python

Snippet ID: breadth_first_search

1. Category: BUG

- **Explanation:**
    - **What the issue is:** The condition in the while loop `while True:` will lead to an infinite loop if the queue is empty and the `goalnode` is not found.
    - **Why it's a problem:** The code will continue executing indefinitely, which might cause the program to hang if the goal node is not part of the graph.
    - **How to fix it:** Change the condition to `while queue:` to ensure the loop stops when there are no more nodes to process.

2. Category: BUG

- **Explanation:**
    - **What the issue is:** The line `nodesseen.update(node.successors)` updates the set `nodesseen` with all successors, even those already seen.
    - **Why it's a problem:** This can lead to incorrect results or wasted operations by trying to add already seen nodes.
    - **How to fix it:** Update only with successors that haven't been seen: `nodesseen.update(node for node in node.successors if node not in nodesseen)`.

3. Category: QUALITY

- **Explanation:**
    - **What the issue is:** Inconsistent and non-descriptive variable naming (`startnode`, `goalnode`, `nodesseen`).
    - **Why it's a problem:** It decreases code readability and understandability.
    - **How to fix it:** Use snake_case for variable names: `start_node`, `goal_node`, `nodes_seen`.

4. Category: QUALITY

- **Explanation:**
    - **What the issue is:** Redundant use of `from collections import deque as Queue` and referring to it as `Queue`.
    - **Why it's a problem:** Using `Queue` to refer to `deque` might confuse readers into thinking it's related to the `queue` module.
    - **How to fix it:** Use `deque` directly instead of aliasing it, or use a more descriptive alias if aliasing is necessary.

5. Category: BUG

- **Explanation:**
    - **What the issue is:** The variable `node` is iteratively overwritten in the line `queue.extend(node for node in node.successors if node not in nodesseen)`, which conflates two different usages of `node`.
    - **Why it's a problem:** This can confuse the intended iteration logic and might lead to errors if a similar context changes the node later.
    - **How to fix it:** Use a different variable name inside the generator expression, such as `successor` for clarity: `queue.extend(successor for successor in node.successors if successor not in nodesseen)`.

Total bugs: 3
Total quality issues: 2

Snippet ID: cli_calculator

1. Category: BUG

- **Explanation:**
    - Division by zero is not handled.

    - This will cause a runtime error if the user inputs zero as the second number and chooses division.
    - Add a check to ensure that if the operation is division, the second number is not zero.

2. Category: QUALITY

- **Explanation:**
    - The function `calculate()` does not return any value, which makes testing and reuse difficult.
    - Returning the result of the calculation allows this function to be used programmatically rather than being limited to just console I/O.
    - Modify the `calculate()` function to return the calculation result instead of or in addition to printing it.

3. Category: QUALITY

- **Explanation:**
    - Magic strings are used directly in the code for operations ('+', '-', '*', '/').
    - Magic strings can lead to errors and make the code less readable.
    - Define constants for the operations to improve readability and maintainability.

Total bugs: 1
Total quality issues: 2

Snippet ID: find_in_sorted

1. Category: BUG

- **Explanation:**
    - The current implementation of the binary search checks if `start == end` to terminate the search and return -1. This condition is incorrect. In a typical binary search, you should check `if start >= end` to terminate the search, as this implies the target element

doesn't exist in the array.

    - This is a problem because the function might return -1 prematurely, even if the item exists in the array.

    - To fix it, change the condition to `if start >= end`.

2. Category: BUG

- **Explanation:**

    - When `mid` is compared to `x` and found to be greater, the function calls `binsearch(start, mid)`. However, `mid` should be excluded from the next search range since it's already confirmed that `x` is not at `mid` and `x < arr[mid]`.

    - This can cause an infinite loop for a single-element segment, perpetually narrowing the range without making progress.

    - Fix it by modifying the call to `binsearch(start, mid - 1)`.

3. Category: BUG

- **Explanation:**

    - When `x` is greater than `arr[mid]`, the function calls `binsearch(mid, end)`. Since `mid` is already processed and `x > arr[mid]`, the next search should start from `mid + 1`.

    - This mistake may lead to an infinite loop for similar reasons as above.

    - Fix it by modifying the call to `binsearch(mid + 1, end)`.

4. Category: CODE QUALITY

- **Explanation:**

    - The function could benefit from better naming of the internal variables. For example, `binsearch` is not very descriptive. Renaming it to something like `binary_search` would improve code readability.

    - Poor naming can make the code harder to understand and maintain, especially for those not familiar with it.

    - Fix it by renaming `binsearch` to `binary_search`.

5. Category: CODE QUALITY

- **Explanation:**
    - The function `find_in_sorted` lacks a docstring explaining what it does, what the parameters are, and what it returns.
    - Lack of documentation makes the code less understandable and maintainable, especially for new developers or when revisiting the code after a long time.
    - Fix it by adding a docstring at the beginning of the function.

Total bugs: 3
Total quality issues: 2

Snippet ID: flask_app

1. Category: BUG

- **Explanation:**
    - The login function assumes that the username exists in the users dictionary without checking. If a non-existent username is provided, it will raise a KeyError.
    - **Why it's a problem:** This can lead to server errors when users attempt to log in with a username that isn't registered.
    - **How to fix it:** Check if the username exists in the users dictionary before comparing the password.

2. Category: BUG

- **Explanation:**
    - Both the register and login functions are storing and verifying passwords in plain text.
    - **Why it's a problem:** Storing plain text passwords is a significant security risk, as it exposes users' passwords to anyone with access to the data.
    - **How to fix it:** Use a password hashing library, such as bcrypt, to securely hash and verify passwords.

3. Category: QUALITY

- **Explanation:**
    - The users dictionary is stored at the global level, which is not thread-safe.
    - **Why it's a problem:** Flask's default server may handle requests in parallel, leading to race conditions when accessing or modifying shared data.
    - **How to fix it:** Use a thread-safe data storage method, such as a database or an external service, to handle user data.

4. Category: QUALITY

- **Explanation:**
    - Using HTTP methods incorrectly; the register and login routes respond successfully with a 200 status code by default.
    - **Why it's a problem:** This is technically correct but can be improved by explicitly defining the status code for clarity and maintainability.
    - **How to fix it:** Return an explicit 201 status code for successful registration and 200 for successful login.

Total bugs: 2
Total quality issues: 2

Snippet ID: knapsack

1. Category: BUG

- **Explanation:**
    - The conditional statement `if weight < j` is incorrect. It should be `if weight <= j`. The current logic skips the scenario where the item weight is exactly equal to the current capacity being checked (`j`).
    - **Why it's a problem:** Items whose weights exactly match the current capacity will not be considered, leading to incorrect results when they should be included.
    - **How to fix it:** Change the condition to `if weight <= j`.

2. Category: QUALITY

- **Explanation:**
    - The import statement `from collections import defaultdict` is inside the function.
    - **Why it's a problem:** Importing inside a function is not a common practice and can lead to confusion. It can also incur a small performance penalty if the function is called frequently.
    - **How to fix it:** Move the import statement to the top of the file.

3. Category: QUALITY

- **Explanation:**
    - The use of `defaultdict(int)` initializes all values with `0`, which can mask potential logic issues by returning a default value rather than an error.
    - **Why it's a problem:** Using a `defaultdict` without explicit initializations can lead to hidden bugs, as access to an uninitialized key returns a default value instead of raising a `KeyError`.
    - **How to fix it:** Use a regular dictionary `dict()` and manage key initialization explicitly within the algorithm.

4. Category: QUALITY

- **Explanation:**
    - Variable names `i` and `j` are not descriptive.
    - **Why it's a problem:** Non-descriptive names can make the code harder to read and maintain.
    - **How to fix it:** Use more meaningful names such as `item_idx` and `current_capacity`.

Total bugs: 1
Total quality issues: 3

Snippet ID: rpn_eval

1. BUG

- The issue is with the operand order in the operations.
- When popping elements from the stack, the first pop should be for the second operand and the second pop for the first operand as Reverse Polish Notation evaluates expressions by taking the second operand first.
- Swap the positions of `a` and `b` in the `op` function call like so: `op(token, b, a)`.

2. BUG

- The issue is missing error handling for division by zero.
- Dividing by zero will cause a runtime error and unexpected behavior.
- Add a check before performing division to handle division by zero, possibly by raising an exception or returning an error message.

3. QUALITY

- The issue is with the type check for tokens only accepting `float`.
- Restricting to `float` can be too limiting when dealing with whole numbers.
- Use `isinstance(token, (int, float))` instead to accommodate both integers and floats.

4. QUALITY

- The issue is with no error handling for invalid tokens.
- Invalid or unrecognized tokens will lead to a `KeyError` or empty stack errors.
- Introduce error handling to manage unexpected tokens, such as raising an exception if the token is not a number or a supported operation.

5. QUALITY

- The issue is with using dictionary access without checking key existence.
- Attempting to access a dictionary key without checking its existence can cause a `KeyError`.

- Utilize the `dict.get()` method with a default action to handle cases where the symbol might not be valid.

Total bugs: 2
Total quality issues: 3

Snippet ID: shortest_path_length

1. Category: BUG

- Explanation: The term "FibHeap" is misleading in the comment for `unvisited_nodes`. The implementation uses a binary heap (via `heapq`), not a Fibonacci heap.
- Why it's a problem: The performance characteristics differ significantly, which can mislead assumptions.
- How to fix it: Correct the comment or replace `heapq` with a Fibonacci heap if needed.

2. Category: BUG

- Explanation: The conditional check `if node is goalnode:` uses the `is` operator.
- Why it's a problem: Object identity check may fail even for logically equal nodes.
- How to fix it: Use `==` instead of `is`.

3. Category: BUG

- Explanation: The function `get` returns `0` when a node is not found.
- Why it's a problem: It should return `float('inf')` to reflect unknown distance.
- How to fix it: Change the return value to `float('inf')`.

4. Category: BUG

- Explanation: Redundant call to `get(unvisited_nodes, nextnode)` in `min()` function.
- Why it's a problem: Logical oversight and unnecessary duplication.
- How to fix it: Simplify by computing the new potential distance directly.

5. Category: QUALITY

- Explanation: Generic function names like `get` and `insert_or_update`.
- Why it's a problem: Reduces readability and clarity.
- How to fix it: Rename to `get_distance_or_inf`, `insert_or_update_node`.

6. Category: QUALITY

- Explanation: Code assumes `node.successors` always exists.
- Why it's a problem: Can cause runtime errors.
- How to fix it: Add check with `hasattr(node, 'successors')`.

Total bugs: 4
Total quality issues: 2

Snippet ID: shunting_yard

1. BUG

- Explanation: The function does not handle parentheses.
- Why it's a problem: It results in incorrect order of operations.
- How to fix it: Add logic for `(` and `)` handling with appropriate stack operations.

2. BUG

- Explanation: Operator not found in precedence dictionary can cause `KeyError`.
- Why it's a problem: Leads to runtime errors.
- How to fix it: Validate all tokens before accessing precedence.

3. QUALITY

- Explanation: Token type check only accepts `int`.

