdist (process-level) and pytest-run-parallel (thread-level) across two open-source Python repositories. The study identifies flaky tests, analyzes execution speedups, and examines issues like shared resource conflicts. By comparing different parallelization modes, this lab evaluates the readiness of test suites for parallel execution and highlights key limitations.

SETUP:
I created two directories, 'lab6a' and 'lab6b' for two repositories to proceed with the other steps. First navigate into that directory and then I did the other steps.

```
C:\Users\sirig>cd C:\Users\sirig\OneDrive\Desktop\lab6a

C:\Users\sirig\OneDrive\Desktop\lab6a>
```

TOOLS:
• pytest (test execution)
• pytest-xdist (process level test parallelization2)
• pytest-run-parallel (thread level test parallelization3)

METHODOLOGY AND EXECUTION:

1. To clone the given github repository I used the command:
   git clone https://github.com/keon/algorithms.git
   Then navigate into the folder using *cd algorithms*
   Then to get the current commit hash we can run the command: *git rev-parse HEAD*
   It returned the current commit hash: cad4754bc71742c2d6fcbd3b92ae74834d359844

```
C:\Users\sirig\OneDrive\Desktop\lab6a>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 | 3.34 MiB/s, done.
Resolving deltas: 100% (3241/3241), done.

C:\Users\sirig\OneDrive\Desktop\lab6a>cd algorithms

C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>git rev-parse HEAD
cad4754bc71742c2d6fcbd3b92ae74834d359844

C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>
```

Then created a virtual environment in the algorithms directory:

```
C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>python -m venv lab6a

C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>lab6a\Scripts\activate

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pip install -r requirements.txt

[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pip install -r test_requirements.txt
```

Installed the dependencies using the command: *pip install -r test_requirements.txt*

```
(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pip install -r test_requirements.txt
Collecting flake8 (from -r test_requirements.txt (line 1))
  Downloading flake8-7.1.1-py2.py3-none-any.whl.metadata (3.8 kB)
Collecting python-coveralls (from -r test_requirements.txt (line 2))
  Downloading python_coveralls-2.9.3-py2.py3-none-any.whl.metadata (6.1 kB)
Collecting coverage (from -r test_requirements.txt (line 3))
  Downloading coverage-7.6.12-cp312-cp312-win_amd64.whl.metadata (8.7 kB)
Collecting nose (from -r test_requirements.txt (line 4))
  Downloading nose-1.3.7-py3-none-any.whl.metadata (1.7 kB)
Collecting pytest (from -r test_requirements.txt (line 5))
  Downloading pytest-8.3.4-py3-none-any.whl.metadata (7.5 kB)
Collecting tox (from -r test_requirements.txt (line 6))
  Downloading tox-4.24.1-py3-none-any.whl.metadata (3.7 kB)
Collecting black (from -r test_requirements.txt (line 7))
  Downloading black-25.1.0-cp312-cp312-win_amd64.whl.metadata (81 kB)
Collecting mccabe<0.8.0,>=0.7.0 (from flake8->-r test_requirements.txt (line 1))
  Downloading 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))
  Downloading 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))
  Downloading pyflakes-3.2.0-py2.py3-none-any.whl.metadata (3.5 kB)
Collecting PyYAML (from python-coveralls->-r test_requirements.txt (line 2))
  Downloading PyYAML-6.0.2-cp312-cp312-win_amd64.whl.metadata (2.1 kB)
Collecting requests (from python-coveralls->-r test_requirements.txt (line 2))
  Using cached requests-2.32.3-py3-none-any.whl.metadata (4.6 kB)
Collecting six (from python-coveralls->-r test_requirements.txt (line 2))
  Using cached six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Collecting colorama (from pytest->-r test_requirements.txt (line 5))
```

I ran the command *pip list,* to check the packages and it had all the requirements present in test_requirements:

```
(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pip list
Package            Version
------------------ ---------
black              25.1.0
cachetools         5.5.1
certifi            2025.1.31
chardet            5.2.0
charset-normalizer 3.4.1
click              8.1.8
colorama           0.4.6
coverage           7.6.12
distlib            0.3.9
filelock           3.17.0
flake8             7.1.1
idna               3.10
iniconfig          2.0.0
mccabe             0.7.0
mypy-extensions    1.0.0
nose               1.3.7
packaging          24.2
pathspec           0.12.1
pip                24.3.1
platformdirs       4.3.6
pluggy             1.5.0
pycodestyle        2.12.1
pyflakes           3.2.0
pyproject-api      1.9.0
pytest             8.3.4
python-coveralls   2.9.3
PyYAML             6.0.2
```

2. **Sequential Test Execution:**

   I ran the command *pytest --disable-warnings > sequential_run_1.txt* for 10 times to execute the existing full test suite.

