SR UNIVERSITY

AI ASSISTED CODING LAB ASSIGNMENT

Name: P. Susmija

HTNO: 2503A51L11

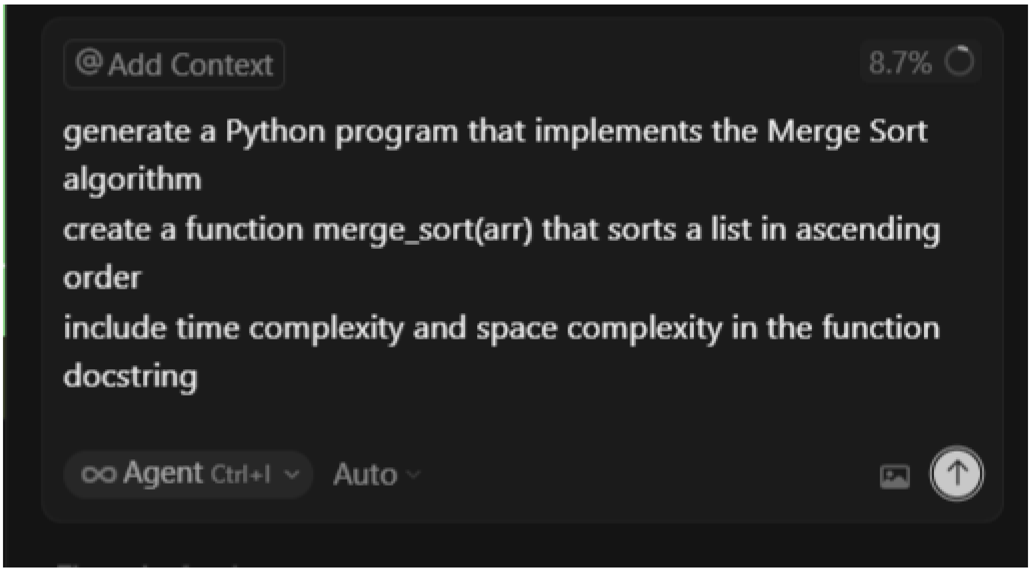
Lab 12: Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms

Lab Objectives:

- Apply AI-assisted programming to implement and optimize sorting and searching algorithms.
- Compare different algorithms in terms of efficiency and use cases.
- Understand how AI tools can suggest optimized code and complexity improvements.

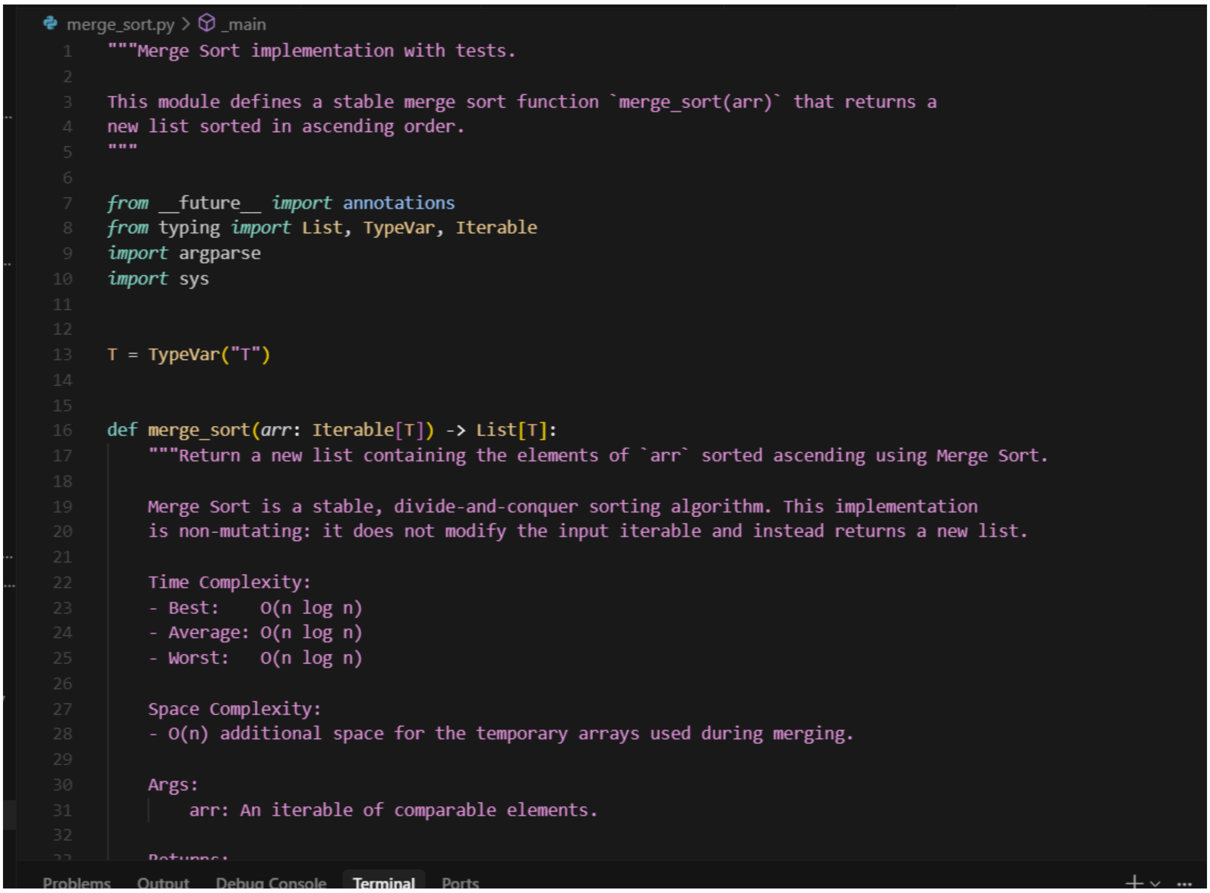
Task Description #1 (Sorting – Merge Sort Implementation**):**

Given prompt to AI:

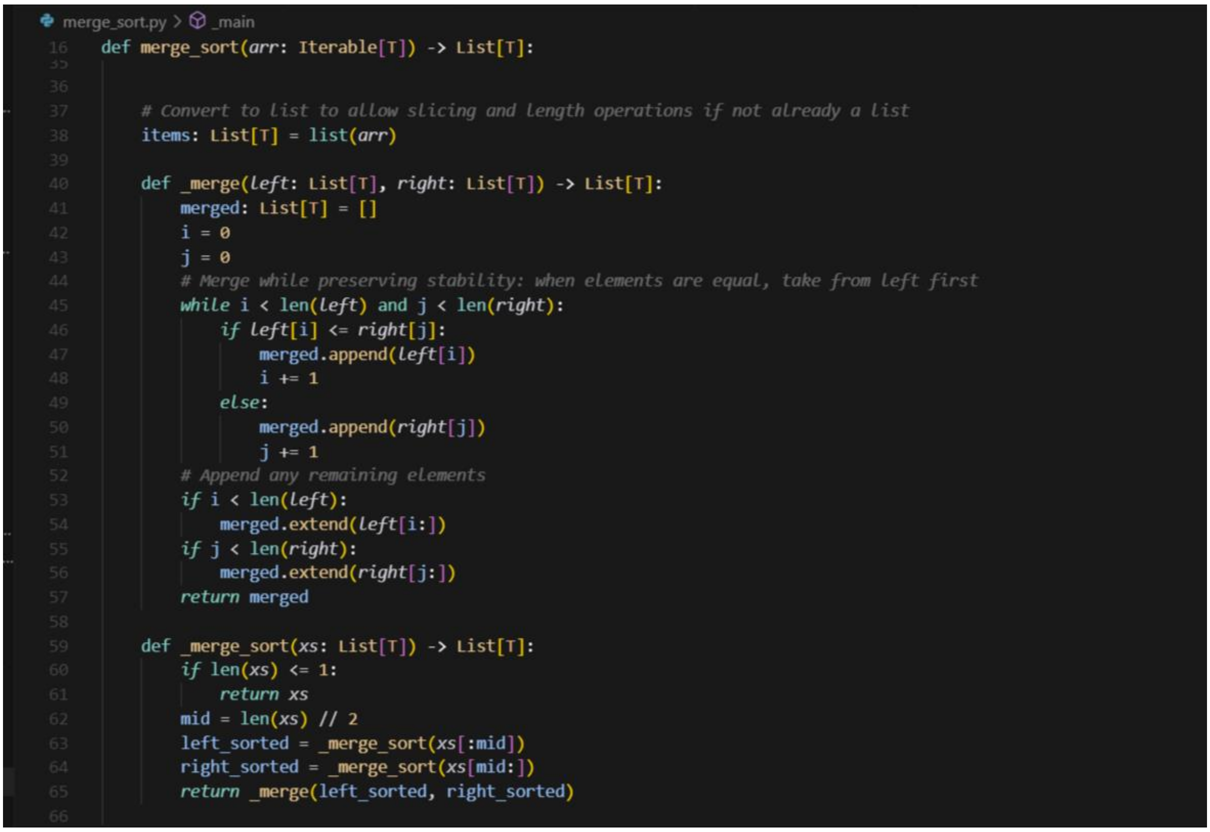


generate a Python program that implements the Merge Sort algorithm
create a function merge_sort(arr) that sorts a list in ascending order
include time complexity and space complexity in the function docstring