- Why it's a problem: Fails to handle `float` and other numeric types.
- How to fix it: Use `isinstance(token, (int, float))`.

4. QUALITY

- Explanation: No comments or docstrings.
- Why it's a problem: Decreases understandability and maintainability.
- How to fix it: Add function-level documentation and comments.

5. QUALITY

- Explanation: Poor variable names like `rpntokens`, `opstack`.
- Why it's a problem: Decreases readability.
- How to fix it: Rename to `output_queue`, `operator_stack`.

Total bugs: 2
Total quality issues: 3

Snippet ID: topological_ordering

1. Category: BUG

- Explanation: Incorrect logic for appending nodes based on `outgoing_nodes`.
- Why it's a problem: Results in invalid topological order.
- How to fix it: Use Kahn's algorithm with in-degree tracking.

2. Category: BUG

- Explanation: Condition `if set(ordered_nodes).issuperset(nextnode.outgoing_nodes)` is flawed.
- Why it's a problem: Checks the wrong relation.
- How to fix it: Base processing on incoming edges or in-degrees.

3. Category: QUALITY

- Explanation: Variable `nextnode` lacks clarity and doesn't follow naming conventions.
- Why it's a problem: Poor readability.
- How to fix it: Rename to `next_node`.

4. Category: QUALITY

- Explanation: Inefficient node scanning with list comprehension in loop.
- Why it's a problem: Performance cost.
- How to fix it: Precompute in-degrees and update iteratively.

Total bugs: 2
Total quality issues: 2

Snippet ID: wrap

1. Category: BUG

- Explanation: Remaining text after the loop isn't handled.
- Why it's a problem: Final segment may be dropped.
- How to fix it: Append remaining text after the loop ends.

2. Category: QUALITY

- Explanation: Function name `wrap` is misleading.
- Why it's a problem: Doesn't accurately describe behavior.
- How to fix it: Rename to `wrap_text_to_lines`.

3. Category: QUALITY

- Explanation: Variables `line` and `text` are reassigned confusingly.
- Why it's a problem: Reduces code clarity.

- How to fix it: Assign them separately for readability.

Total bugs: 1
Total quality issues: 2

Snippet ID: common

1. Category: BUG

- **Explanation:**
    - **Issue:** Using a mutable default argument (`resultList=[]`).
    - **Problem:** This can lead to unexpected behavior because default mutable arguments are shared across function calls, potentially accumulating changes from previous invocations.
    - **Fix:** Use `None` as the default value and instantiate a new list inside the function, if needed:
      `resultList=None` and then inside the function, check and set it: `if resultList is None: resultList = []`.

2. Category: QUALITY

- **Explanation:**
    - **Issue:** Poor naming of the function and variables.
    - **Problem:** Names like `doTask`, `x`, and `y` are generic and do not describe their purpose or the function's behavior, which reduces code readability and maintainability.
    - **Fix:** Use descriptive names for the function and variables like `calculate_area`, `radius`, `offset`, and `results` depending on their intended use.

3. Category: QUALITY

- **Explanation:**
    - **Issue:** Magic numbers (3.14 and 100).

- **Problem:** These numbers appear in the code without context or explanation, which makes the code harder to understand and update.
- **Fix:** Define constants with descriptive names such as `PI = 3.14` and `DEFAULT_Y = 100` at the top of the code file and use those constants in the function.

4. Category: QUALITY

- **Explanation:**
  - **Issue:** Unused variable `unused`.
  - **Problem:** Declaring a variable that isn't used is wasteful and potentially confusing for anyone reading the code.
  - **Fix:** Remove the unused variable to clean up the code.

Total bugs: 1
Total quality issues: 3

Snippet ID: deep_nesting

1. Category: BUG

- **Explanation:**
  - The function only prints "Odd and within range" but does not return any value for other cases of `x`.
  - **Why it's a problem:** Users of the function won't know what the function does for even numbers or numbers outside the 0–10 range, which can lead to confusion or incorrect assumptions.
  - **How to fix it:** Add print statements or return values for other conditions.

2. Category: QUALITY

- **Explanation:**
  - The function has excessive nested if-statements.
  - **Why it's a problem:** Makes the code harder to read and increases cognitive load.

- **How to fix it:** Refactor to use a single if statement, like: `if 0 < x < 10 and x % 2 == 1:`, and then use `elif`/`else` blocks.

Total bugs: 1
Total quality issues: 1

Snippet ID: hanoi

1. Category: QUALITY

- **Explanation:**
  - **What the issue is:** The use of a set comprehension `{1, 2, 3} - {start} - {end}` to determine the helper pole is unnecessary overhead.
  - **Why it's a problem:** It creates a new set and performs extra operations each time.
  - **How to fix it:** Use simple math: `helper = 6 - start - end`.

2. Category: QUALITY

- **Explanation:**
  - **What the issue is:** The variable name `helper` is not descriptive.
  - **Why it's a problem:** Poor readability and maintainability.
  - **How to fix it:** Rename to `auxiliary` or `intermediate_pole`.

Total bugs: 0
Total quality issues: 2

Snippet ID: kheapsort

1. Category: BUG

- **Explanation:**
  - **What the issue is:** `kheapsort` starts yielding elements after heapifying the first k but doesn't sort the entire array.

    - **Why it's a problem:** Misleading behavior based on name.
    - **How to fix it:** Process entire array properly, yield sorted smallest k elements.

2. Category: BUG

- **Explanation:**
    - **What the issue is:** Final while-loop yields heap elements not in order.
    - **Why it's a problem:** Output isn't fully sorted.
    - **How to fix it:** Sort and yield the remaining heap elements correctly.

3. Category: QUALITY

- **Explanation:**
    - **What the issue is:** The function name is misleading.
    - **Why it's a problem:** Implies full sort rather than partial.
    - **How to fix it:** Rename to something like `k_smallest_sort`.

4. Category: QUALITY

- **Explanation:**
    - **What the issue is:** No error handling for invalid `k` or empty input.
    - **Why it's a problem:** Can cause crashes or bad output.
    - **How to fix it:** Add input validation and edge case handling.

Total bugs: 2
Total quality issues: 2

Snippet ID: naming

1. Category: QUALITY

- **Explanation:**

- **What the issue is:** The function name `proc_data` is not descriptive.
- **Why it's a problem:** Harder to understand and maintain.
- **How to fix it:** Rename to something like `double_value` to clearly express its purpose.

Total bugs: 0
Total quality issues: 1

Snippet ID: naming_and_magic

1. Category: QUALITY

- **Explanation:**
  - **What the issue is:** The function name `calcArea` is vague and does not specify which area it is calculating (in this case, the area of a circle).
  - **Why it's a problem:** Vague naming reduces code readability and makes it harder to quickly understand the function's purpose.
  - **How to fix it:** Rename the function to something more descriptive, like `calculate_circle_area`.

2. Category: QUALITY

- **Explanation:**
  - **What the issue is:** The function uses a "magic number" for π (pi).
  - **Why it's a problem:** Magic numbers reduce code readability and using an approximation might lead to precision issues.
  - **How to fix it:** Replace the hardcoded number with `math.pi` from the standard library.

Total bugs: 0
Total quality issues: 2

Snippet ID: next_palindrome

1. Category: BUG
Explanation:
The code incorrectly handles arithmetic operations on digit positions when handling palindromes. Specifically, it does not appropriately mirror the increment across the symmetry of the list.

- **Why it's a problem:**
  The wrong positions are modified, leading to incorrect results for certain inputs.

- **How to fix it:**
  Adjust the logic to correctly identify and update the mirrored indices as you process towards the center, ensuring both sides of the palindrome are correctly managed.

2. Category: BUG
Explanation:
The condition `if digit_list[high_mid] == 9:` assumes that the only case that needs differing treatment is when the digit is 9, but doesn't correctly handle cases where multiple re-adjustments might be needed due to carried values.

- **Why it's a problem:**
  This doesn't account for transitions where multiple 9s lead to cascading carries (e.g., 999 transforming into 1001).

- **How to fix it:**
  Include a carry-over mechanism that will handle cascading through all positions effectively.

3. Category: QUALITY
Explanation:
The function name `next_palindrome` is somewhat misleading without context.

- **Why it's a problem:**
  It doesn't clearly indicate what the function is expected to do; naming could lead to misunderstandings of the function's purpose.

- **How to fix it:**
  Use a more descriptive name like `create_next_palindrome_number`, or add detailed documentation.

4. Category: QUALITY
Explanation:
Variable names `high_mid` and `low_mid` don't clearly convey their role.

- **Why it's a problem:** This reduces code readability and maintainability.

- **How to fix it:** Rename to `left_index` and `right_index`.

5. Category: QUALITY
Explanation:
The code has inconsistent use of `high_mid` and `low_mid`.

- **Why it's a problem:**
  It can cause confusion or errors due to asymmetrical control logic.

- **How to fix it:**
  Standardize the logic and loop conditions to maintain symmetry.

Total bugs: 2
Total quality issues: 3

Snippet ID: next_permutation

1. Category: BUG

- **Explanation:**
  - **What the issue is:** The function does not return a result if no next permutation is found (i.e., when the input permutation is the last in lexicographical order)
  - **Why it's a problem:** It returns `None` implicitly, leading to unexpected behavior.
  - **How to fix it:** Add an explicit return for the original permutation or raise an exception.

2. Category: QUALITY

- **Explanation:**
  - **What the issue is:** The variable `next_perm` is created unnecessarily as a copy of `perm`.
  - **Why it's a problem:** This introduces overhead and can be avoided.
  - **How to fix it:** Modify `perm` in place.

3. Category: QUALITY

- **Explanation:**
  - **What the issue is:** The function lacks comments or documentation.

- **Why it's a problem:** Reduces readability and maintainability.
- **How to fix it:** Add comments explaining key steps of the algorithm.

3. Category: QUALITY

- **Explanation:**
  - **What the issue is:** The use of variable names like `i` and `j` lacks clarity.
  - **Why it's a problem:** Obscures the logic of identifying pivot and successor.
  - **How to fix it:** Use descriptive names like `pivot` and `successor`.

Total bugs: 1
Total quality issues: 3

Snippet ID: powerset

1. Category: BUG

- **Explanation:**
  - **What the issue is:** No explicit base case for empty input.
  - **Why it's a problem:** Can cause confusion about how edge cases are handled.
  - **How to fix it:** Add `if not arr` as a clear base case.

2. Category: QUALITY

- **Explanation:**
  - **What the issue is:** Assumes `arr` is always a list.
  - **Why it's a problem:** Can confuse users passing other iterables.
  - **How to fix it:** Rename to `iterable` or coerce input with `list(arr)`.

3. Category: QUALITY

- **Explanation:**
    - **What the issue is:** No type hints provided.
    - **Why it's a problem:** Hurts readability and static analysis.
    - **How to fix it:** Add type hints, e.g., `def powerset(arr: list) -> list`.