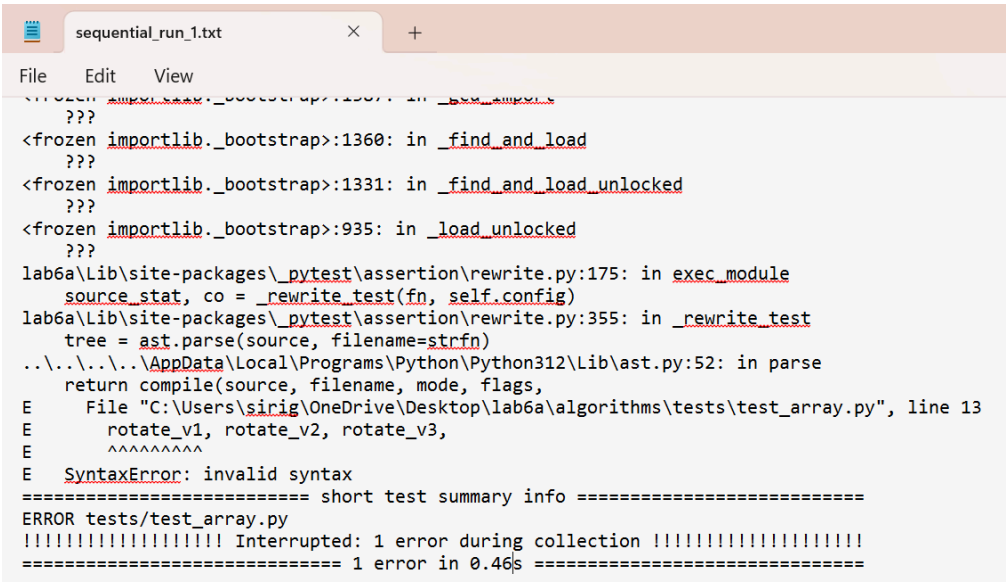
   At first it was throwing 29 errors most of them were ‘No module named algorithms’

   So I ran the command "pip install -e ."

```
(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pytest --disable-warnings > sequential_run_1.txt

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pytest --disable-warnings > sequential_run_2.txt

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pytest --disable-warnings > sequential_run_3.txt

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pytest --disable-warnings > sequential_run_4.txt

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pytest --disable-warnings > sequential_run_5.txt

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pytest --disable-warnings > sequential_run_6.txt

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pytest --disable-warnings > sequential_run_7.txt

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pytest --disable-warnings > sequential_run_8.txt

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pytest --disable-warnings > sequential_run_9.txt

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pytest --disable-warnings > sequential_run_10.txt

(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>
```

   For all the 10 times I got the same error:

   It is Syntax error:



Each txt file has the same error. Then to eliminate the failing test case, I commented out the tests/test_array.py and also tests/test_unix.py

Now there were no failures. I again ran the command *pytest --disable-warnings > sequential_run_11.txt* 3 times, so I got output in sequential_run_11.txt, sequential_run_12.txt, sequential_run_13.txt. I got the times:

== 383 passed in 3.87s==

== 383 passed in 3.77s ==

== 383 passed in 3.94s ==

T_{seq} = average of the three times, which is (3.87 + 3.77 + 3.94)/3 = 3.86.