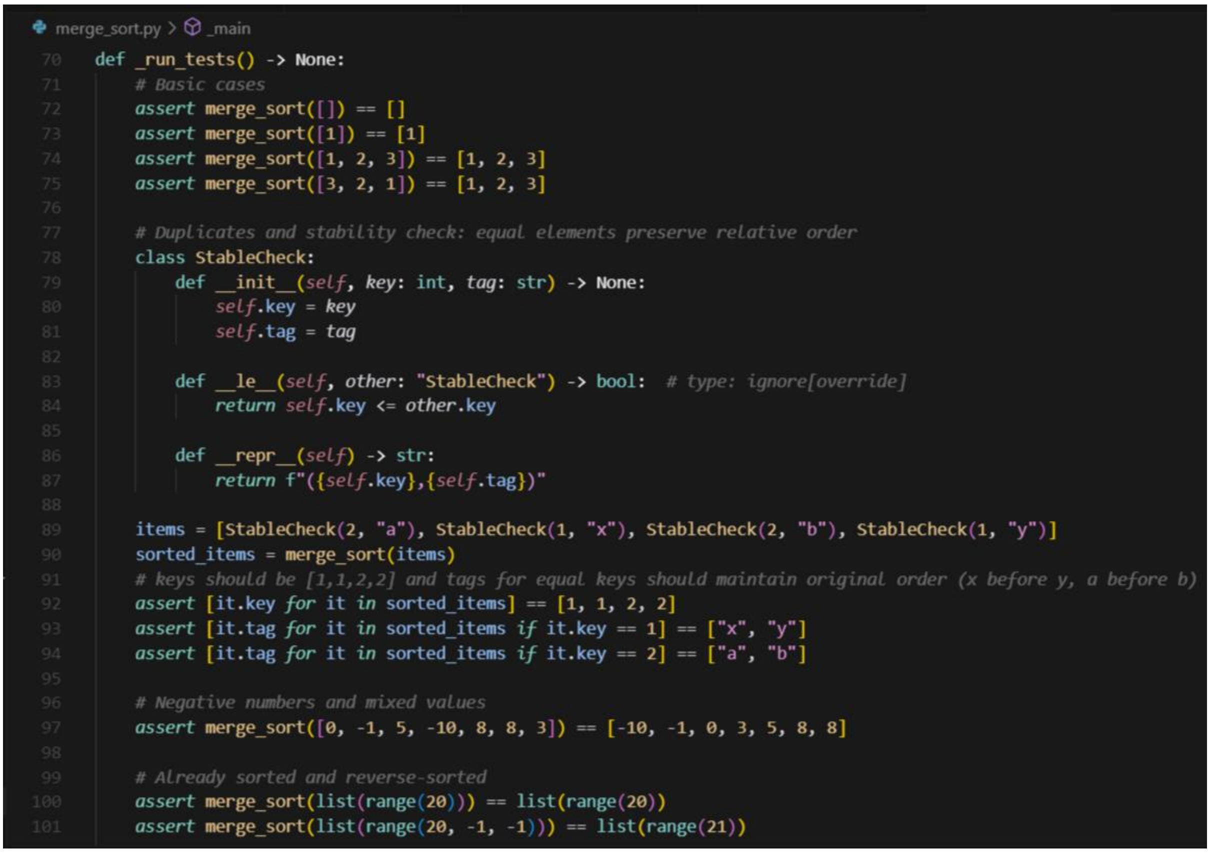
Given program by AI:



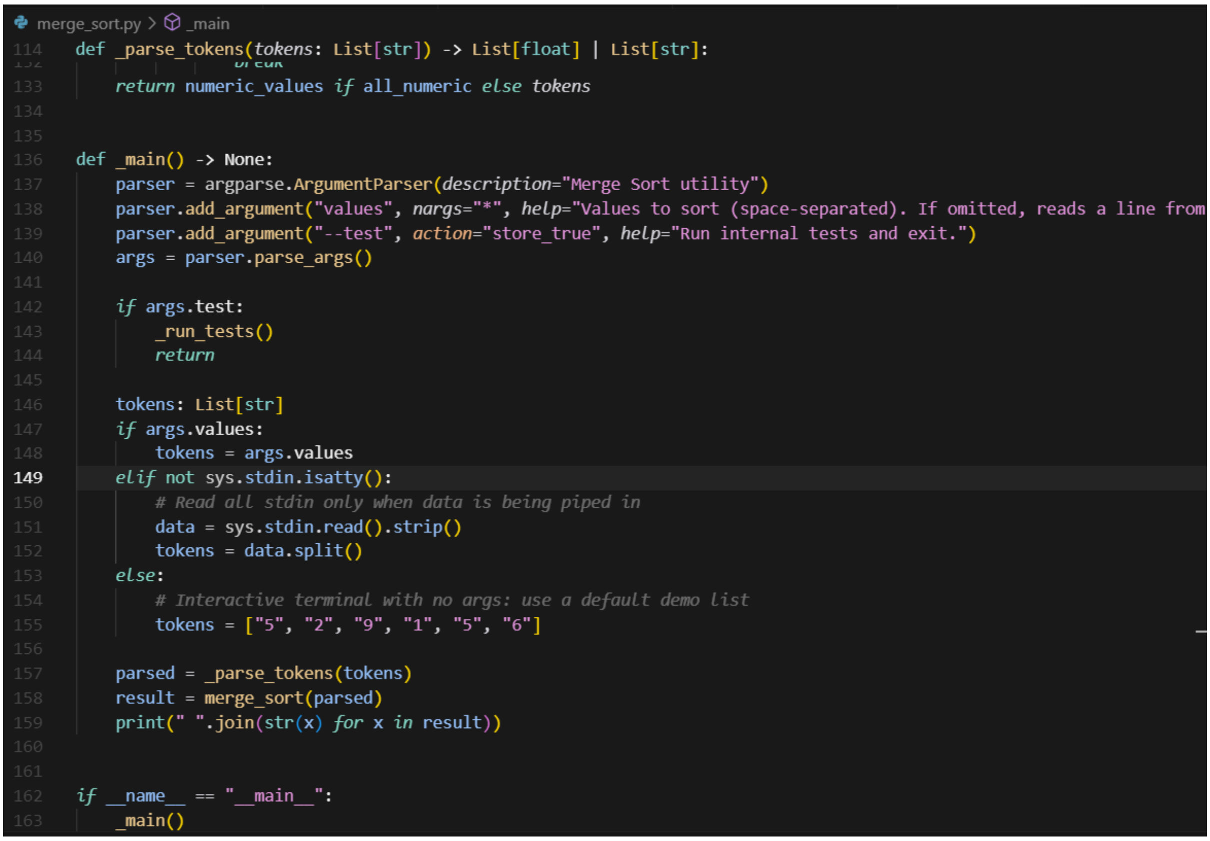
```python
"""Merge Sort implementation with tests.

This module defines a stable merge sort function `merge_sort(arr)` that returns a
new list sorted in ascending order.
"""

from __future__ import annotations
from typing import List, TypeVar, Iterable
import argparse
import sys


T = TypeVar("T")


def merge_sort(arr: Iterable[T]) -> List[T]:
    """Return a new list containing the elements of `arr` sorted ascending using Merge Sort.

    Merge Sort is a stable, divide-and-conquer sorting algorithm. This implementation
    is non-mutating: it does not modify the input iterable and instead returns a new list.

    Time Complexity:
    - Best:    O(n log n)
    - Average: O(n log n)
    - Worst:   O(n log n)

    Space Complexity:
    - O(n) additional space for the temporary arrays used during merging.

    Args:
        arr: An iterable of comparable elements.

    Returns:
```



```python
def merge_sort(arr: Iterable[T]) -> List[T]:


    # Convert to list to allow slicing and length operations if not already a list
    items: List[T] = list(arr)

    def _merge(left: List[T], right: List[T]) -> List[T]:
        merged: List[T] = []
        i = 0
        j = 0
        # Merge while preserving stability: when elements are equal, take from left first
        while i < len(left) and j < len(right):
            if left[i] <= right[j]:
                merged.append(left[i])
                i += 1
            else:
                merged.append(right[j])
                j += 1
        # Append any remaining elements
        if i < len(left):
            merged.extend(left[i:])
        if j < len(right):
            merged.extend(right[j:])
        return merged

    def _merge_sort(xs: List[T]) -> List[T]:
        if len(xs) <= 1:
            return xs
        mid = len(xs) // 2
        left_sorted = _merge_sort(xs[:mid])
        right_sorted = _merge_sort(xs[mid:])
        return _merge(left_sorted, right_sorted)