Total bugs: 1
Total quality issues: 2

Snippet ID: unused_and_mutable

1. Category: BUG

- **Explanation:**
    - The function `store_data` uses a mutable default (`data=[]`).
    - **Why it's a problem:** Causes shared state across calls.
    - **How to fix it:** Use `data=None` and instantiate inside function:

```
def store_data(val, data=None):
    if data is None:
        data = []
    data.append(val)
    return data
```

2. Category: QUALITY

- **Explanation:**
    - Variable `tmp` is declared but unused.
    - **Why it's a problem:** Adds noise and confusion.
    - **How to fix it:** Remove the unused variable.

Total bugs: 1
Total quality issues: 1

---

php

Snippet ID: array_filter

1. Category: BUG

- The issue is that the code assigns a single value to `$result` when `$val` is not greater than `$limit`.
- **Why it's a problem:** `$result` is supposed to be an array collecting values, but current logic overwrites it.
- **How to fix it:** Remove the `else` clause or change it to skip adding `$val`.

2. Category: CODE QUALITY

- The issue is the lack of braces `{}` for the `if` and `else` statements.
- **Why it's a problem:** Reduces readability and increases risk of bugs.
- **How to fix it:** Add `{}` to both code blocks.

Total bugs: 1
Total quality issues: 1

Snippet ID: auth_check

1. Category: BUG

- **Explanation:**
    - `$password = "1234"` is an assignment, not a comparison.
    - **Why it's a problem:** Always evaluates as true; incorrect logic.

- **Fix:** Use `==` for comparison:
  `if ($username == "admin" && $password == "1234")`.

2. Category: BUG

- **Explanation:**
  - `if ($username = "")` is also an assignment.
  - **Why it's a problem:** Overwrites `$username` and breaks logic.
  - **Fix:** Change to `if ($username == "")`.

3. Category: QUALITY

- **Explanation:**
  - Function name `authenticate` is too generic.
  - **Why it's a problem:** May confuse devs in systems with multiple auth methods.
  - **Fix:** Rename to `authenticateWithStaticCredentials`.

4. Category: QUALITY

- **Explanation:**
  - Redundant conditions and return statements.
  - **Why it's a problem:** Decreases clarity and efficiency.
  - **Fix:** Refactor to:

```
function authenticate($username, $password) {

    if ($username == "admin" && $password == "1234") {
        return true;
    } elseif ($username == "") {
        return false;
    }
    return false;
```

```
}
```

Total bugs: 2
Total quality issues: 2

Snippet ID: discount_calculator

1. Category: BUG

- **Explanation:**
    - The discount is added instead of subtracted.
    - **Why it's a problem:** Returns incorrect price.
    - **Fix:** Use: `return $price - $discount;`.

2. Category: QUALITY

- **Explanation:**
    - No input validation for `$price` and `$rate`.
    - **Why it's a problem:** Risk of runtime warnings or errors.
    - **Fix:** Add type checks or casts for both inputs.

3. Category: QUALITY

- **Explanation:**
    - Inline `if` statement lacks braces.
    - **Why it's a problem:** Reduces readability and maintainability.

Fix: Use:

php

CopyEdit

```
if ($rate > 100 || $rate < 0) {
    return $price;
}
```

-

4. Category: QUALITY

- **Explanation:**
  - Function name `calculateDiscount` is misleading.
  - **Why it's a problem:** It returns final price, not discount.
  - **Fix:** Rename to `applyDiscount`.

Total bugs: 1
Total quality issues: 3

Snippet ID: email_validator

1. Category: BUG

- **Explanation:**
  - Uses `strpos()` incorrectly to check for "@" and "."
  - **Why it's a problem:** Fails if characters are at position 0.
  - **Fix:** Use `strpos() !== false`.

2. Category: BUG

- **Explanation:**
  - Checks only for presence of characters, not format.
  - **Why it's a problem:** Accepts invalid emails like `"com@"`.

- **Fix:** Use `filter_var($email, FILTER_VALIDATE_EMAIL)`.

3. Category: QUALITY

- **Explanation:**
  - Redundant character checks.
  - **Why it's a problem:** Harder to maintain.
  - **Fix:** Replace manual checks with `filter_var`.

Total bugs: 2
Total quality issues: 1

Snippet ID: file_write

1. Category: BUG

- **Explanation:**
  - Incorrect error check for `fopen`.
  - **Why it's a problem:** May print "File opened" even if it fails.
  - **Fix:** Use:

```
if ($handle === false) die("Failed to open file.");
```

2. Category: BUG

- **Explanation:**
  - No error check after `fwrite`.
  - **Why it's a problem:** May silently fail.
  - **Fix:** Check with:

```
if (fwrite($handle, $text) === false) die("Failed to write
to file.");
```

3. Category: BUG

- **Explanation:**
    - File is never closed.
    - **Why it's a problem:** Leads to resource leaks.
    - **Fix:** Add `fclose($handle);`.

4. Category: QUALITY

- **Explanation:**
    - Function name `write` is too generic.
    - **Why it's a problem:** Doesn't clarify if it appends/overwrites.
    - **Fix:** Rename to `overwriteFile`.

5. Category: QUALITY

- **Explanation:**
    - No validation for `$file` or `$text`.
    - **Why it's a problem:** May cause invalid behavior.
    - **Fix:** Validate both are strings and `$file` is a valid path.

Total bugs: 3
Total quality issues: 2

Snippet ID: password_strength

1. Category: BUG

- **Explanation:**
    - The function `isStrongPassword` returns `true` for weak passwords (e.g., under 8 characters or missing uppercase letters).
    - **Why it's a problem:** The return value contradicts the function's name and expected behavior.
    - **How to fix it:** Flip the logic to return `false` when the conditions aren't met:

```
return strlen($password) >= 8 && preg_match("/[A-Z]/", $password);
```

2. Category: QUALITY

- **Explanation:**
    - The current regex only checks for uppercase letters.
    - **Why it's a problem:** Strong passwords usually require a mix of uppercase, lowercase, numbers, and symbols.
    - **How to fix it:** Add more conditions to check for digits and special characters.

Total bugs: 1
Total quality issues: 1

Snippet ID: temperature_converter

1. Category: BUG

- **Explanation:**
    - The function returns a string with a unit, e.g., `" F"`.

- **Why it's a problem:** This prevents further computation with the value.
- **How to fix it:** Return just the numeric value; format the string elsewhere.

2. Category: QUALITY

- **Explanation:**
  - Function name `toFahrenheit` is vague.
  - **Why it's a problem:** Doesn't clarify that it's converting Celsius.
  - **How to fix it:** Rename to `convertCelsiusToFahrenheit`.

3. Category: QUALITY

- **Explanation:**
  - No type checks or input validation.
  - **Why it's a problem:** Passing a non-numeric value may break the function.
  - **How to fix it:** Use `is_numeric()` or typed parameters (PHP 7+).

Total bugs: 1
Total quality issues: 2

Snippet ID: triangle_type

1. Category: BUG

- **Explanation:**
  - Uses `=` instead of `==` in conditions.
  - **Why it's a problem:** Causes variable assignment instead of comparison.
  - **How to fix it:** Replace `=` with `==`.

2. Category: QUALITY

- **Explanation:**
    - No type declarations for parameters.
    - **Why it's a problem:** Accepts unintended inputs.
    - **How to fix it:** Use `float $a`, `float $b`, etc.

3. Category: QUALITY

- **Explanation:**
    - Missing `{}` around control blocks.
    - **Why it's a problem:** Can introduce bugs when adding logic.
    - **How to fix it:** Use curly braces for all control structures.

4. Category: QUALITY

- **Explanation:**
    - Return value `"Invalid"` is vague.
    - **Why it's a problem:** Doesn't clearly indicate the issue.
    - **How to fix it:** Use more descriptive messages or exceptions.

5. Category: QUALITY

- **Explanation:**
    - No documentation or comments.
    - **Why it's a problem:** Makes code harder to maintain.
    - **How to fix it:** Add docstrings and inline comments.

Total bugs: 1
Total quality issues: 4

Snippet ID: user_age_check

1. Category: BUG

- **Explanation:**
    - Uses `=` instead of `==`.
    - **Why it's a problem:** Always returns `true` due to assignment.
    - **How to fix it:** Replace with `==` or `===`.

2. Category: QUALITY

- **Explanation:**
    - Code is unnecessarily verbose.
    - **Why it's a problem:** Reduces readability.
    - **How to fix it:** Simplify to:

      ```
      return $age === 18;
      ```

Total bugs: 1
Total quality issues: 1

Snippet ID: user_login

1. Category: BUG

- **Explanation:**
    - Uses `=` instead of `==` in condition.

- **Why it's a problem:** Causes incorrect logic (always true).
- **How to fix it:** Use:

  ```
  if ($user == $validUser && $pass == $validPass)
  ```

2. Category: QUALITY

- **Explanation:**
  - Credentials are hardcoded.
  - **Why it's a problem:** Not secure or flexible.
  - **How to fix it:** Load from env vars or config.

3. Category: QUALITY

- **Explanation:**
  - Variable names `$user`, `$pass` are not descriptive.
  - **Why it's a problem:** Reduces clarity.
  - **How to fix it:** Rename to `$username` and `$password`.

Total bugs: 1
Total quality issues: 2

Snippet ID: add

1. Code quality issue

- **Explanation:**
  - The function lacks type hinting and return type declarations.

- **Why it's a problem:** Makes the code more error-prone and less readable by not explicitly defining input and output types.
- **How to fix it:** Use:

```
function add(int $a, int $b): int
```

Total bugs: 0
Total quality issues: 1

Snippet ID: average

1. Category: QUALITY

- **Explanation:**
  - The function name `calculate_average` uses an underscore.
  - **Why it's a problem:** PHP generally prefers camelCase.
  - **How to fix it:** Rename to `calculateAverage` or `calculateaverage`, depending on project style.

2. Category: QUALITY

- **Explanation:**
  - The return statement is placed inline with the `if` condition.
  - **Why it's a problem:** Hurts readability in more complex logic.
  - **How to fix it:** Use block format:

```
if (empty($numbers)) {
    return 0;
}
```

Total bugs: 0
Total quality issues: 2

Snippet ID: factorial

1. Category: BUG

- **Explanation:**
    - No handling for negative input values.
    - **Why it's a problem:** Causes infinite recursion (stack overflow).
    - **How to fix it:** Add a guard clause for negative input.

2. Category: QUALITY

- **Explanation:**
    - Uses `===` to compare with zero.
    - **Why it's a problem:** Overly strict for simple numeric comparison.
    - **How to fix it:** Use `== 0` instead of `=== 0` (optional; subjective style).

Total bugs: 1
Total quality issues: 1

Snippet ID: greet

1. Category: QUALITY

- **Explanation:**
    - Function name `greet` is too generic.
    - **Why it's a problem:** Doesn't clearly express purpose.
    - **How to fix it:** Rename to `generateGreetingMessage`.

2. Category: QUALITY

- **Explanation:**
    - No comments or documentation.
    - **Why it's a problem:** Decreases understandability and maintainability.
    - **How to fix it:** Add a PHPDoc block or descriptive comment above the function.

Total bugs: 0
Total quality issues: 2

Snippet ID: is_even

The code is correct and clean.

- **Total bugs:** 0
- **Total quality issues:** 0
- **Comment:** The function is simple, correct, and follows good practices.