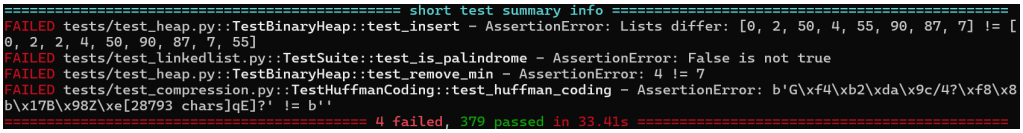
Hence T_{seq} = 3.86 seconds

3. Parallel Test Execution:

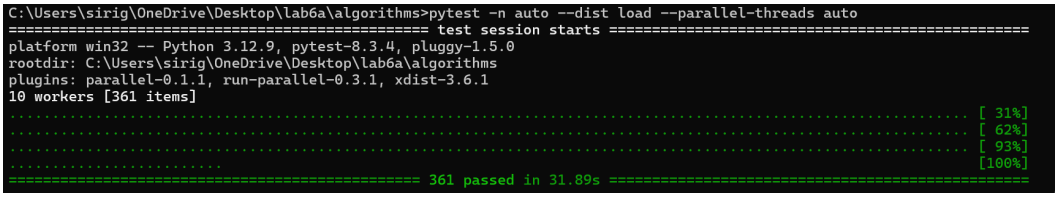
I installed pytest-xdist

```
(lab6a) C:\Users\sirig\OneDrive\Desktop\lab6a\algorithms>pip install pytest-xdist
Collecting pytest-xdist
  Downloading pytest_xdist-3.6.1-py3-none-any.whl.metadata (4.3 kB)
Collecting execnet>=2.1 (from pytest-xdist)
  Downloading execnet-2.1.1-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: pytest>=7.0.0 in c:\users\sirig\onedrive\desktop\lab6a\algorithms\lab6a\lib\site-packages
 (from pytest-xdist) (8.3.4)
Requirement already satisfied: colorama in c:\users\sirig\onedrive\desktop\lab6a\algorithms\lab6a\lib\site-packages (fro
m pytest>=7.0.0->pytest-xdist) (0.4.6)
Requirement already satisfied: iniconfig in c:\users\sirig\onedrive\desktop\lab6a\algorithms\lab6a\lib\site-packages (fr
om pytest>=7.0.0->pytest-xdist) (2.0.0)
Requirement already satisfied: packaging in c:\users\sirig\onedrive\desktop\lab6a\algorithms\lab6a\lib\site-packages (fr
om pytest>=7.0.0->pytest-xdist) (24.2)
Requirement already satisfied: pluggy<2,>=1.5 in c:\users\sirig\onedrive\desktop\lab6a\algorithms\lab6a\lib\site-package
s (from pytest>=7.0.0->pytest-xdist) (1.5.0)
Downloading pytest_xdist-3.6.1-py3-none-any.whl (46 kB)
Downloading execnet-2.1.1-py3-none-any.whl (40 kB)
Installing collected packages: execnet, pytest-xdist
Successfully installed execnet-2.1.1 pytest-xdist-3.6.1

[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

- So first combination of configuration: -n auto , –parallel-threads auto and parallelization mode: --dist load by running the command *pytest -n auto --dist load --parallel-threads auto* three times,
  First I got the errors:



  Then after removing those errors:



  I got the execution times: 361 passed in 31.89s, 361 passed in 55.95s, 361 passed in 30.86s
  Hence T_{avg} for this configuration is T_{par} = (31.89 + 55.95 + 30.86)/3 = 39.567
  T_{par} = 39.567s

- For the second combination, configuration: -n auto , –parallel-threads 1 and parallelization mode: --dist load
  no, by running the command *pytest -n auto --dist load --parallel-threads 1* three times:
  ======= 361 passed in 6.23s =======
  ======= 361 passed in 6.36s ========
  ======= 361 passed in 7.40s ========