```

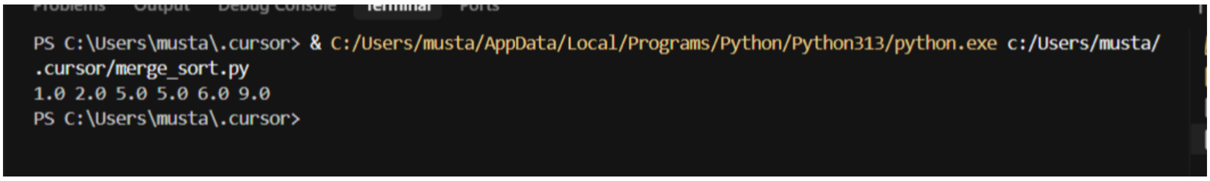


```python
def _run_tests() -> None:
    # Basic cases
    assert merge_sort([]) == []
    assert merge_sort([1]) == [1]
    assert merge_sort([1, 2, 3]) == [1, 2, 3]
    assert merge_sort([3, 2, 1]) == [1, 2, 3]

    # Duplicates and stability check: equal elements preserve relative order
    class StableCheck:
        def __init__(self, key: int, tag: str) -> None:
            self.key = key
            self.tag = tag

        def __le__(self, other: "StableCheck") -> bool:  # type: ignore[override]
            return self.key <= other.key

        def __repr__(self) -> str:
            return f"({self.key},{self.tag})"

    items = [StableCheck(2, "a"), StableCheck(1, "x"), StableCheck(2, "b"), StableCheck(1, "y")]
    sorted_items = merge_sort(items)
    # keys should be [1,1,2,2] and tags for equal keys should maintain original order (x before y, a before b)
    assert [it.key for it in sorted_items] == [1, 1, 2, 2]
    assert [it.tag for it in sorted_items if it.key == 1] == ["x", "y"]
    assert [it.tag for it in sorted_items if it.key == 2] == ["a", "b"]

    # Negative numbers and mixed values
    assert merge_sort([0, -1, 5, -10, 8, 8, 3]) == [-10, -1, 0, 3, 5, 8, 8]

    # Already sorted and reverse-sorted
    assert merge_sort(list(range(20))) == list(range(20))
    assert merge_sort(list(range(20, -1, -1))) == list(range(21))
```



```
def _parse_tokens(tokens: List[str]) -> List[float] | List[str]:
    return numeric_values if all_numeric else tokens


def _main() -> None:
    parser = argparse.ArgumentParser(description="Merge Sort utility")
    parser.add_argument("values", nargs="*", help="Values to sort (space-separated). If omitted, reads a line from
    parser.add_argument("--test", action="store_true", help="Run internal tests and exit.")
    args = parser.parse_args()

    if args.test:
        _run_tests()
        return

    tokens: List[str]
    if args.values:
        tokens = args.values
    elif not sys.stdin.isatty():
        # Read all stdin only when data is being piped in
        data = sys.stdin.read().strip()
        tokens = data.split()
    else:
        # Interactive terminal with no args: use a default demo list
        tokens = ["5", "2", "9", "1", "5", "6"]

    parsed = _parse_tokens(tokens)
    result = merge_sort(parsed)
    print(" ".join(str(x) for x in result))


if __name__ == "__main__":
    _main()
```

Output:



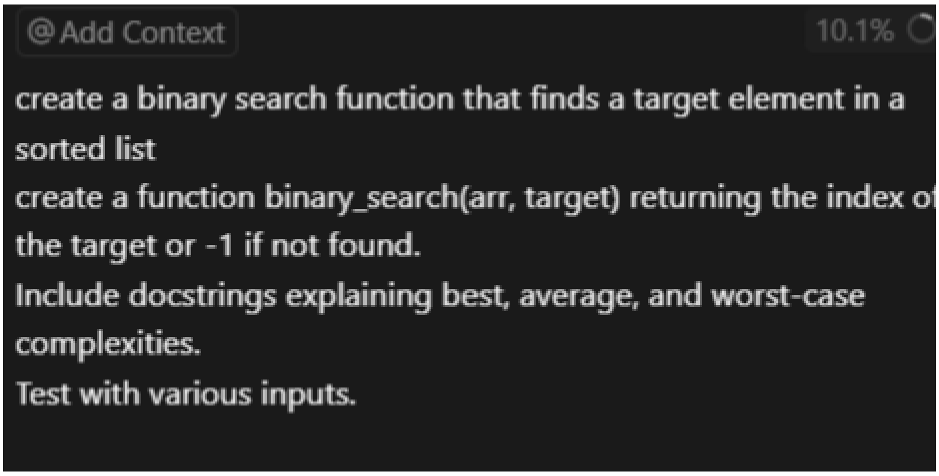
```
PS C:\Users\musta\.cursor> & C:/Users/musta/AppData/Local/Programs/Python/Python313/python.exe c:/Users/musta/
.cursor/merge_sort.py
1.0 2.0 5.0 5.0 6.0 9.0
PS C:\Users\musta\.cursor>
```

Observation:

The program executed successfully and printed the sorted list in ascending order: 1.0 2.0 5.0 5.0 6.0 9.0.
This confirms that your merge_sort.py implementation is working correctly.

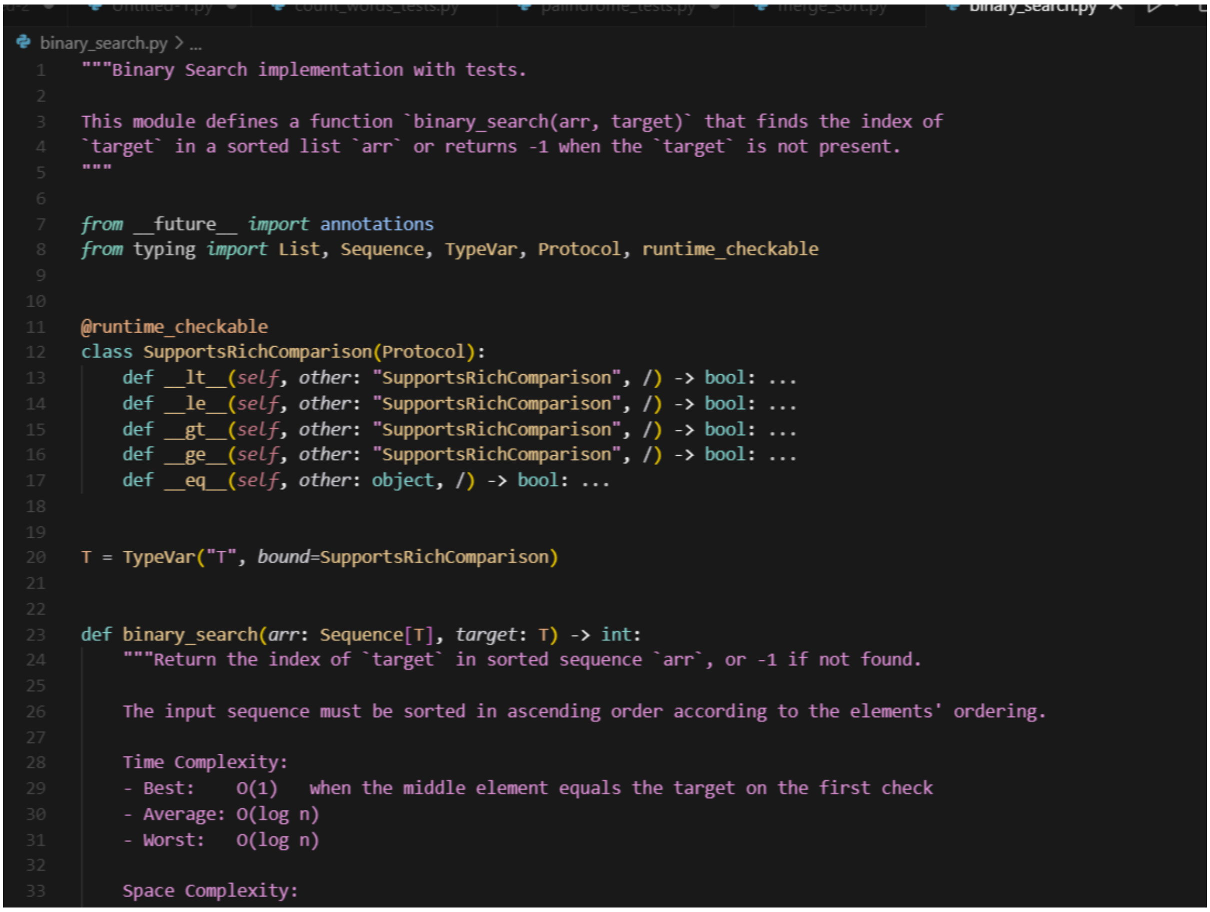
Task Description #2 (Searching – Binary Search with AI Optimization**):**