Snippet ID: magic_number

1. Category: QUALITY

- **Explanation:**
    - The code uses a hard-coded "magic number" (0.85).
    - **Why it's a problem:** It lacks context and makes the code less maintainable.
    - **How to fix it:** Replace with a named constant or variable:

```
$discountRate = 0.85;
return $price * $discountRate;
```

Total bugs: 0
Total quality issues: 1

Snippet ID: mutable_default

1. Category: BUG

- **Explanation:**
    - The function `append_value` uses a reference with a default `null`.
    - **Why it's a problem:** Unexpected behavior; list does not retain values between calls.
    - **How to fix it:** Remove the default value and require explicit list input.

2. Category: QUALITY

- **Explanation:**
    - Using a reference with a default value is bad practice.
    - **Why it's a problem:** Confuses how the function should be used.
    - **How to fix it:** Require the caller to pass the list explicitly or encapsulate state in an object.

Total bugs: 1
Total quality issues: 1

Snippet ID: naming

1. Category: QUALITY

- **Explanation:**
    - Function name `dOaCt` is unclear and uses inconsistent casing.
    - **Why it's a problem:** Reduces readability and violates naming conventions.

- **How to fix it:** Rename to `addNumbers` or something descriptive using camelCase.

2. Category: QUALITY

- **Explanation:**
  - Variable names `$x`, `$y`, `$z` are not descriptive.
  - **Why it's a problem:** Makes understanding the code harder.
  - **How to fix it:** Rename to `$firstNumber`, `$secondNumber`, `$sum`.

Total bugs: 0
Total quality issues: 2

Snippet ID: nested

1. Category: BUG

- **Explanation:**
  - The function only handles the "even" case; odd and out-of-range inputs have no output.
  - **Why it's a problem:** Creates ambiguity about whether the function is working.
  - **How to fix it:** Add conditions and outputs for odd or invalid values.

2. Category: CODE QUALITY

- **Explanation:**
  - Unnecessarily nested `if` statements.
  - **Why it's a problem:** Reduces readability.
  - **How to fix it:** Use combined conditionals like:

```
if ($x > 0 && $x < 100 && $x % 2 === 0)
```

3. Category: CODE QUALITY

- **Explanation:**
    - Function name `check` is too vague.
    - **Why it's a problem:** Doesn't describe what is being checked.
    - **How to fix it:** Rename to `isEvenWithinRange`.

Total bugs: 1
Total quality issues: 2

Snippet ID: unused_var

1. Category: QUALITY

- **Explanation:**
    - Variable `$unused` is declared but never used.
    - **Why it's a problem:** Adds clutter.
    - **How to fix it:** Remove the unused variable.

2. Category: QUALITY

- **Explanation:**
    - Function `ask_number` lacks input validation.
    - **Why it's a problem:** May accept invalid input.
    - **How to fix it:** Add a loop to validate input is numeric.

3. Category: QUALITY

- **Explanation:**

- Function name `ask_number` is vague.
- **Why it's a problem:** Doesn't clarify what type of number is expected.
- **How to fix it:** Rename it to something like `get_user_input_number`.

Total bugs: 0
Total quality issues: 3

---

javascript

Snippet ID: `calculateOpenState`

1. Category: BUG

- **What the issue is**: The variable `STATES` is used without being defined or passed as a parameter.
- **Why it's a problem**: It causes a reference error, which will result in the function failing to execute properly.
- **How to fix it**: Ensure that `STATES` is defined in the scope or passed as an argument to the function.

2. Category: QUALITY

- **What the issue is**: The use of `var` for variable declarations.
- **Why it's a problem**: `var` has function scope and can lead to unintentional variable hoisting, which can cause bugs in larger and more complex codebases.
- **How to fix it**: Replace `var` with `let` or `const` depending on whether the variable will be reassigned.

3. Category: QUALITY

- **What the issue is**: The redundancy in the conditional structure.
- **Why it's a problem**: The final `else` statement is unnecessary if the previous conditions are mutually exclusive and exhaustive.
- **How to fix it**: Remove the final `else` and directly assign `rank = 3`.

4. Category: QUALITY

- **What the issue is**: The use of `else` after `if` and `else if` statements when each includes a return or full logic.
- **Why it's a problem**: It makes the code less readable and more complex than necessary.
- **How to fix it**: Reconsider the logic to simplify or document it clearly if truly needed.

Total bugs: 1
Total quality issues: 3

Snippet ID: `comparator_bug`

1. Category: BUG

- **What the issue is**: The comparator function does not handle `null` or `undefined` values in `lastAccessTime`.
- **Why it's a problem**: Subtracting these can result in `NaN`, causing incorrect sorting.
- **How to fix it**: Check that both values are defined before subtracting. Handle edge cases with consistent return values.

2. Category: QUALITY

- **What the issue is**: Use of `var` for variable declarations.
- **Why it's a problem**: `var` can lead to hoisting issues, reducing clarity and safety.

- **How to fix it**: Use `let` or `const` depending on reassignment needs.

3. Category: QUALITY

- **What the issue is**: The else-if ladder is overly complex.
- **Why it's a problem**: Harder to read and maintain as conditions grow.
- **How to fix it**: Simplify by checking undefined/null first, then compare.

4. Category: QUALITY

- **What the issue is**: The function name `comparator` is too generic.
- **Why it's a problem**: It doesn't clearly express the sorting logic.
- **How to fix it**: Rename to something like `compareByLastAccessTime`.

Total bugs: 1
Total quality issues: 3

Snippet ID: `filteredArray_bug`

1. Category: BUG

- **What the issue is**: Modifying the array while iterating causes skipped elements.
- **Why it's a problem**: `splice` changes the length and shifts indices.
- **How to fix it**: Iterate backwards to avoid index shifting issues.

2. Category: BUG

- **What the issue is**: Function returns unexpected results due to `indexOf` on sub-arrays.
- **Why it's a problem**: `indexOf` doesn't check deep equality, so sub-arrays aren't properly filtered.

- **How to fix it**: Use a loop or `Array.prototype.some()` to check for elements in sub-arrays.

3. Category: QUALITY

- **What the issue is**: The variable name `elemCheck` is not descriptive.
- **Why it's a problem**: Non-descriptive names reduce readability and maintainability.
- **How to fix it**: Rename it to something clearer, like `elementIndex`.

4. Category: QUALITY

- **What the issue is**: Unnecessary use of spread operator to copy the array at the start.
- **Why it's a problem**: Adds extra computation with no benefit.
- **How to fix it**: Use `filter` or build the array during iteration instead.

Total bugs: 2
Total quality issues: 2

Snippet ID: `Greeter_bug`

1. Category: BUG

- **What the issue is**: Incorrect assignment inside `if (greeting = undefined)`.
- **Why it's a problem**: `=` is an assignment, not a comparison, leading to logic bugs.
- **How to fix it**: Use `===` to check equality: `if (greeting === undefined)`.

2. Category: QUALITY

- **What the issue is**: Redundant reassignment of `this.name` and `this.greeting` to local constants.

- **Why it's a problem**: Adds unnecessary code.
- **How to fix it**: Use `this.name` and `this.greeting` directly.

3. Category: BUG

- **What the issue is**: Template string uses double quotes instead of backticks.
- **Why it's a problem**: It results in incorrect string interpolation.
- **How to fix it**: Use backticks: `return `${greeting}, ${name}!`;`

4. Category: BUG

- **What the issue is**: `Greeter` instance created with one argument, but constructor expects two.
- **Why it's a problem**: This may leave `greeting` as `undefined`.
- **How to fix it**: Pass two arguments or define defaults in the constructor.

5. Category: QUALITY

- **What the issue is**: Lack of default values in constructor.
- **Why it's a problem**: Reduces clarity and robustness of instantiation.
- **How to fix it**: Use `constructor(name = "Anonymous", greeting = "Hello")`.

Total bugs: 3
Total quality issues: 2

Snippet ID: `hideAll_bug`

1. Category: BUG

- **What the issue is**: Assignment `config = name;` inside an unrelated condition.

- **Why it's a problem**: Overwrites `config` unexpectedly, likely a logic error.
- **How to fix it**: Clarify or remove this line depending on the intended behavior.

2. Category: QUALITY

- **What the issue is**: Function and parameter names (`hideAll`, `name`) are unclear.
- **Why it's a problem**: Poor naming obscures intent and behavior.
- **How to fix it**: Rename function and parameters for clarity, e.g. `applyTransitionWithConfig`.

3. Category: QUALITY

- **What the issue is**: Use of `name.push` implies `name` might be an array.
- **Why it's a problem**: Inconsistent type expectations confuse maintenance.
- **How to fix it**: Clearly define and document types or use TypeScript.

Total bugs: 1
Total quality issues: 2

Snippet ID: `multiple-of-3-and-5_bug`

1. Category: BUG

- **What the issue is**: The `sum` variable is defined outside the function and is being accumulated across multiple calls.
- **Why it's a problem**: It causes incorrect results when the function is reused, since `sum` keeps increasing instead of resetting.
- **How to fix it**: Move the `sum` variable declaration inside the function to reset it each time it's called.

2. Category: QUALITY

- **What the issue is**: The condition `(i % 3 === 0 && i % 5 === 0)` is redundant.
- **Why it's a problem**: It complicates the condition unnecessarily; `(i % 3 === 0 || i % 5 === 0)` is sufficient.
- **How to fix it**: Remove the redundant condition from the `if` statement.

3. Category: QUALITY

- **What the issue is**: Poor naming of the function and variable; `multiplesOf3and5` is unclear.
- **Why it's a problem**: It reduces readability and might mislead developers into thinking it returns a list.
- **How to fix it**: Rename to something like `sumOfMultiplesOf3And5`.

Total bugs: 1
Total quality issues: 2

Snippet ID: `palindrome_bug`

1. Category: BUG

- **What the issue is**: Incorrect regular expression in the `split` method: `/\W*\d*[_]*/`.
- **Why it's a problem**: It breaks the string unnecessarily and doesn't correctly remove non-alphanumerics.
- **How to fix it**: Use `/[\W_]/` with `join('').toLowerCase()` to clean the string properly.

2. Category: QUALITY

- **What the issue is**: Use of bitwise operator `>>` for flooring.

- **Why it's a problem**: It's an implicit trick that's unclear to readers.
- **How to fix it**: Replace with `Math.floor(str.length / 2)` for clarity.

3. Category: QUALITY

- **What the issue is**: Use of `console.log` for intermediate output.
- **Why it's a problem**: It clutters output and impacts performance.
- **How to fix it**: Remove or comment out debug logging once the function works.

Total bugs: 1
Total quality issues: 2

Snippet ID: `prime-summation_bug`

1. Category: BUG

- **What the issue is**: The `isPrime` function incorrectly uses `number % e` inside `every`.
- **Why it's a problem**: Non-zero results are treated as truthy, misidentifying non-primes.
- **How to fix it**: Change to `number % e !== 0`.

2. Category: BUG

- **What the issue is**: The loop is hardcoded to stop after 10001 primes.
- **Why it's a problem**: The function should respect the `n` parameter instead.
- **How to fix it**: Use `for (let i = 3; i < n; i++)` to stay within the given limit.