  and I got the execution times: . **T_{par} = (6.23 + 6.36 + 7.40)/3 = 6.663s**
- For 3rd combination, configuration: -n 1, --parallel-threads auto and parallelization mode: - -dist load, by running the command *pytest -n 1 --dist load --parallel-threads auto* three times and I got execution time:
  ===== 361 passed in 39.69s =====
  ===== 361 passed in 36.26s =====
  ====== 361 passed in 34.08s =======
  T_{par} = (39.69 + 36.26 + 34.08)/3 = 36.67s

- For 4th combination, configuration: -n 1, --parallel-threads 1 and parallelization mode: - -dist load, by running the command *pytest -n 1 --dist load --parallel-threads 1* three times and I got execution time:
  ====== 361 passed in 4.63s ======
  ======= 361 passed in 4.28s ======

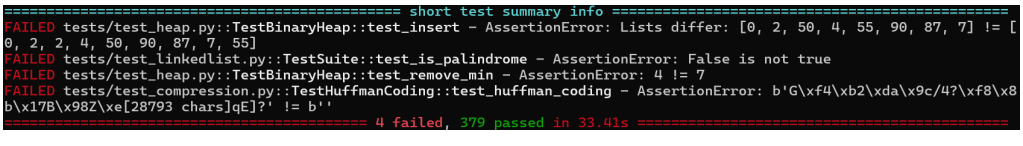
======= 361 passed in 4.41s =======

T_{par} = (4.63 + 4.28 + 4.41)/3 = 4.44s

- For 5th combination, configuration: -n auto, --parallel-threads auto and parallelization mode: - -dist no, by running the command *pytest -n auto --dist no --parallel-threads auto* three times and I got execution time:
  ======= 361 passed in 31.94s =======
  ======= 361 passed in 33.15s ==========
  ======= 361 passed in 38.92s =======
  T_{par} = (31.94 + 33.15 + 38.92)/3 = 34.67s
- For 6th combination, configuration: -n auto, --parallel-threads 1 and parallelization mode: - -dist no, by running the command *pytest -n auto --dist no --parallel-threads 1* three times and I got execution time:
  ======= 361 passed in 6.48s =======
  ======= 361 passed in 6.15s ======
  ======= 361 passed in 6.20s ========
  T_{par} = (6.48 + 6.15 + 6.20)/3 = 6.277s
- For 7th combination, configuration: -n 1, --parallel-threads 1 and parallelization mode: - -dist no, by running the command *pytest -n 1 --dist no --parallel-threads 1* three times and I got execution time:
  ======= 361 passed in 4.68s =======
  ====== 361 passed in 4.35s ======
  ======== 361 passed in 4.51s ======
  T_{par} = (4.68 + 4.35 + 4.51)/3 = 4.513s
- For 8th combination, configuration: -n 1, --parallel-threads auto and parallelization mode: - -dist no, by running the command *pytest -n 1 --dist no --parallel-threads auto* three times and I got execution time:
  ====== 361 passed in 36.72s ======
  ====== 361 passed in 35.12s ======
  ======= 361 passed in 34.19s =======