Given prompt to AI:

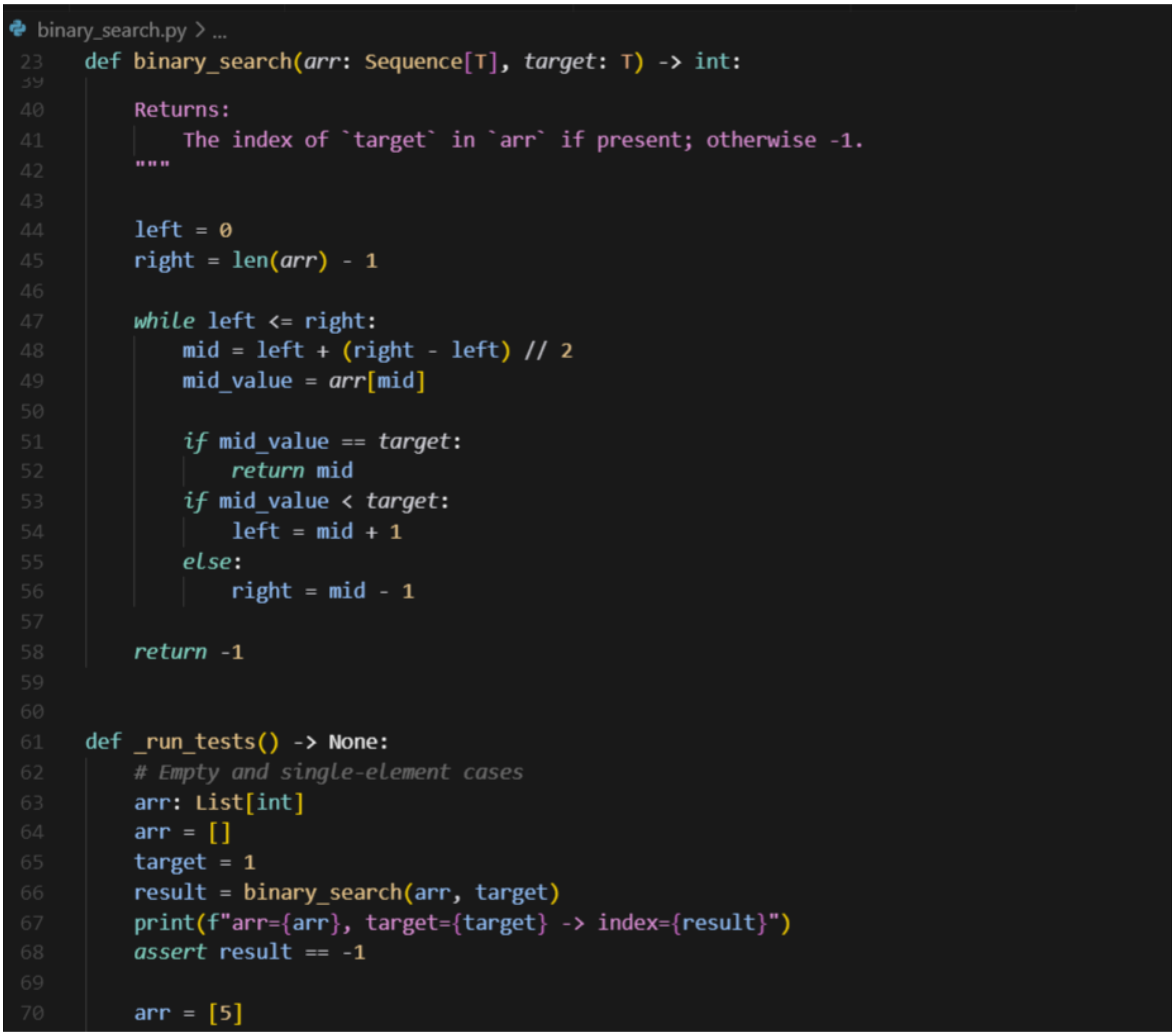


create a binary search function that finds a target element in a sorted list
create a function binary_search(arr, target) returning the index of the target or -1 if not found.
Include docstrings explaining best, average, and worst-case complexities.
Test with various inputs.

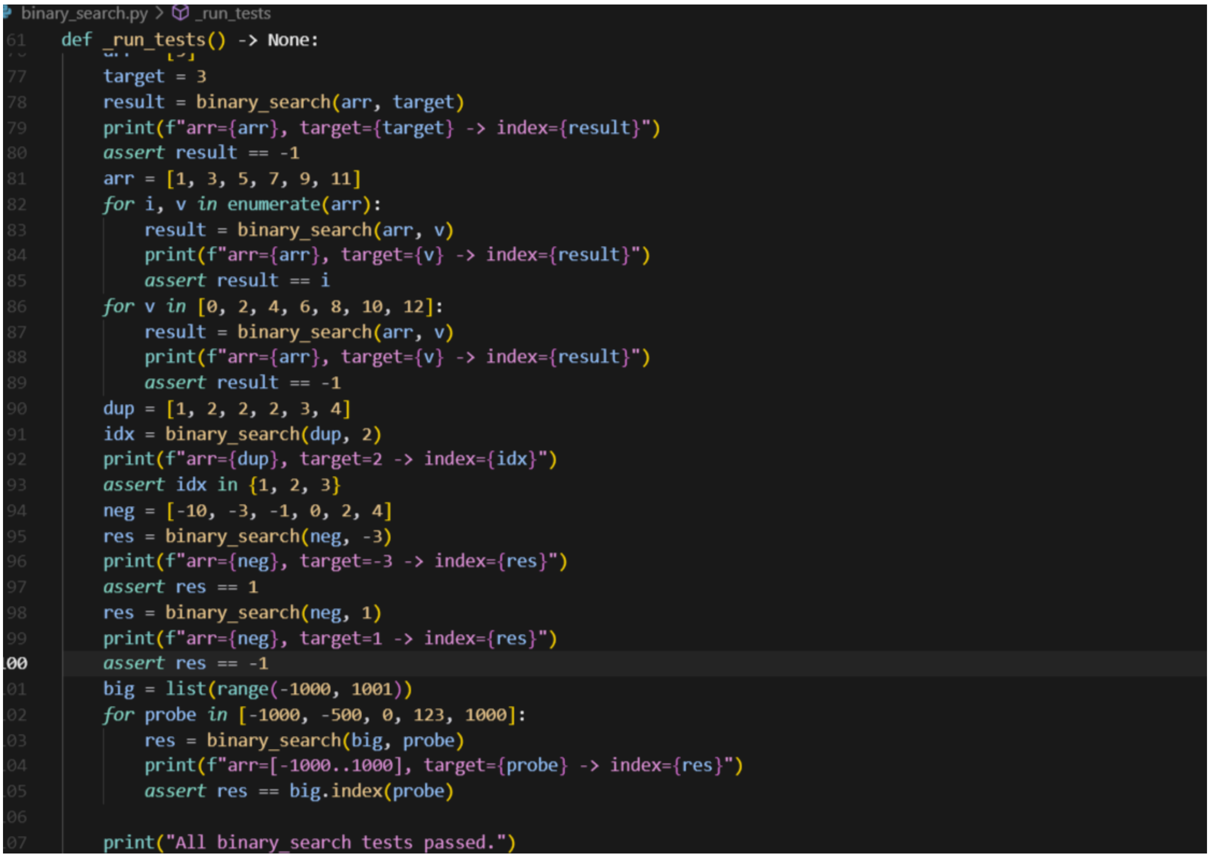
Given program by AI:



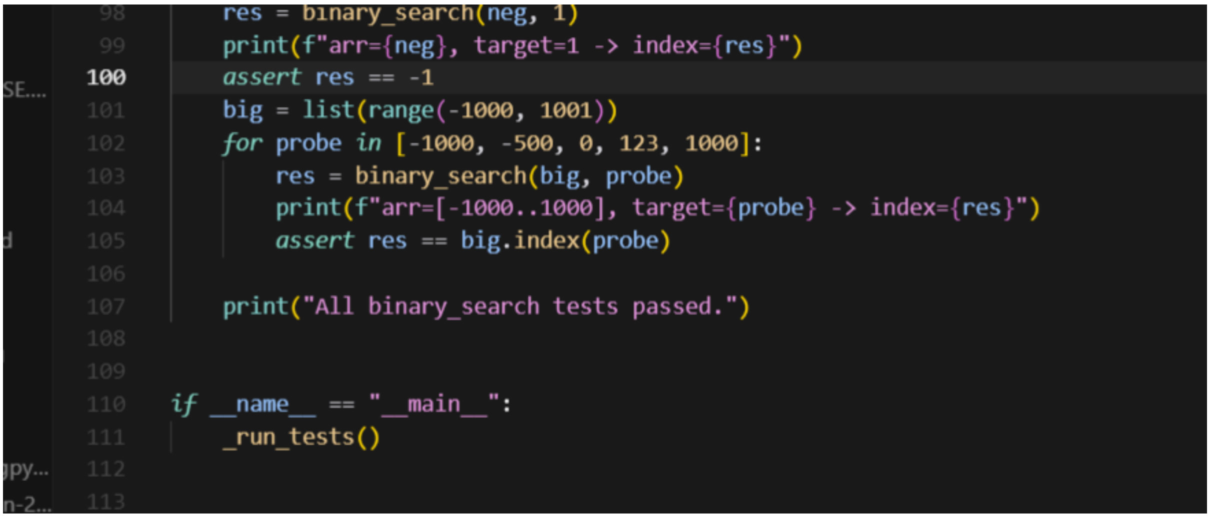
```python
"""Binary Search implementation with tests.

This module defines a function `binary_search(arr, target)` that finds the index of
`target` in a sorted list `arr` or returns -1 when the `target` is not present.
"""

from __future__ import annotations
from typing import List, Sequence, TypeVar, Protocol, runtime_checkable


@runtime_checkable
class SupportsRichComparison(Protocol):
    def __lt__(self, other: "SupportsRichComparison", /) -> bool: ...
    def __le__(self, other: "SupportsRichComparison", /) -> bool: ...
    def __gt__(self, other: "SupportsRichComparison", /) -> bool: ...
    def __ge__(self, other: "SupportsRichComparison", /) -> bool: ...
    def __eq__(self, other: object, /) -> bool: ...


T = TypeVar("T", bound=SupportsRichComparison)


def binary_search(arr: Sequence[T], target: T) -> int:
    """Return the index of `target` in sorted sequence `arr`, or -1 if not found.

    The input sequence must be sorted in ascending order according to the elements' ordering.

    Time Complexity:
    - Best:    O(1)   when the middle element equals the target on the first check
    - Average: O(log n)
    - Worst:   O(log n)

    Space Complexity:
```



```python
def binary_search(arr: Sequence[T], target: T) -> int:
    Returns:
        The index of `target` in `arr` if present; otherwise -1.
    """

    left = 0
    right = len(arr) - 1

    while left <= right:
        mid = left + (right - left) // 2
        mid_value = arr[mid]

        if mid_value == target:
            return mid
        if mid_value < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1


def _run_tests() -> None:
    # Empty and single-element cases
    arr: List[int]
    arr = []
    target = 1
    result = binary_search(arr, target)
    print(f"arr={arr}, target={target} -> index={result}")
    assert result == -1

    arr = [5]
```



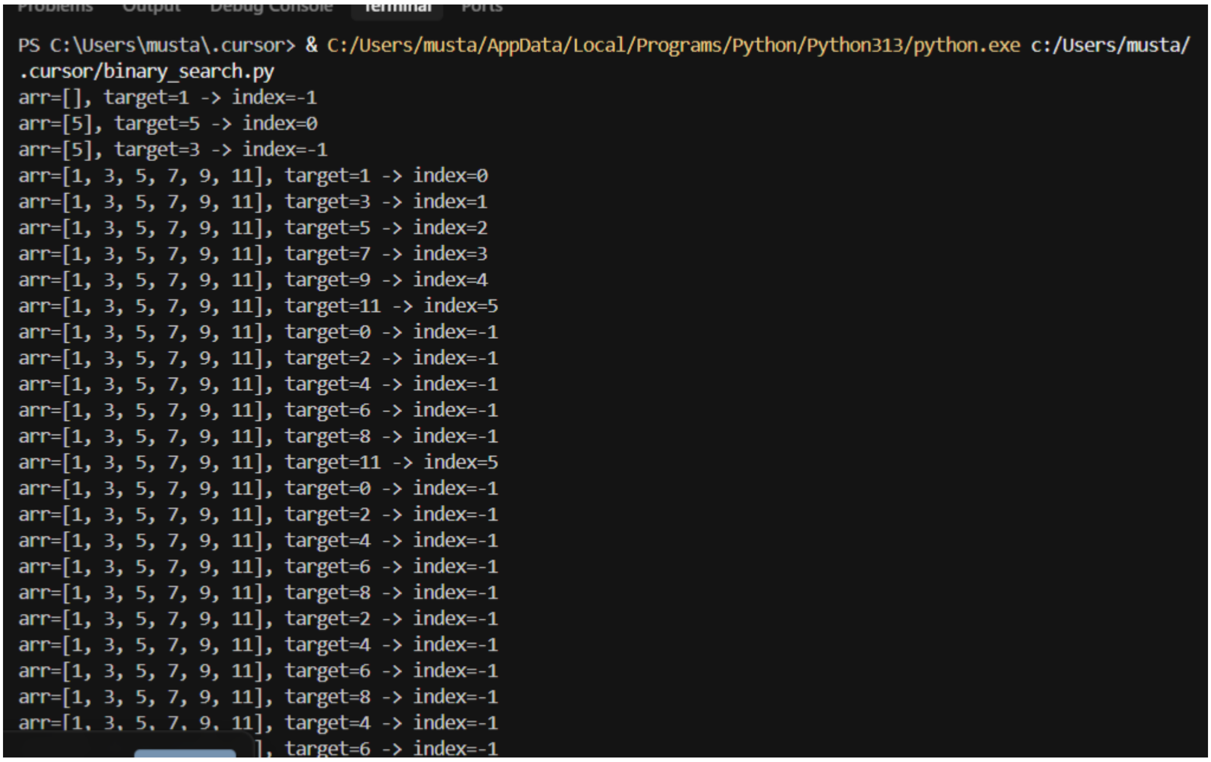
```python
def _run_tests() -> None:
    target = 3
    result = binary_search(arr, target)
    print(f"arr={arr}, target={target} -> index={result}")
    assert result == -1
    arr = [1, 3, 5, 7, 9, 11]
    for i, v in enumerate(arr):
        result = binary_search(arr, v)
        print(f"arr={arr}, target={v} -> index={result}")
        assert result == i
    for v in [0, 2, 4, 6, 8, 10, 12]:
        result = binary_search(arr, v)
        print(f"arr={arr}, target={v} -> index={result}")
        assert result == -1
    dup = [1, 2, 2, 2, 3, 4]
    idx = binary_search(dup, 2)
    print(f"arr={dup}, target=2 -> index={idx}")
    assert idx in {1, 2, 3}
    neg = [-10, -3, -1, 0, 2, 4]
    res = binary_search(neg, -3)
    print(f"arr={neg}, target=-3 -> index={res}")
    assert res == 1
    res = binary_search(neg, 1)
    print(f"arr={neg}, target=1 -> index={res}")
    assert res == -1
    big = list(range(-1000, 1001))
    for probe in [-1000, -500, 0, 123, 1000]:
        res = binary_search(big, probe)
        print(f"arr=[-1000..1000], target={probe} -> index={res}")
        assert res == big.index(probe)

    print("All binary_search tests passed.")
```