3. Category: QUALITY

- **What the issue is**: The variable `vecOfPrimes` is poorly named.

- **Why it's a problem**: `vec` is not descriptive; naming should be intuitive.
- **How to fix it**: Rename to `primes` or `primeNumbers`.

4. Category: QUALITY

- **What the issue is**: Use of `isSmallEnough` helper is redundant.
- **Why it's a problem**: It adds unnecessary abstraction for a simple check.
- **How to fix it**: Inline the logic with `vecOfPrimes.filter(value => value < n)`.

5. Category: QUALITY

- **What the issue is**: Misleading comment claims recursion is used.
- **Why it's a problem**: This creates confusion for maintainers.
- **How to fix it**: Correct or remove the comment.

Total bugs: 2
Total quality issues: 3

Snippet ID: `titlecase_bug`

1. Category: BUG

- **What the issue is**: `toUpperCase` is not applied correctly to the first character.
- **Why it's a problem**: It fails to capitalize the first letter as intended.
- **How to fix it**: Use `temp = temp.charAt(0).toUpperCase() + temp.slice(1);`.

2. Category: BUG

- **What the issue is**: `arr[i].replace(arr[i], temp);` does not modify `arr[i]`.

- **Why it's a problem**: `replace` returns a string but doesn't alter the original.
- **How to fix it**: Assign `temp` back to `arr[i]` directly.

3. Category: QUALITY

- **What the issue is**: Misuse of `replace` in the update line.
- **Why it's a problem**: It's redundant and misleading.
- **How to fix it**: Remove the line and directly assign `arr[i] = temp;`.

4. Category: QUALITY

- **What the issue is**: Vague variable name `temp`.
- **Why it's a problem**: It doesn't clarify the variable's purpose.
- **How to fix it**: Rename to something like `capitalizedWord`.

Total bugs: 2
Total quality issues: 2

Snippet ID: `wtfs_bug`

1. Category: BUG

- **What the issue is**: The `notifier.update` condition is not safely checked.
- **Why it's a problem**: If `notifier.update` is `null` or `undefined`, accessing `.latest` will throw.
- **How to fix it**: Use `if (notifier.update && notifier.update.latest)`.

2. Category: QUALITY

- **What the issue is**: Poor use of logging and lack of error handling.

- **Why it's a problem**: Errors are logged with `console.log` instead of `console.error`, and stream errors aren't handled.
- **How to fix it**: Use `console.error` and add `error` event handling on the stream.

3. Category: QUALITY

- **What the issue is**: Redundant setting of default environment variables.
- **Why it's a problem**: Duplicate logic adds noise.
- **How to fix it**: Use a helper function or simplify assignments.

4. Category: QUALITY

- **What the issue is**: Hard-coded magic strings for file paths.
- **Why it's a problem**: Harder to update and prone to errors.
- **How to fix it**: Define constants for file paths.

5. Category: QUALITY

- **What the issue is**: Unclear variable names like `l` and `i`.
- **Why it's a problem**: They reduce clarity.
- **How to fix it**: Use descriptive names like `languagePart` and `index`.

Total bugs: 1
Total quality issues: 4

---

java

Snippet ID: `DEPTH_FIRST_SEARCH`

1. Category: BUG

- **What the issue is**: The `Node` class may not override `equals` and `hashCode`.
- **Why it's a problem**: Without these, the `HashSet` cannot correctly track visited nodes, leading to repeated visits.
- **How to fix it**: Override `equals` and `hashCode` in `Node` using its unique attributes (e.g., ID).

2. Category: QUALITY

- **What the issue is**: Naming conventions are not followed (`DEPTH_FIRST_SEARCH`, `nodesvisited`).
- **Why it's a problem**: It reduces readability and goes against Java standards.
- **How to fix it**: Rename to `DepthFirstSearch`, `nodesVisited`, etc.

3. Category: QUALITY

- **What the issue is**: Inline class definition for `Search` inside a method.
- **Why it's a problem**: It reduces readability and reusability.
- **How to fix it**: Move `Search` to a private static nested or top-level class.

4. Category: QUALITY

- **What the issue is**: Use of `else` after return statements.
- **Why it's a problem**: Adds unnecessary nesting.
- **How to fix it**: Remove `else` and allow direct returns.

Total bugs: 1
Total quality issues: 3

Snippet ID: `DETECT_CYCLE`

1. Category: BUG

- **What the issue is**: May throw `NullPointerException` on `hare.getSuccessor().getSuccessor()`.
- **Why it's a problem**: It crashes if `hare.getSuccessor()` is null.
- **How to fix it**: Add null checks before access.

2. Category: BUG

- **What the issue is**: Input `node` might be `null`, causing `NullPointerException`.
- **Why it's a problem**: The method doesn't handle a `null` input.
- **How to fix it**: Add `if (node == null) return false;`.

3. Category: QUALITY

- **What the issue is**: Use of `while(true)` loop without clear termination condition.
- **Why it's a problem**: Hurts readability and maintainability.
- **How to fix it**: Use a meaningful condition like `while (hare != null && hare.getSuccessor() != null)`.

4. Category: QUALITY

- **What the issue is**: Class name `DETECT_CYCLE` uses all caps.
- **Why it's a problem**: Violates Java naming conventions.
- **How to fix it**: Rename to `DetectCycle`.

Total bugs: 2
Total quality issues: 2

Snippet ID: `FIND_IN_SORTED`

1. Category: BUG

- **What the issue is**: Incorrect recursion using `mid` as the next start/end.
- **Why it's a problem**: Can lead to infinite recursion or miss values.
- **How to fix it**: Use `mid + 1` or `mid - 1` accordingly.

2. Category: BUG

- **What the issue is**: Incorrect base case: `start == end`.
- **Why it's a problem**: May skip valid final comparisons.
- **How to fix it**: Use `start > end` as the termination condition.

3. Category: QUALITY

- **What the issue is**: Poor naming and comments.
- **Why it's a problem**: Violates Java naming conventions; comments are unclear.
- **How to fix it**: Use `FindInSorted`, `findInSorted`, and update/remove vague comments.

4. Category: QUALITY

- **What the issue is**: No validation for `null` array input.
- **Why it's a problem**: Causes `NullPointerException`.
- **How to fix it**: Add a check for `null` input.

5. Category: QUALITY

- **What the issue is**: Irrelevant or outdated comments at class level.

- **Why it's a problem**: Creates confusion.
- **How to fix it**: Update or delete the comment.

Total bugs: 2
Total quality issues: 3

Snippet ID: `IS_VALID_PARENTHESIZATION`

1. Category: BUG

- **What the issue is**: Always returns `true` if parentheses are balanced up to a point.
- **Why it's a problem**: It doesn't check if all opened parentheses are closed.
- **How to fix it**: Ensure `depth == 0` before returning `true`.

2. Category: QUALITY

- **What the issue is**: Class name in all caps.
- **Why it's a problem**: Not consistent with Java CamelCase conventions.
- **How to fix it**: Rename to `IsValidParenthesization`.

3. Category: QUALITY

- **What the issue is**: Method name uses underscores.
- **Why it's a problem**: Doesn't follow Java camelCase method naming.
- **How to fix it**: Rename to `isValidParenthesization`.

4. Category: QUALITY

- **What the issue is**: Uses `Character` instead of `char`.
- **Why it's a problem**: Adds unnecessary boxing overhead.

- **How to fix it**: Use primitive `char` instead.

5. Category: QUALITY

- **What the issue is**: Contains template-generated comment block.
- **Why it's a problem**: Irrelevant and misleading.
- **How to fix it**: Remove the block starting with `/* To change this template... */`.

6. Category: QUALITY

- **What the issue is**: Uses `@author` in Javadoc.
- **Why it's a problem**: Can become outdated and distract from code.
- **How to fix it**: Rely on version control instead.

Total bugs: 1
Total quality issues: 5

Snippet ID: KTH

1. Category: BUG

- **What the issue is**: Recursive calls do not adjust `k` for sublists.
- **Why it's a problem**: It returns incorrect results in certain `k` ranges.
- **How to fix it**: Call `kth(above, k - num_lessoreq)` instead of `kth(above, k)`.

2. Category: QUALITY

- **What the issue is**: No check for empty input list.
- **Why it's a problem**: Accessing `arr.get(0)` on empty list throws error.

- **How to fix it**: Add null/empty check before using list.

3. Category: QUALITY

- **What the issue is**: Preallocating size for `below` and `above` unnecessarily.
- **Why it's a problem**: Wasteful memory allocation.
- **How to fix it**: Let them use default capacity.

4. Category: QUALITY

- **What the issue is**: Empty documentation comment block.
- **Why it's a problem**: Adds clutter with no value.
- **How to fix it**: Remove or populate it with meaningful info.

5. Category: QUALITY

- **What the issue is**: Vague variable names `num_less`, `num_lessoreq`.
- **Why it's a problem**: Reduces readability.
- **How to fix it**: Use names like `numLessThanPivot`, `numLessOrEqualToPivot`.

Total bugs: 1
Total quality issues: 4

Snippet ID: `LEVENSHTEIN`

1. Category: BUG

- **What the issue is**: The recursive case incorrectly adds `1` for equal characters.

- **Why it's a problem**: It miscalculates the Levenshtein distance by counting a change where there isn't one.
- **How to fix it**: Return the recursive call without adding `1` when characters are equal.

2. Category: BUG

- **What the issue is**: The base case doesn't handle both strings being empty.
- **Why it's a problem**: May produce incorrect results.
- **How to fix it**: Add a check for both strings being empty: `if (source.isEmpty() && target.isEmpty()) return 0;`.

3. Category: QUALITY

- **What the issue is**: Code indentation is inconsistent.
- **Why it's a problem**: Makes the code harder to read and maintain.
- **How to fix it**: Reformat code for clarity and consistency.

4. Category: QUALITY

- **What the issue is**: Class name is in uppercase (`LEVENSHTEIN`).
- **Why it's a problem**: Violates Java naming conventions.
- **How to fix it**: Rename the class to `Levenshtein`.

5. Category: QUALITY

- **What the issue is**: Recursive approach lacks optimization.
- **Why it's a problem**: Causes stack overflows on large inputs.
- **How to fix it**: Use dynamic programming with a matrix to store subproblem results.

6. Category: QUALITY

- **What the issue is**: Top comment is a boilerplate template.
- **Why it's a problem**: It adds noise without value.
- **How to fix it**: Remove or replace with a meaningful description.

Total bugs: 2
Total quality issues: 4

Snippet ID: `LIS`

1. Category: BUG

- **What the issue is**: Possible `NullPointerException` from `ends.get(length+1)`.
- **Why it's a problem**: May crash the program.
- **How to fix it**: Add checks to validate access to the array.

2. Category: BUG

- **What the issue is**: Incorrect logic updating `ends` map.
- **Why it's a problem**: Breaks subsequence tracking.
- **How to fix it**: Correctly update only when appropriate.

3. Category: QUALITY

- **What the issue is**: Arbitrary initial capacities for `HashMap` and `ArrayList`.
- **Why it's a problem**: May waste memory.
- **How to fix it**: Let them grow dynamically.