  T_{par} = (36.72 + 35.12 + 34.19)/3 = 35.343s

ANALYSIS:

The new test cases failed are(other than sequential: flaky tests):



Causes of all these test cases are Assertion errors.

1. tests/test_heap.py : Possible Causes:

Shared state issue: Multiple tests modifying the same heap instance concurrently.

Race condition: Insertions are interleaved, leading to incorrect heap order.

2. tests/test_linkedlist.py: Possible Causes:

Shared object mutation: If multiple tests operate on the same linked list instance, mutations can affect expected results.

Concurrency issue: Multiple threads modifying the list structure simultaneously.

3. tests/test_heap.py: Possible causes:

Race condition: Another test may have modified the heap structure before remove_min() executed.

Timing issue: Parallel execution changes the sequence of insertions/removals.

4. tests/test_compression.py

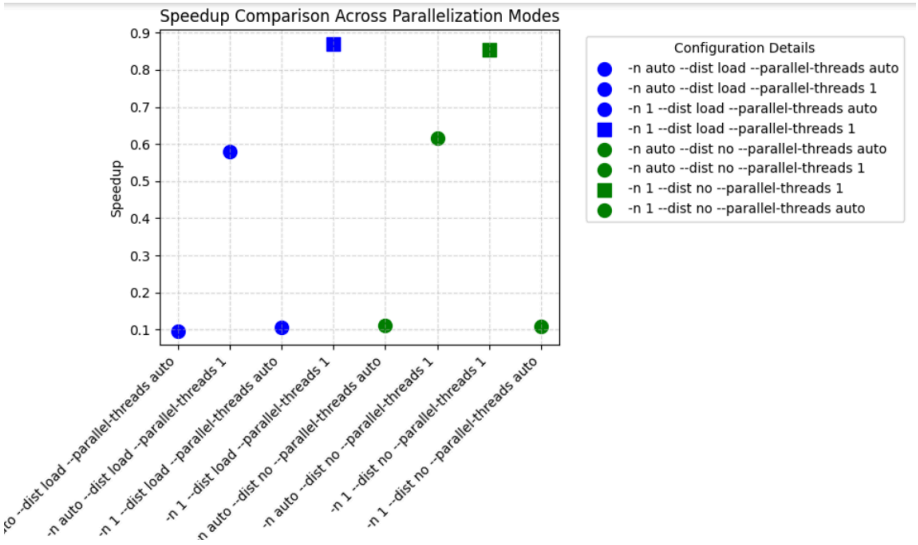
File system conflicts: Multiple tests writing to the same file concurrently.

Concurrency issue: Parallel encoding/decoding causing inconsistent outputs.

Speedup with respect to Sequential mode:

Parallelisation Mode and worker counts	**Worker count**	**Avg Execution Time**	**SpeedUp = T_{seq} / T_{par}**
-n auto --dist load --parallel-threads auto	auto	39.567	0.097
-n auto --dist load --parallel-threads 1	auto	6.663	0.579
-n 1 --dist load --parallel-threads auto	1	36.67	0.105
-n 1 --dist load --parallel-threads 1	1	4.44	0.869
-n auto --dist no --parallel-threads auto	auto	34.67	0.111
-n auto --dist no --parallel-threads 1	auto	6.277	0.615
-n 1 --dist no --parallel-threads 1	1	4.513	0.855
-n 1 --dist no --parallel-threads auto	1	35.343	0.109

The plot I generated is:



Challenges:

- Encountered multiple errors in the test suite. Resolved by identifying problematic files (test_array.py and test_unix.py) and excluding them.
- The repository initially lacked proper module references (No module named 'algorithms'). Fixed by running pip install -e ..

Summary and Takeaways:

Through this lab, I have successfully explored the impact of parallel test execution using pytest-xdist and pytest-run-parallel. Among parallel configurations the best time was 4.44s . The highest execution time was with -n auto --dist load --parallel-threads auto (39.57s) probably due to the overhead of excessive parallelism. Flaky tests arised due to race conditions, shared resource conflicts during parallelization. With lower thread counts, the execution time was less.