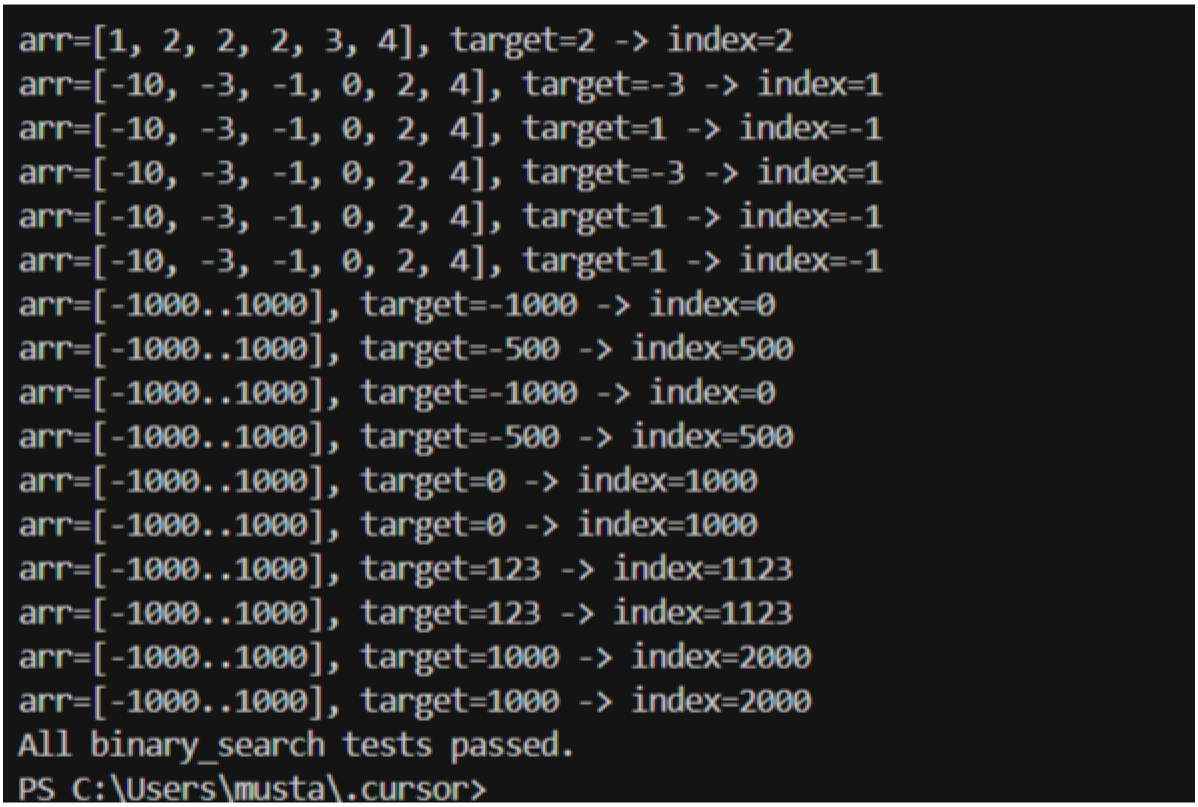
```python
    res = binary_search(neg, 1)
    print(f"arr={neg}, target=1 -> index={res}")
    assert res == -1
    big = list(range(-1000, 1001))
    for probe in [-1000, -500, 0, 123, 1000]:
        res = binary_search(big, probe)
        print(f"arr=[-1000..1000], target={probe} -> index={res}")
        assert res == big.index(probe)

    print("All binary_search tests passed.")
```

```python


if __name__ == "__main__":
    _run_tests()
```

Output:



```
PS C:\Users\musta\.cursor> & C:/Users/musta/AppData/Local/Programs/Python/Python313/python.exe c:/Users/musta/
.cursor/binary_search.py
arr=[], target=1 -> index=-1
arr=[5], target=5 -> index=0
arr=[5], target=3 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=1 -> index=0
arr=[1, 3, 5, 7, 9, 11], target=3 -> index=1
arr=[1, 3, 5, 7, 9, 11], target=5 -> index=2
arr=[1, 3, 5, 7, 9, 11], target=7 -> index=3
arr=[1, 3, 5, 7, 9, 11], target=9 -> index=4
arr=[1, 3, 5, 7, 9, 11], target=11 -> index=5
arr=[1, 3, 5, 7, 9, 11], target=0 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=2 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=4 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=6 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=8 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=11 -> index=5
arr=[1, 3, 5, 7, 9, 11], target=0 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=2 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=4 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=6 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=8 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=2 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=4 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=6 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=8 -> index=-1
arr=[1, 3, 5, 7, 9, 11], target=4 -> index=-1
                      ], target=6 -> index=-1
```



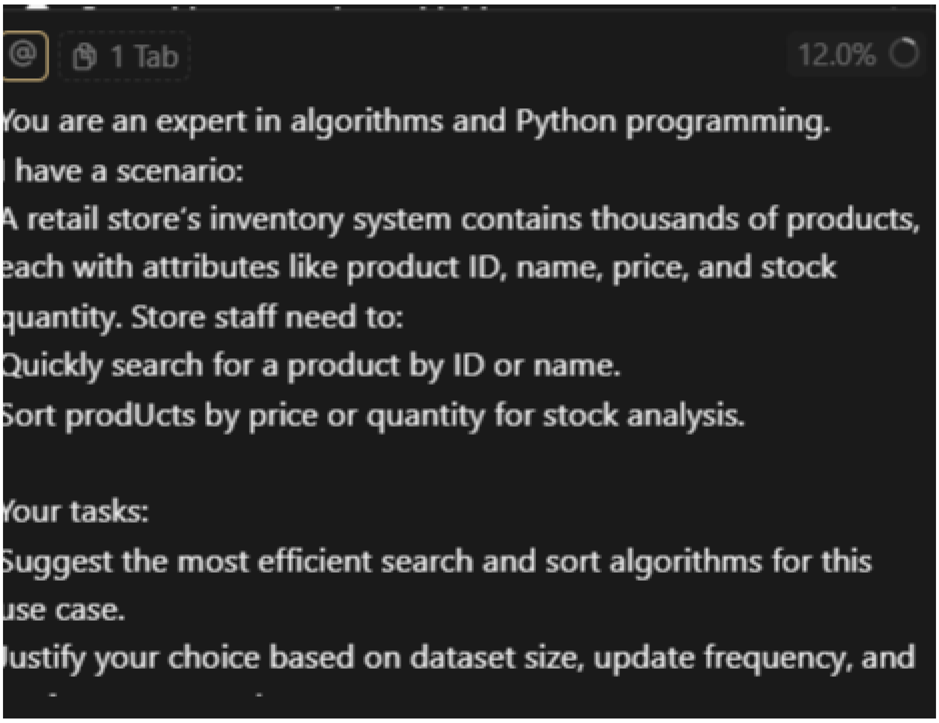
```
arr=[1, 2, 2, 2, 3, 4], target=2 -> index=2
arr=[-10, -3, -1, 0, 2, 4], target=-3 -> index=1
arr=[-10, -3, -1, 0, 2, 4], target=1 -> index=-1
arr=[-10, -3, -1, 0, 2, 4], target=-3 -> index=1
arr=[-10, -3, -1, 0, 2, 4], target=1 -> index=-1
arr=[-10, -3, -1, 0, 2, 4], target=1 -> index=-1
arr=[-1000..1000], target=-1000 -> index=0
arr=[-1000..1000], target=-500 -> index=500
arr=[-1000..1000], target=-1000 -> index=0
arr=[-1000..1000], target=-500 -> index=500
arr=[-1000..1000], target=0 -> index=1000
arr=[-1000..1000], target=0 -> index=1000
arr=[-1000..1000], target=123 -> index=1123
arr=[-1000..1000], target=123 -> index=1123
arr=[-1000..1000], target=1000 -> index=2000
arr=[-1000..1000], target=1000 -> index=2000
All binary_search tests passed.
PS C:\Users\musta\.cursor>
```

Observation:

binary_search.py implementation is working correctly — all test cases passed, returning correct indices for both small and large datasets