4. Category: QUALITY

- **What the issue is**: Poor variable names like `arr`, `val`.

- **Why it's a problem**: Reduces clarity.
- **How to fix it**: Use descriptive names like `inputArray`.

5. Category: QUALITY

- **What the issue is**: Suboptimal use of loop variables.
- **Why it's a problem**: Makes code harder to follow.
- **How to fix it**: Consider enhanced loops or clearer iteration.

6. Category: QUALITY

- **What the issue is**: Irrelevant template comment block.
- **Why it's a problem**: Adds clutter.
- **How to fix it**: Delete or replace with useful documentation.

Total bugs: 2
Total quality issues: 4

Snippet ID: `POSSIBLE_CHANGE`

1. Category: BUG

- **What the issue is**: Accessing `coins[0]` when array is empty.
- **Why it's a problem**: Causes `ArrayIndexOutOfBoundsException`.
- **How to fix it**: Add a check for empty array and return `0`.

2. Category: BUG

- **What the issue is**: Missing base case logic for recursion.
- **Why it's a problem**: Leads to incorrect or infinite recursion.

- **How to fix it**: Add condition to return `0` when coins are empty but total > 0.

3. Category: QUALITY

- **What the issue is**: Class name `POSSIBLE_CHANGE` is not in CamelCase.
- **Why it's a problem**: Breaks Java naming conventions.
- **How to fix it**: Rename to `PossibleChange`.

4. Category: QUALITY

- **What the issue is**: Method name `possible_change` is not camelCase.
- **Why it's a problem**: Reduces consistency with Java standards.
- **How to fix it**: Rename to `possibleChange`.

5. Category: QUALITY

- **What the issue is**: Top comment is a default IDE template.
- **Why it's a problem**: Adds no value.
- **How to fix it**: Remove or replace with a proper description.

Total bugs: 2
Total quality issues: 3

Snippet ID: SQRT

1. Category: BUG

- **What the issue is**: Incorrect convergence check in `while` loop.
- **Why it's a problem**: Leads to incorrect termination condition.
- **How to fix it**: Use `Math.abs(approx * approx - x) > epsilon` instead.

2. Category: QUALITY

- **What the issue is**: Presence of IDE template comment.
- **Why it's a problem**: Adds noise without value.
- **How to fix it**: Remove the comment.

3. Category: QUALITY

- **What the issue is**: Class name `SQRT` is uppercase.
- **Why it's a problem**: Not in CamelCase.
- **How to fix it**: Rename to `SqrtCalculator`.

4. Category: QUALITY

- **What the issue is**: No method-level comment for `sqrt`.
- **Why it's a problem**: Reduces clarity and documentation quality.
- **How to fix it**: Add a description explaining parameters and purpose.

Total bugs: 1
Total quality issues: 3

Snippet ID: `WeightedEdge`

1. Category: BUG

- **What the issue is**: Unsafe integer subtraction in `compareTo`.
- **Why it's a problem**: Can cause overflow and incorrect ordering.
- **How to fix it**: Use `Integer.compare(this.weight, compareNode.weight)`.

2. Category: QUALITY

- **What the issue is**: Manual toggling comments for sort order.
- **Why it's a problem**: Prone to user error and poor maintainability.
- **How to fix it**: Use a boolean flag to control sort direction.

3. Category: QUALITY

- **What the issue is**: Parameter name `compareNode` is misleading.
- **Why it's a problem**: It compares edges, not nodes.
- **How to fix it**: Rename to `compareEdge`.

4. Category: QUALITY

- **What the issue is**: Fields `node1`, `node2`, `weight` are public.
- **Why it's a problem**: Violates encapsulation principles.
- **How to fix it**: Make them `private` and expose via getters.

Total bugs: 1
Total quality issues: 3

Snippet ID: `Addition`

1. Category: QUALITY

- **What the issue is**: The class name `JavaClean01_Addition` does not follow standard Java naming conventions and includes unnecessary prefixes.
- **Why it's a problem**: It reduces readability and maintainability.
- **How to fix it**: Rename the class to `Addition`.

2. Category: QUALITY

- **What the issue is**: The class lacks a private constructor to prevent instantiation.
- **Why it's a problem**: Utility classes with only static methods should not be instantiated.
- **How to fix it**: Add `private Addition() {}` to prevent instantiation.

Total bugs: 0
Total quality issues: 2

Snippet ID: `Average`

1. Category: BUG

- **What the issue is**: The method returns `0` for an empty array.
- **Why it's a problem**: Misleading result implies a valid average.
- **How to fix it**: Throw an exception or return `Double.NaN`.

2. Category: QUALITY

- **What the issue is**: Class name `JavaClean05_Average` is non-standard.
- **Why it's a problem**: Makes code less readable and violates naming conventions.
- **How to fix it**: Rename the class to `AverageCalculator`.

3. Category: QUALITY

- **What the issue is**: The method lacks comments or documentation.
- **Why it's a problem**: Makes the code harder to understand or maintain.
- **How to fix it**: Add comments explaining the method's purpose and parameters.

Total bugs: 1
Total quality issues: 2

Snippet ID: `Customer`

1. Category: BUG

- **What the issue is**: `joiningDate` is always set to the current date.
- **Why it's a problem**: Prevents representing past dates and may not reflect intended data.
- **How to fix it**: Allow setting `joiningDate` via constructor or setter method.

2. Category: QUALITY

- **What the issue is**: Uses `java.util.Date`, which is outdated.
- **Why it's a problem**: `Date` is error-prone and less expressive.
- **How to fix it**: Use `java.time.LocalDate`.

3. Category: QUALITY

- **What the issue is**: Package name implies JPA entity, but class does not conform.
- **Why it's a problem**: Could mislead about the class's purpose.
- **How to fix it**: Move to a more appropriate package or adapt to typical entity conventions.

Total bugs: 1
Total quality issues: 2

Snippet ID: `ExposeState`

1. Category: BUG

- **What the issue is**: The method `writerConfig()` is called but not defined.
- **Why it's a problem**: Causes a compilation error.

- **How to fix it**: Define `writerConfig()` or replace it with valid logic.

2. Category: QUALITY

- **What the issue is**: `getConfig()` exposes internal `HashMap`.
- **Why it's a problem**: Violates encapsulation and allows external modification.
- **How to fix it**: Return an unmodifiable map or a deep copy.

3. Category: QUALITY

- **What the issue is**: Class name `JavaBad05_ExposeState` is non-standard.
- **Why it's a problem**: Makes the class less readable and harder to locate meaning.
- **How to fix it**: Rename to `ConfigManager`.

4. Category: QUALITY

- **What the issue is**: Uses concrete type `HashMap` for field.
- **Why it's a problem**: Reduces flexibility and abstraction.
- **How to fix it**: Declare field as `Map<String, String>`.

Total bugs: 1
Total quality issues: 3

Snippet ID: `Factorial`

1. Category: BUG

- **What the issue is**: Does not handle negative integers.
- **Why it's a problem**: Causes infinite recursion and stack overflow.
- **How to fix it**: Add validation to reject or handle negative inputs.

2. Category: QUALITY

- **What the issue is**: No documentation or handling of edge cases.
- **Why it's a problem**: Users may not understand expected inputs.
- **How to fix it**: Add JavaDoc or comments explaining behavior and assumptions.

Total bugs: 1
Total quality issues: 1

Snippet ID: `Greeting`

1. Category: QUALITY

- **What the issue is**: The class name `JavaClean03_Greeting` contains a numeric identifier ("03").
- **Why it's a problem**: It's unconventional and can confuse readers, especially when versioning should be handled by a VCS.
- **How to fix it**: Rename the class to `Greeting` or `GreetingGenerator`.

2. Category: QUALITY

- **What the issue is**: The `greet` method is static.
- **Why it's a problem**: Static methods limit flexibility and hinder object-oriented design (e.g., inheritance).
- **How to fix it**: Convert `greet` to an instance method unless utility behavior is required.

Total bugs: 0
Total quality issues: 2

Snippet ID: `IsEven`

1. Category: QUALITY

- **What the issue is**: The class name `JavaClean04_IsEven` does not follow Java naming conventions.
- **Why it's a problem**: Java prefers `PascalCase` without underscores or numbers unless meaningful.
- **How to fix it**: Rename the class to something like `EvenChecker`.

Total bugs: 0
Total quality issues: 1

Snippet ID: `MagicNumber`

1. Category: BUG

- **What the issue is**: The method uses a hardcoded 15% discount (0.85).
- **Why it's a problem**: It lacks flexibility and assumes the discount rate never changes.
- **How to fix it**: Pass the discount rate as a parameter.

2. Category: QUALITY

- **What the issue is**: Uses a magic number (`0.85`) without explanation.
- **Why it's a problem**: Makes code less readable and harder to maintain.
- **How to fix it**: Define a constant like `DEFAULT_DISCOUNT_RATE`.

Total bugs: 1
Total quality issues: 1

Snippet ID: `Nesting`

1. Category: BUG

- **What the issue is**: The method only prints `"Even"` for a narrow set of conditions and says nothing otherwise.
- **Why it's a problem**: Omits feedback for most inputs, which can be misleading.
- **How to fix it**: Add `else` cases for other scenarios like odd, negative, or out-of-range values.

2. Category: QUALITY

- **What the issue is**: Deeply nested `if-else` statements.
- **Why it's a problem**: Hurts readability and maintainability.
- **How to fix it**: Flatten the logic using compound conditions.

Total bugs: 1
Total quality issues: 1

Snippet ID: `UnusedVariable`

1. Category: QUALITY

- **What the issue is**: Declares a variable (`unused`) that is never used.
- **Why it's a problem**: Adds clutter and can confuse maintainers.
- **How to fix it**: Remove the unused variable.

Total bugs: 0
Total quality issues: 1

C++

Snippet ID: `const`

1. Category: BUG

- **What the issue is**: `whoami()` is not marked with `override` in struct `D`.
- **Why it's a problem**: Could result in unintended behavior if the base signature changes.
- **How to fix it**: Add `override` to `void whoami()`.

2. Category: QUALITY

- **What the issue is**: Use of `using namespace std;`.
- **Why it's a problem**: Pollutes global namespace.
- **How to fix it**: Remove it and use `std::` prefixes.

3. Category: QUALITY

- **What the issue is**: Inconsistent formatting of function definitions.
- **Why it's a problem**: Hurts readability.
- **How to fix it**: Standardize formatting across all methods.

Total bugs: 1
Total quality issues: 2

Snippet ID: `fun`

1. Category: BUG

- **What the issue is**: Division by zero (`x /= a`) since `a` is set to 0.

- **Why it's a problem**: Causes a runtime crash.
- **How to fix it**: Ensure `a` is non-zero before division.

2. Category: BUG

- **What the issue is**: Array out-of-bounds access (`buffer[10]`).
- **Why it's a problem**: Leads to undefined behavior.
- **How to fix it**: Use indices within the valid range (0–9).

3. Category: BUG

- **What the issue is**: Uninitialized variable `a` used.
- **Why it's a problem**: Causes undefined behavior.
- **How to fix it**: Initialize `a` before using it.