CS202 SOFTWARE TOOLS AND TECHNIQUES
LAB -7&8 : Vulnerability Analysis on Open-Source Software Repositories

Lab Topic: Vulnerability Analysis on Open-Source Software Repositories

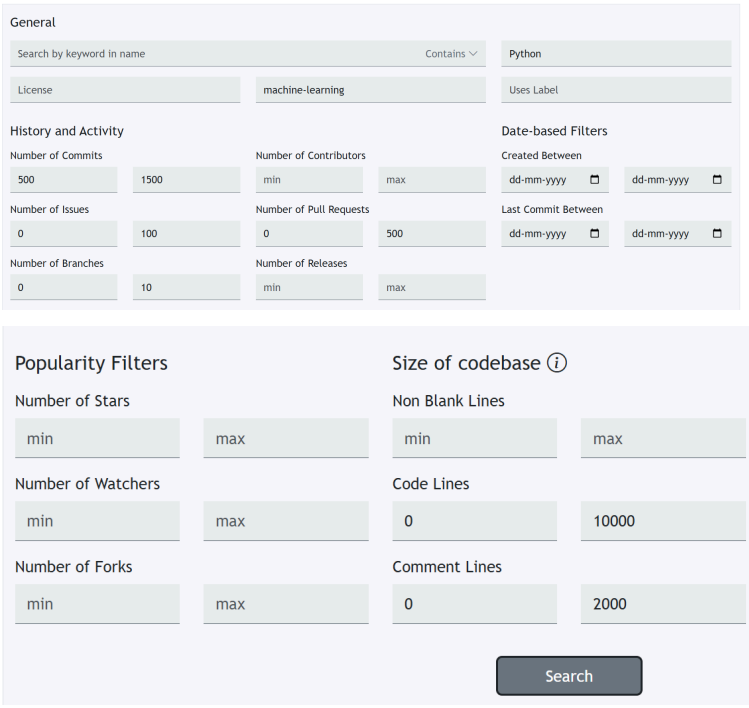
Introduction: In this lab, we get familiarized with the bandit tool. Identifying vulnerabilities in open-source repositories helps improve code quality and mitigate potential risks. In this lab, I utilized the Bandit tool to perform a vulnerability analysis on selected Python-based open-source repositories. The objective is to analyze vulnerabilities based on their confidence, severity, and associated CWE identifiers.

Setup and Tools:

- Operating system
- Python
- Bandit
- Created a directory named 'lab7' in the set-iitgn-vm in which I cloned the three repositories to perform bandit analysis.

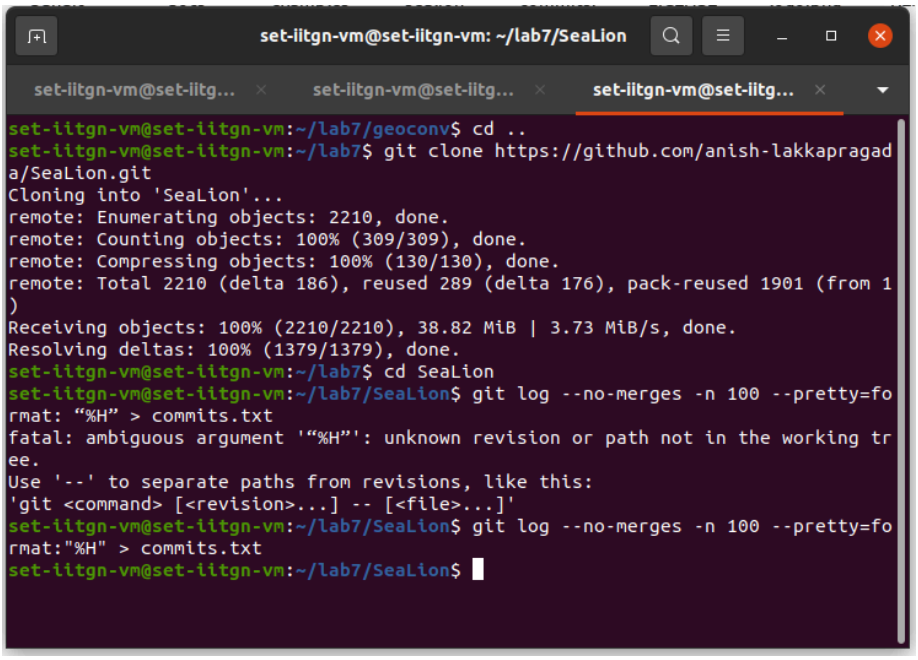
Methodology and Execution:

1. Choosing open source repositories in python:
   Using github search engine, I kept the following criteria
   Topic - Machine learning
   Language - Python
   Number of Commits : 500-1500
   No. of issues : 0-100
   No. of branches : 0-10
   No. of pull requests : 0-100
   No. of code lines : 0-10000
   No. of comments : 0-2000
   I used these criteria to get a large open source project.



And choose the following repositories:
https://github.com/andreazignoli/pyoxynet
https://github.com/andreasMazur/geoconv
https://github.com/anish-lakkapragada/sealion

2. I cloned the repositories and



I installed bandit using command *pip install bandit*

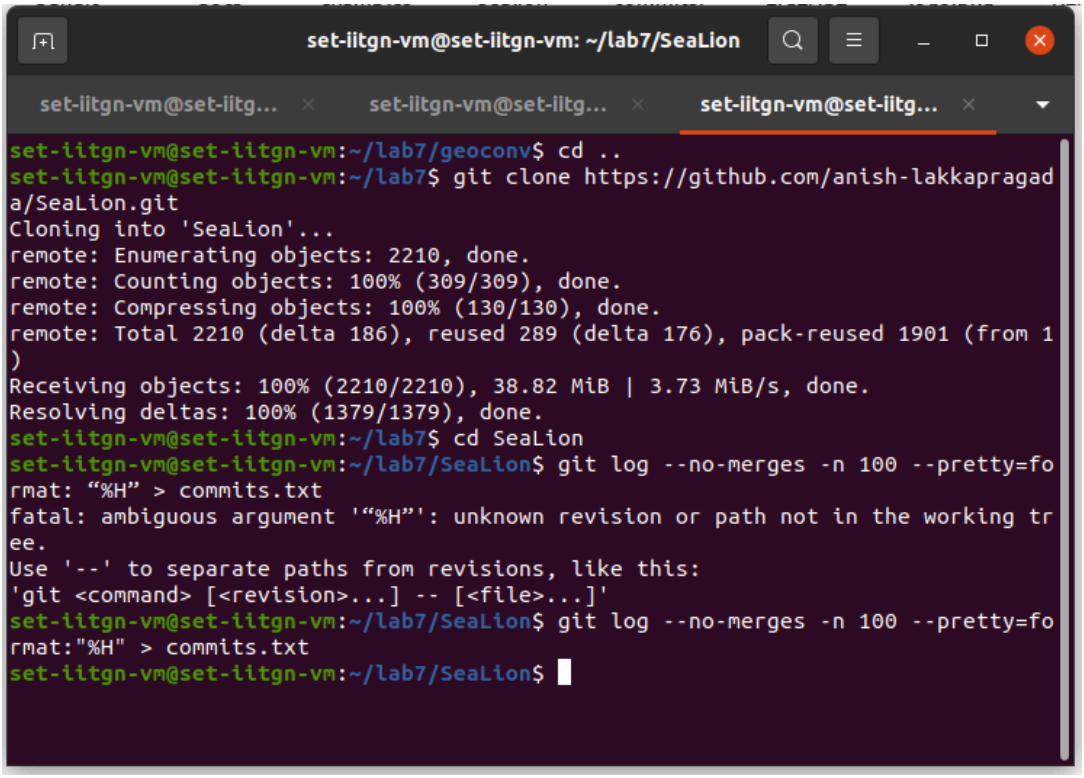
Then for the repo, I first stored the last 100 commits using the command:
git log --no-merges -n 100 --pretty=format: “%H” > commits.txt

The commits are stored in commits.txt. Then I ran the script(run.sh):

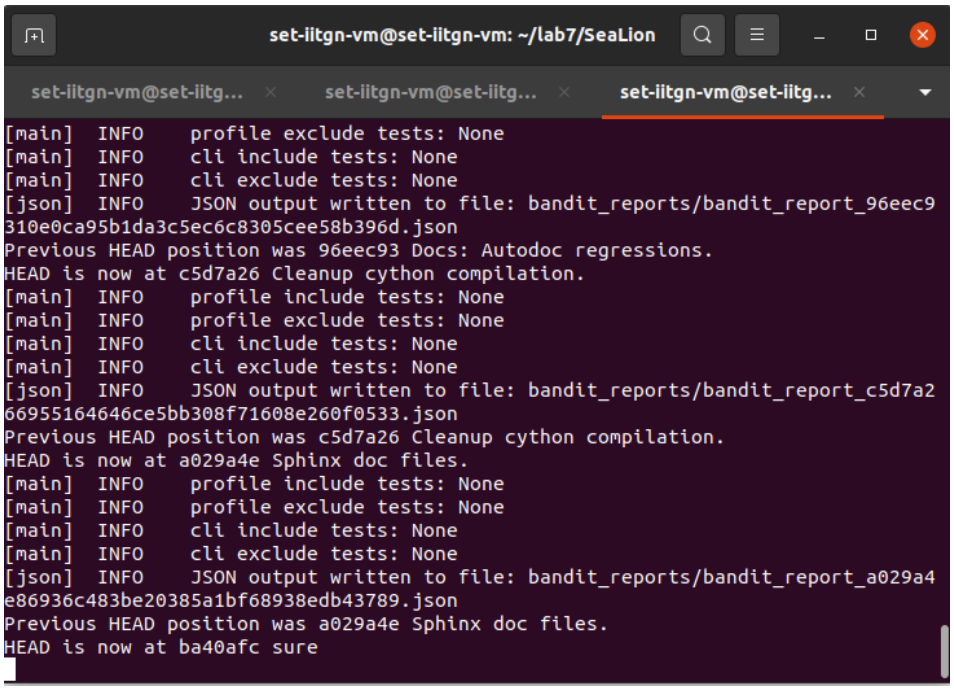
```
#!/bin/bash
while read commit; do
    git checkout $commit
    bandit -r . -f json -o bandit_reports/bandit_report_$commit.json
done < commits.txt
# Return to the main branch after processing
git checkout main
```

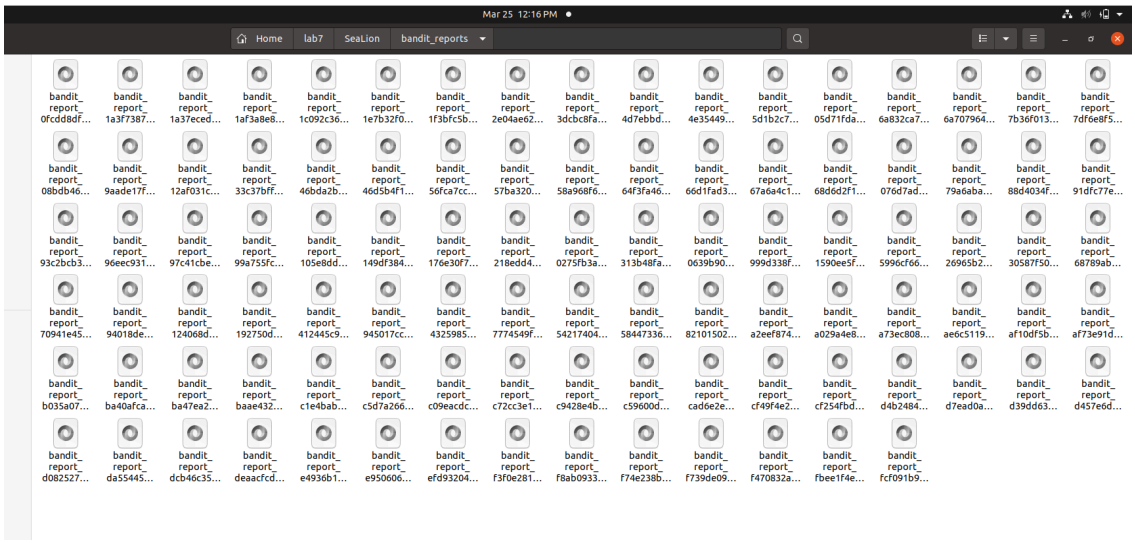
To collect bandit reports in json format.