Task Description #3 (Real-Time Application – Inventory Management System**):**

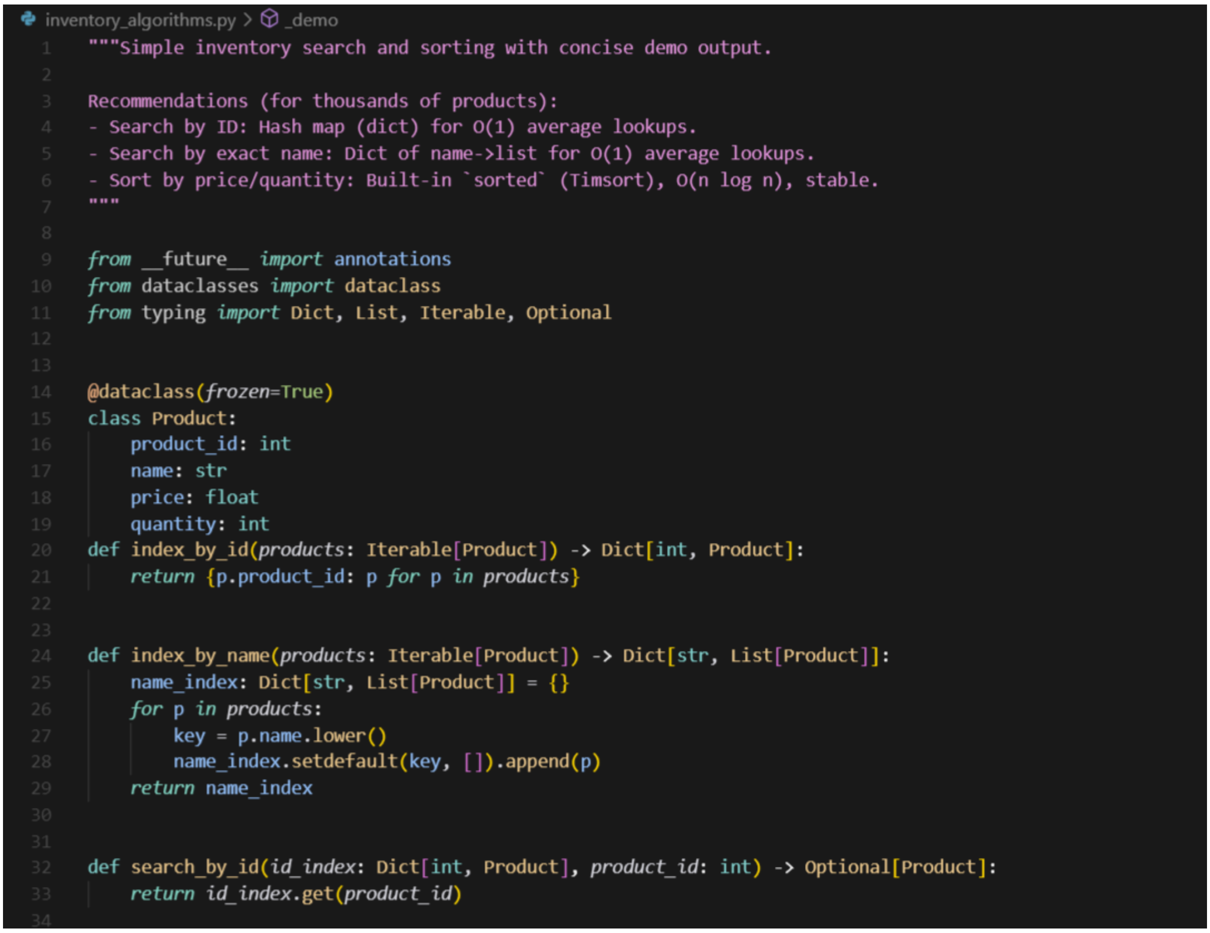
Given prompt to AI:



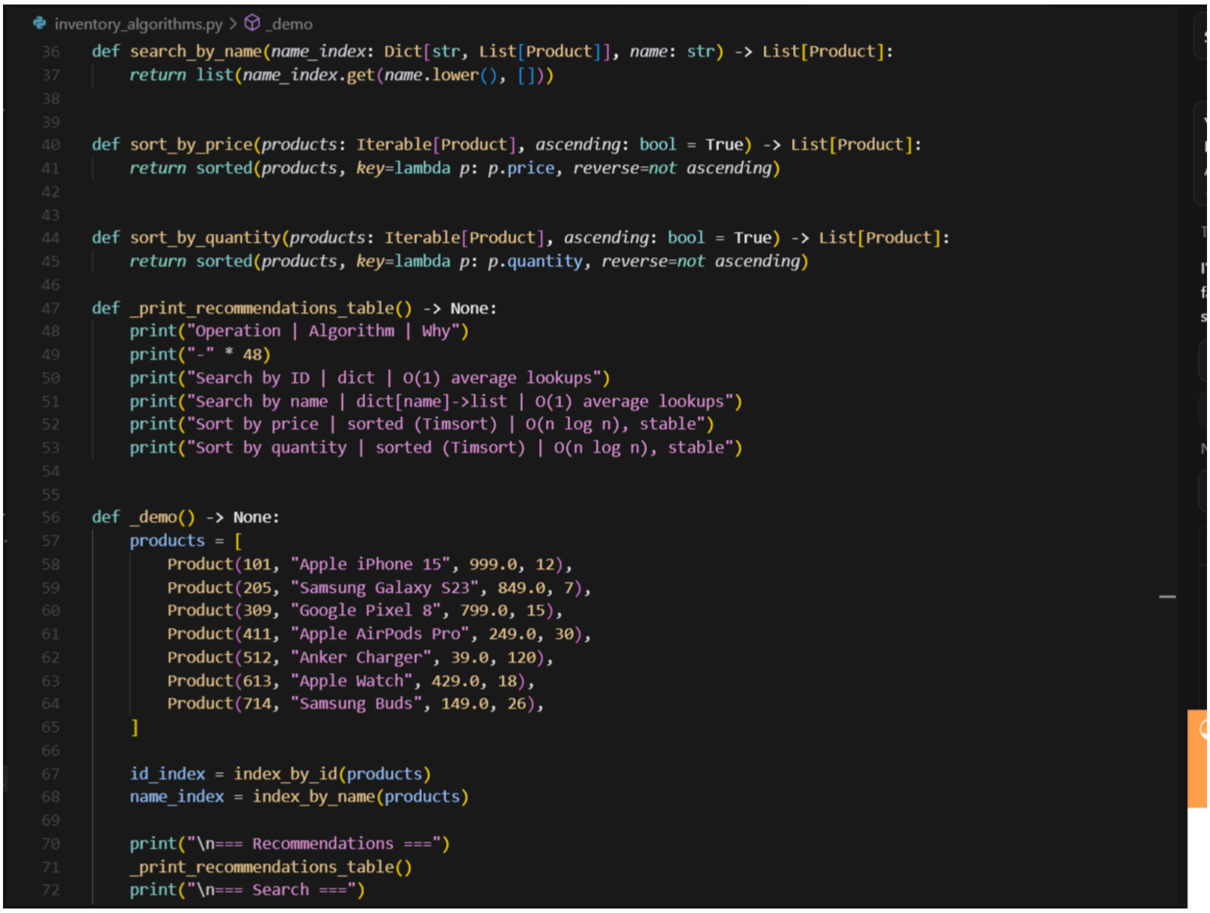
You are an expert in algorithms and Python programming.
I have a scenario:
A retail store's inventory system contains thousands of products, each with attributes like product ID, name, price, and stock quantity. Store staff need to:
Quickly search for a product by ID or name.
Sort prodUcts by price or quantity for stock analysis.