4. Category: QUALITY

- **What the issue is**: Non-descriptive function name `fun`.
- **Why it's a problem**: Makes code harder to understand.
- **How to fix it**: Rename to something descriptive like `processInput`.

5. Category: QUALITY

- **What the issue is**: Empty `if` block (`if (x != 42) { }`).
- **Why it's a problem**: Adds useless code.
- **How to fix it**: Remove or implement meaningful logic.

6. Category: QUALITY

- **What the issue is**: Mixed variable declarations.

- **Why it's a problem**: Reduces readability.
- **How to fix it**: Group variables logically and close to usage.

Total bugs: 3
Total quality issues: 3

Snippet ID: `grading`

1. Category: BUG

- **What the issue is**: `while(theMark >= 0. && theMark <= maxMark)` logic is inverted.
- **Why it's a problem**: Accepts incorrect marks and rejects correct ones.
- **How to fix it**: Use `while(theMark < 0. || theMark > maxMark)`.

2. Category: BUG

- **What the issue is**: Incorrect formula for grade calculation.
- **Why it's a problem**: Produces inaccurate final marks.
- **How to fix it**: Redefine the grading formula clearly.

3. Category: BUG

- **What the issue is**: Syntax error: `Return` instead of `return`.
- **Why it's a problem**: Code won't compile.
- **How to fix it**: Correct to `return (mark + 0.5);`.

4. Category: QUALITY

- **What the issue is**: Use of `using namespace std;`.

- **Why it's a problem**: Can cause name conflicts.
- **How to fix it**: Use explicit `std::` prefixes.

5. Category: QUALITY

- **What the issue is**: Poor variable and function naming.
- **Why it's a problem**: Reduces clarity.
- **How to fix it**: Use names like `finalExamScore`, `calculateGrade`.

6. Category: QUALITY

- **What the issue is**: Missing parameter type in `getMark(maxMark)`.
- **Why it's a problem**: Causes a compilation error.
- **How to fix it**: Use `double getMark(double maxMark)`.

7. Category: QUALITY

- **What the issue is**: Potential loss of precision from double to int.
- **Why it's a problem**: Might not reflect intended behavior.
- **How to fix it**: Use consistent types or document the conversion.

Total bugs: 3
Total quality issues: 4

Snippet ID: `math`

1. Category: BUG

- **What the issue is**: `distance()` returns `int` instead of `double`.
- **Why it's a problem**: Rounds results and reduces accuracy.

- **How to fix it**: Return `double` without casting.

2. Category: BUG

- **What the issue is**: Loops depend on flawed distance logic.
- **Why it's a problem**: Causes incorrect results.
- **How to fix it**: Fix `distance()` and validate logic again.

3. Category: QUALITY

- **What the issue is**: Poor variable names like `s`, `s1`, `k`.
- **Why it's a problem**: Hard to understand code.
- **How to fix it**: Rename to descriptive terms like `minDist`.

4. Category: QUALITY

- **What the issue is**: Output format lacks spacing.
- **Why it's a problem**: Output is hard to read.
- **How to fix it**: Add a space or newline between coordinates.

5. Category: QUALITY

- **What the issue is**: Use of `using namespace std;`.
- **Why it's a problem**: Leads to namespace pollution.
- **How to fix it**: Remove and use `std::` prefixes.

6. Category: QUALITY

- **What the issue is**: No input validation.
- **Why it's a problem**: Causes runtime errors.

- **How to fix it**: Check inputs for validity.

Total bugs: 2
Total quality issues: 4

Snippet ID: `Nonzerosample`

1. Category: BUG

- **What the issue is**: `intnxt` is declared as a `static const int` with a value of `0`.
- **Why it's a problem**: The method `nxtNonZero()` assigns this constant to `value`, resulting in an infinite loop since `value == 0` is always true.
- **How to fix it**: Change the logic to modify `value` each iteration to become non-zero, for example by generating or retrieving a new value dynamically.

2. Category: QUALITY

- **What the issue is**: The code uses `new` and `delete` to manage a `NonZeroSample` object.
- **Why it's a problem**: Manual memory management increases complexity and risk of leaks.
- **How to fix it**: Use stack allocation instead: replace `new` with a local variable and remove `delete`.

3. Category: QUALITY

- **What the issue is**: Unnecessary use of `this->` to access member `intnxt`.
- **Why it's a problem**: Adds clutter unless resolving a naming conflict.
- **How to fix it**: Access `intnxt` directly without `this->`.

4. Category: QUALITY

- **What the issue is**: The variable name `intnxt` is not descriptive.
- **Why it's a problem**: It obscures the purpose of the variable.
- **How to fix it**: Rename to a more meaningful name that reflects its role.

Total bugs: 1
Total quality issues: 3

Snippet ID: `references`

1. Category: BUG

- **What the issue is**: After `p` is deleted, `ppr` and `prr` become dangling references.
- **Why it's a problem**: Accessing dangling references results in undefined behavior.
- **How to fix it**: Set `p` to `nullptr` after deletion and avoid using `ppr` or `prr` afterward.

2. Category: BUG

- **What the issue is**: Printing `ar` and `ra` directly only prints the address, not the values.
- **Why it's a problem**: Misleads the output and doesn't show intended data.
- **How to fix it**: Iterate through the arrays and print elements individually.

3. Category: QUALITY

- **What the issue is**: Use of raw pointers with `new` and `delete`.
- **Why it's a problem**: Prone to memory leaks and complex error-prone memory handling.
- **How to fix it**: Use smart pointers like `std::unique_ptr` or `std::shared_ptr`.

4. Category: QUALITY

- **What the issue is**: Variable names like `r1`, `ppr`, `prr` are non-descriptive.
- **Why it's a problem**: Makes code harder to read and maintain.
- **How to fix it**: Rename variables to clearly convey their purpose.

5. Category: QUALITY

- **What the issue is**: Array `ar` is uninitialized.
- **Why it's a problem**: May contain garbage values leading to undefined behavior.
- **How to fix it**: Explicitly initialize the array.

Total bugs: 2
Total quality issues: 3

Snippet ID: `static_analysis`

1. Category: BUG

- **What the issue is**: `delete[]` used instead of `delete` for a single element.
- **Why it's a problem**: Leads to undefined behavior.
- **How to fix it**: Replace `delete[] buff;` with `delete buff;`.

2. Category: BUG

- **What the issue is**: Memory allocated in `memleak()` is never freed.
- **Why it's a problem**: Causes memory leaks.
- **How to fix it**: Add corresponding `delete`/`free` calls before exit.

3. Category: BUG

- **What the issue is**: Array out-of-bounds access (`array[11]`).
- **Why it's a problem**: Leads to undefined behavior.
- **How to fix it**: Keep index within bounds (e.g., `array[9]`).

4. Category: BUG

- **What the issue is**: Dynamic array accessed out of bounds.
- **Why it's a problem**: Same as above—undefined behavior.
- **How to fix it**: Use valid indices within allocation size.

5. Category: BUG

- **What the issue is**: Dereferencing a NULL pointer.
- **Why it's a problem**: Crashes the program.
- **How to fix it**: Validate the pointer before dereferencing.

6. Category: BUG

- **What the issue is**: `goto` causes dereferencing of a potentially NULL pointer.
- **Why it's a problem**: Breaks control flow safety.
- **How to fix it**: Refactor logic to eliminate unsafe `goto`.

7. Category: BUG

- **What the issue is**: Mismatched memory management (e.g., `free` on `new`).
- **Why it's a problem**: Causes undefined behavior.
- **How to fix it**: Match `new` with `delete` and `malloc` with `free`.

Total bugs: 7
Total quality issues: 0

Snippet ID: `template-comparison`

1. Category: BUG

- **What the issue is**: `operator<` always returns `true`.
- **Why it's a problem**: Breaks comparison logic.
- **How to fix it**: Implement meaningful comparison logic.

2. Category: BUG

- **What the issue is**: Function `f<x>(1);` has ambiguous meaning under `#ifdef TEMPLATE`.
- **Why it's a problem**: Causes confusion and logical inconsistency.
- **How to fix it**: Avoid conflicting names across branches.

3. Category: QUALITY

- **What the issue is**: `typedef int x;` conflicts with variable name `x`.
- **Why it's a problem**: Confuses meaning of `x`.
- **How to fix it**: Use a more descriptive alias name.

4. Category: QUALITY

- **What the issue is**: Overuse of preprocessor directives alters code behavior.
- **Why it's a problem**: Obfuscates logic and increases maintenance burden.
- **How to fix it**: Use clearer configuration management methods.

5. Category: QUALITY

- **What the issue is**: `operator<` passes parameters by value.

- **Why it's a problem**: Reduces efficiency.
- **How to fix it**: Use `const A&` instead of `A`.

Total bugs: 2
Total quality issues: 3

Snippet ID: `vector`

1. Category: BUG

- **What the issue is**: Modifying a vector while iterating over it with an iterator.
- **Why it's a problem**: Causes iterator invalidation and undefined behavior.
- **How to fix it**: Collect changes separately and apply after iteration.

2. Category: QUALITY

- **What the issue is**: Use of `using namespace std;`.
- **Why it's a problem**: Reduces clarity and may cause naming conflicts.
- **How to fix it**: Use `std::` prefixes.

3. Category: QUALITY

- **What the issue is**: Comment claims STL is unsafe.
- **Why it's a problem**: Misleading and inaccurate.
- **How to fix it**: Reword to explain iterator invalidation clearly.

Total bugs: 1
Total quality issues: 2

Snippet ID: `virtual`

1. Category: QUALITY

- **What the issue is**: Typos in comments (“wether”, “fucntion”).
- **Why it’s a problem**: Hurts readability and professionalism.
- **How to fix it**: Correct to “whether” and “function”.

2. Category: QUALITY

- **What the issue is**: `using namespace std;` used globally.
- **Why it’s a problem**: Leads to namespace pollution.
- **How to fix it**: Remove and use `std::` prefixes.

3. Category: BUG

- **What the issue is**: `f()` in derived classes not marked `override`.
- **Why it’s a problem**: Reduces clarity and error checking.
- **How to fix it**: Add `override` to `f()` in `Middle` and `Derived`.

4. Category: QUALITY

- **What the issue is**: Unused variable `Base b;`.
- **Why it’s a problem**: Adds clutter.
- **How to fix it**: Remove unused variables.

5. Category: BUG

- **What the issue is**: Pointer `pb` is `Middle*` instead of `Base*`.
- **Why it’s a problem**: Reduces clarity in polymorphism usage.
- **How to fix it**: Change `Middle* pb;` to `Base* pb;`.

Total bugs: 2
Total quality issues: 3

Snippet ID: any

1. Category: BUG

- **What the issue is**: Checks if `std::any s2 = 1` is empty using `!s2.has_value()`.
- **Why it's a problem**: `s2` has a value (`1`), so the check will always be false.
- **How to fix it**: Initialize `s2` as `std::any{}` if you want to check for emptiness.

2. Category: BUG

- **What the issue is**: Casting `std::any` containing an `int` to `float`.
- **Why it's a problem**: Throws `std::bad_any_cast` since types don't match.
- **How to fix it**: Cast to `int` or store a `float` instead.