```
Working... ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:16:46
[json]  INFO     JSON output written to file: bandit_reports/bandit_report_d9093c
85c56cf3fd33c2ec55eb02843585dd3e71.json
```



The generating bandit reports:

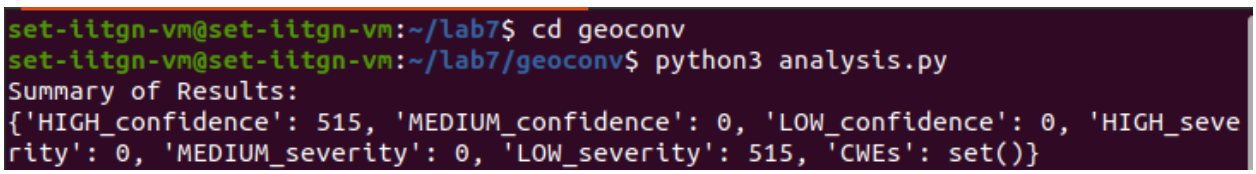


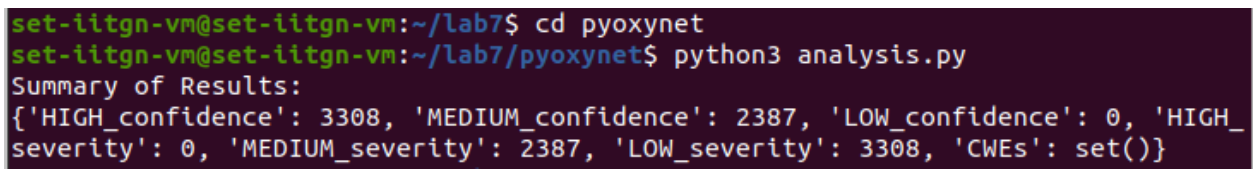


In .json format.

After generating for 3 repositories,

4. After analysis of the bandit reports using analysis.py, I got the following results for:





```
set-iitgn-vm@set-iitgn-vm:~/lab7$ cd SeaLion
set-iitgn-vm@set-iitgn-vm:~/lab7/SeaLion$ python3 analysis.py
Summary of Results:
{'HIGH_confidence': 2039, 'MEDIUM_confidence': 0, 'LOW_confidence': 0, 'HIGH_sev
erity': 104, 'MEDIUM_severity': 59, 'LOW_severity': 1876, 'CWEs': set()}
```

For identifying the CWE's I wrote another python script (cwes.py) to identify the unique CWE's In each one I got:

```
set-iitgn-vm@set-iitgn-vm:~/lab7/SeaLion$ python3 cwes.py
Unique CWEs Identified: {'CWE-78', 'CWE-703', 'CWE-330', 'CWE-502'}
set-iitgn-vm@set-iitgn-vm:~/lab7/SeaLion$
```

```
set-iitgn-vm@set-iitgn-vm:~/lab7/pyoxynet$ python3 cwes.py
Unique CWEs Identified: {'CWE-259', 'CWE-605', 'CWE-330', 'CWE-78', 'CWE-502', '
CWE-703', 'CWE-377'}
set-iitgn-vm@set-iitgn-vm:~/lab7/pyoxynet$
```

```
set-iitgn-vm@set-iitgn-vm:~/lab7/geoconv$ python3 cwes.py
Unique CWEs Identified: {'CWE-703'}
```

Therefore:

Repo	High confidence	Medium confidence	Low confidence	High severity	Medium severity	Low severity	unique CWEs
geoconv	515	0	0	0	0	515	1
pyoxynet	3308	2387	0	0	2387	3308	7
SeaLion	2039	0	0	104	59	1876	4

Unique CWEs in
Geoconv: CWE-703
Pyoxynet: CWE -259, CWE-605, CWE-330, CWE-78, CWE-502, CWE-703, CWE-377
SeaLion: CWE-703

RQ1: When are vulnerabilities with high severity, introduced and fixed3 along the development timeline in OSS repositories?
Purpose: To determine the timeline of high-severity vulnerability introduction and later fixed along the development.
Approach: For each repository, I examined the timestamps of commits where high-severity vulnerabilities were reported. I tracked when these vulnerabilities appeared and when they were subsequently fixed by comparing reports from successive commits.
Results: SeaLion: High-severity vulnerabilities appeared in earlier commits and were mostly resolved in later commits closer to the current state of the repository.
Pyoxynet and Geoconv had zero high-severity vulnerabilities

Takeaway: High-severity vulnerabilities were more frequent in the early stages of development and gradually eliminated as code reviews and updates improved security practices.

RQ2: Do vulnerabilities of different severity have the same pattern of introduction and elimination?
Purpose: To analyze whether vulnerabilities of varying severities follow a similar timeline in their introduction and removal.
Approach: I compared the frequency and timeline of medium- and low-severity issues across the repositories.
Results: Pyoxynet: Medium-severity issues persisted across multiple commits, while low-severity issues were frequently introduced and resolved in quick succession.
Geoconv: Only low-severity issues were present, and they steadily increased over time without active resolution.
SeaLion: High-severity issues were clustered early, while medium- and low-severity issues appeared throughout the timeline.

Takeaway: Low-severity issues tend to persist longer in repositories unless explicitly addressed, while high-severity vulnerabilities are prioritized for removal once identified.

RQ3: Which CWEs are the most frequent across different OSS repositories?
Purpose: To identify the most common CWEs across the selected repositories.
Approach: I extracted CWE identifiers from each repository's Bandit report using cwes.py.
Results: Most identified CWE is CWE-703 (Improper Check or Handling of Exceptional Conditions). It is found in all the 3 repositories.
Takeaway: CWE-703 (Improper Check or Handling of Exceptional Conditions) was common in all repositories, highlighting the importance of robust error handling practices in Python projects.

Challenges:

- Generating bandit reports for a certain repository was very time consuming hence I chose another repository which worked well.
- Parsing the json files for the insights of confidence and severity and cwes.

Summary:

Pyoxynet reported the highest number of high-confidence issues (3308), indicating strongly that these flagged issues are valid concerns requiring attention. Pyoxynet also reported 2387 medium-confidence issues, showing a moderate risk across several parts of the codebase. Geoconv had only low severity issues. SeaLion had 104 high severity indicating need for robust fixes. High-confidence issues often correlate with high-severity vulnerabilities. CWE-703 was

the most frequently observed vulnerability, emphasizing the importance of secure error handling mechanisms in Python projects.