3. Category: QUALITY

- **What the issue is**: Use of `std::any` for simple types like `int`, `bool`.
- **Why it's a problem**: Adds unnecessary overhead.
- **How to fix it**: Use native types unless polymorphic storage is required.

4. Category: QUALITY

- **What the issue is**: Unhelpful block comments clutter the code.
- **Why it's a problem**: Reduces readability without adding value.
- **How to fix it**: Remove or replace with meaningful comments.

5. Category: QUALITY

- **What the issue is**: Variable names like `a`, `s2`, `a3` are non-descriptive.
- **Why it's a problem**: Makes the code harder to follow.
- **How to fix it**: Use descriptive variable names.

Total bugs: 2
Total quality issues: 3

Snippet ID: `clock`

1. Category: BUG

- **What the issue is**: Casts `cpu_time_used` to `float`.
- **Why it's a problem**: Unnecessary precision loss.
- **How to fix it**: Remove the cast or use `double`.

2. Category: BUG

- **What the issue is**: Misuses variables in `d3 = end2 - start2`.
- **Why it's a problem**: Incorrect logic—should use `end3` and `start3`.
- **How to fix it**: Correct to `auto d3 = end3 - start3`.

3. Category: QUALITY

- **What the issue is**: Redundant comment `// usually nanoseconds`.
- **Why it's a problem**: May be misleading.
- **How to fix it**: Be explicit using `duration_cast`.

4. Category: QUALITY

- **What the issue is**: Unused includes and misleading comments.
- **Why it's a problem**: Adds noise to the code.
- **How to fix it**: Clean up includes and clarify comments.

5. Category: QUALITY

- **What the issue is**: Variable names with trailing underscores.
- **Why it's a problem**: Not conventional, unclear meaning.
- **How to fix it**: Rename to more descriptive terms.

6. Category: QUALITY

- **What the issue is**: Square-bracket comment blocks.
- **Why it's a problem**: Unconventional style.
- **How to fix it**: Use standard comment styles.

7. Category: QUALITY

- **What the issue is**: Redundant timing code blocks.
- **Why it's a problem**: Violates DRY principle.
- **How to fix it**: Generalize timing logic into a function.

Total bugs: 2
Total quality issues: 5

Snippet ID: `functions`

1. Category: QUALITY

- **What the issue is**: Uses raw pointers.

- **Why it's a problem**: Leads to memory leaks.
- **How to fix it**: Use `std::unique_ptr` or `std::shared_ptr`.

2. Category: QUALITY

- **What the issue is**: Uses raw dynamic arrays.
- **Why it's a problem**: Error-prone and outdated.
- **How to fix it**: Use `std::vector<int>` instead.

3. Category: QUALITY

- **What the issue is**: Redundant check `if (x)` after allocation.
- **Why it's a problem**: Pointless check—allocation success is assumed.
- **How to fix it**: Dereference `x` directly.

4. Category: QUALITY

- **What the issue is**: Outdated comment style.
- **Why it's a problem**: Reduces professionalism and clarity.
- **How to fix it**: Use modern `//` or Doxygen-style comments.

Total bugs: 0
Total quality issues: 4

Snippet ID: `lambda`

1. Category: BUG

- **What the issue is**: Sorts `v` instead of `v2`.
- **Why it's a problem**: Logic error—wrong container is sorted.

- **How to fix it**: Use `std::sort(v2.begin(), v2.end(), ...)`.

2. Category: BUG

- **What the issue is**: Type mismatch in lambda—`const double` for `int` container.
- **Why it's a problem**: May cause warnings or errors.
- **How to fix it**: Use `int c` instead of `double c`.

3. Category: QUALITY

- **What the issue is**: Overuses `std::bind`.
- **Why it's a problem**: Reduces clarity and flexibility.
- **How to fix it**: Replace with a lambda function.

4. Category: QUALITY

- **What the issue is**: Custom functor `add` for a simple operation.
- **Why it's a problem**: Unnecessary complexity.
- **How to fix it**: Use a lambda or `std::plus`.

5. Category: QUALITY

- **What the issue is**: Uses `std::function` unnecessarily.
- **Why it's a problem**: Adds overhead and verbosity.
- **How to fix it**: Use `auto` unless polymorphism is needed.

Total bugs: 2
Total quality issues: 3

Snippet ID: `Multithreading`

1. Category: QUALITY

- **What the issue is**: Threads not checked with `joinable()` before `join()`.
- **Why it's a problem**: May cause runtime errors.
- **How to fix it**: Use `if (t.joinable())`.

2. Category: QUALITY

- **What the issue is**: Lambda functions lack explicit return types.
- **Why it's a problem**: Can reduce clarity.
- **How to fix it**: Use `[]() -> void {}`.

3. Category: QUALITY

- **What the issue is**: Uses `std::async` without a thread pool.
- **Why it's a problem**: May create too many threads inefficiently.
- **How to fix it**: Use a thread pool or manage concurrency explicitly.

4. Category: QUALITY

- **What the issue is**: Misleading comment: `// do some other work`.
- **Why it's a problem**: Nothing happens—comment is inaccurate.
- **How to fix it**: Update comment or perform real work.

5. Category: QUALITY

- **What the issue is**: All thread logic is in `main()`.
- **Why it's a problem**: Reduces readability.

- **How to fix it**: Refactor into functions like `runThreads()`.

Total bugs: 0
Total quality issues: 5

Snippet ID: `optional`

1. Category: BUG

- **What the issue is**: `get_even_random_number2` returns an `optional` with value 0 or 1, regardless of evenness.
- **Why it's a problem**: Misleading—optional is never empty, contradicting its intended use.
- **How to fix it**: Return the actual even number when applicable, or an empty optional otherwise.

2. Category: QUALITY

- **What the issue is**: `float` is used in `std::sqrt(static_cast<float>(*i))`.
- **Why it's a problem**: Reduces precision unnecessarily.
- **How to fix it**: Use `double` instead.

3. Category: QUALITY

- **What the issue is**: Poor function name `get_even_random_number2`.
- **Why it's a problem**: Unclear purpose, hard to distinguish from similarly named functions.
- **How to fix it**: Rename to something descriptive like `get_optional_even_flag_random_number`.

Total bugs: 1
Total quality issues: 2

Snippet ID: `polymorphism`

1. Category: BUG

- **What the issue is**: `square::area()` incorrectly multiplies `_side1 * _side2`.
- **Why it's a problem**: For squares, both sides must be equal—this breaks geometry rules.
- **How to fix it**: Ensure both sides are the same and return `_side1 * _side1`.

2. Category: QUALITY

- **What the issue is**: `shape::area()` returns 0 instead of being abstract.
- **Why it's a problem**: No generic implementation of area; the base class should be abstract.
- **How to fix it**: Declare `area()` as `pure virtual`.

3. Category: QUALITY

- **What the issue is**: `shape` constructors are public.
- **Why it's a problem**: Base class shouldn't be directly instantiated.
- **How to fix it**: Make class abstract to prevent instantiation.

4. Category: QUALITY

- **What the issue is**: Class name `shape` doesn't follow PascalCase.
- **Why it's a problem**: Violates C++ naming conventions.
- **How to fix it**: Rename to `Shape`.

5. Category: QUALITY

- **What the issue is**: `triangle::area()` assumes `_side1` is base and `_side2` is height.
- **Why it's a problem**: Ambiguity reduces clarity and flexibility.
- **How to fix it**: Clarify assumption or use better structure.

6. Category: QUALITY

- **What the issue is**: Type check with `dynamic_cast` in `main()`.
- **Why it's a problem**: Indicates poor polymorphic design.
- **How to fix it**: Add a virtual type identifier function in `shape`.

Total bugs: 1
Total quality issues: 5

Snippet ID: `random`

1. Category: BUG

- **What the issue is**: Invalid string literal concatenation in `fnv(...)`.
- **Why it's a problem**: Can cause compile-time errors.
- **How to fix it**: Add spaces between string macros.

2. Category: QUALITY

- **What the issue is**: Inconsistent use of `default_random_engine` and `mt19937`.
- **Why it's a problem**: Causes confusion.
- **How to fix it**: Use `mt19937` consistently.

3. Category: QUALITY

- **What the issue is**: Entropy from memory addresses is non-portable.
- **Why it's a problem**: Leads to inconsistent behavior.
- **How to fix it**: Use portable entropy sources.

4. Category: QUALITY

- **What the issue is**: Magic number `45` in seed mix.
- **Why it's a problem**: Reduces clarity.
- **How to fix it**: Replace with a named constant and document it.

5. Category: QUALITY

- **What the issue is**: Unconventional comment styles.
- **Why it's a problem**: Reduces readability.
- **How to fix it**: Use standard comment conventions.

6. Category: QUALITY

- **What the issue is**: No error check on `std::random_device`.
- **Why it's a problem**: `entropy()` might return 0.
- **How to fix it**: Check entropy and provide fallback.

Total bugs: 1
Total quality issues: 5

Snippet ID: `tempcast`

1. Category: BUG

- **What the issue is**: No check for division by zero.

- **Why it's a problem**: Leads to undefined behavior.
- **How to fix it**: Check and handle zero divisor before division.

2. Category: BUG

- **What the issue is**: Uninitialized variable is used.
- **Why it's a problem**: Causes unpredictable behavior.
- **How to fix it**: Always initialize variables before use.

3. Category: QUALITY

- **What the issue is**: Poorly named variables.
- **Why it's a problem**: Reduces clarity and readability.
- **How to fix it**: Use meaningful names.

4. Category: QUALITY

- **What the issue is**: Inconsistent indentation and formatting.
- **Why it's a problem**: Makes code harder to maintain.
- **How to fix it**: Use consistent formatting tools or IDE features.

5. Category: QUALITY

- **What the issue is**: Lacks documentation or comments.
- **Why it's a problem**: Code becomes difficult to understand.
- **How to fix it**: Add explanatory comments to complex sections.

Total bugs: 2
Total quality issues: 3

Snippet ID: `tuples`

1. Category: BUG

- **What the issue is**: A `std::pair` is labeled as `std::tuple` in comments.
- **Why it's a problem**: Misleads readers.
- **How to fix it**: Correct the comment or rename the variable.

2. Category: QUALITY

- **What the issue is**: Unused header `<unordered_map>`.
- **Why it's a problem**: Adds compilation time and clutter.
- **How to fix it**: Remove unused headers.

3. Category: QUALITY

- **What the issue is**: Redundant struct `my_pair` mimics `std::pair`.
- **Why it's a problem**: Unnecessary code duplication.
- **How to fix it**: Use `std::pair` unless extra functionality is needed.

4. Category: QUALITY

- **What the issue is**: Inline comments like `//[return_ref ...]` clutter the code.
- **Why it's a problem**: Reduce readability.
- **How to fix it**: Replace with standard comments or remove.

5. Category: QUALITY

- **What the issue is**: Explicit template args in `std::make_pair`.
- **Why it's a problem**: Verbose and unidiomatic.

- **How to fix it**: Rely on type deduction.

Total bugs: 1
Total quality issues: 4