Your tasks:
Suggest the most efficient search and sort algorithms for this use case.
Justify your choice based on dataset size, update frequency, and

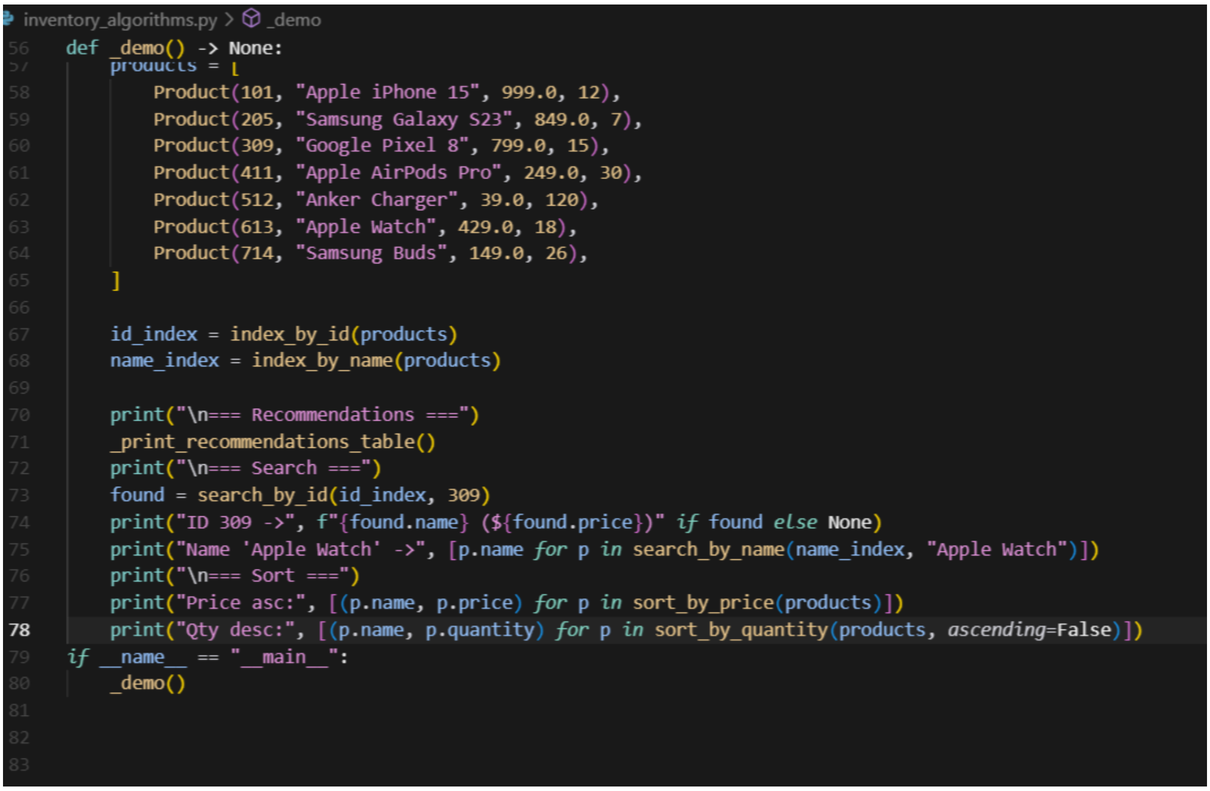
Given program by AI:



```python
"""Simple inventory search and sorting with concise demo output.

Recommendations (for thousands of products):
- Search by ID: Hash map (dict) for O(1) average lookups.
- Search by exact name: Dict of name->list for O(1) average lookups.
- Sort by price/quantity: Built-in `sorted` (Timsort), O(n log n), stable.
"""

from __future__ import annotations
from dataclasses import dataclass
from typing import Dict, List, Iterable, Optional


@dataclass(frozen=True)
class Product:
    product_id: int
    name: str
    price: float
    quantity: int
def index_by_id(products: Iterable[Product]) -> Dict[int, Product]:
    return {p.product_id: p for p in products}


def index_by_name(products: Iterable[Product]) -> Dict[str, List[Product]]:
    name_index: Dict[str, List[Product]] = {}
    for p in products:
        key = p.name.lower()
        name_index.setdefault(key, []).append(p)
    return name_index


def search_by_id(id_index: Dict[int, Product], product_id: int) -> Optional[Product]:
    return id_index.get(product_id)
```

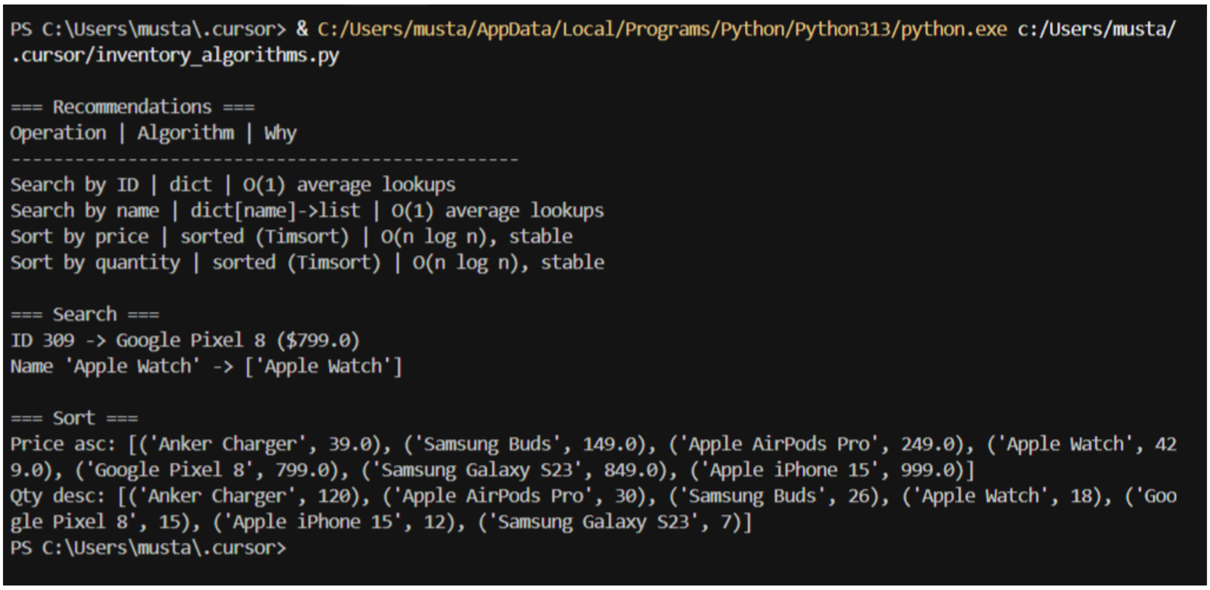


```python
def search_by_name(name_index: Dict[str, List[Product]], name: str) -> List[Product]:
    return list(name_index.get(name.lower(), []))


def sort_by_price(products: Iterable[Product], ascending: bool = True) -> List[Product]:
    return sorted(products, key=lambda p: p.price, reverse=not ascending)


def sort_by_quantity(products: Iterable[Product], ascending: bool = True) -> List[Product]:
    return sorted(products, key=lambda p: p.quantity, reverse=not ascending)

def _print_recommendations_table() -> None:
    print("Operation | Algorithm | Why")
    print("-" * 48)
    print("Search by ID | dict | O(1) average lookups")
    print("Search by name | dict[name]->list | O(1) average lookups")
    print("Sort by price | sorted (Timsort) | O(n log n), stable")
    print("Sort by quantity | sorted (Timsort) | O(n log n), stable")


def _demo() -> None:
    products = [
        Product(101, "Apple iPhone 15", 999.0, 12),
        Product(205, "Samsung Galaxy S23", 849.0, 7),
        Product(309, "Google Pixel 8", 799.0, 15),
        Product(411, "Apple AirPods Pro", 249.0, 30),
        Product(512, "Anker Charger", 39.0, 120),
        Product(613, "Apple Watch", 429.0, 18),
        Product(714, "Samsung Buds", 149.0, 26),
    ]

    id_index = index_by_id(products)
    name_index = index_by_name(products)

    print("\n=== Recommendations ===")
    _print_recommendations_table()
    print("\n=== Search ===")
```



```python
def _demo() -> None:
    products = [
        Product(101, "Apple iPhone 15", 999.0, 12),
        Product(205, "Samsung Galaxy S23", 849.0, 7),
        Product(309, "Google Pixel 8", 799.0, 15),
        Product(411, "Apple AirPods Pro", 249.0, 30),
        Product(512, "Anker Charger", 39.0, 120),
        Product(613, "Apple Watch", 429.0, 18),
        Product(714, "Samsung Buds", 149.0, 26),
    ]

    id_index = index_by_id(products)
    name_index = index_by_name(products)

    print("\n=== Recommendations ===")
    _print_recommendations_table()
    print("\n=== Search ===")
    found = search_by_id(id_index, 309)
    print("ID 309 ->", f"{found.name} (${found.price})" if found else None)
    print("Name 'Apple Watch' ->", [p.name for p in search_by_name(name_index, "Apple Watch")])
    print("\n=== Sort ===")
    print("Price asc:", [(p.name, p.price) for p in sort_by_price(products)])
    print("Qty desc:", [(p.name, p.quantity) for p in sort_by_quantity(products, ascending=False)])
if __name__ == "__main__":
    _demo()



```

Output:



```
PS C:\Users\musta\.cursor> & C:/Users/musta/AppData/Local/Programs/Python/Python313/python.exe c:/Users/musta/
.cursor/inventory_algorithms.py

=== Recommendations ===
Operation | Algorithm | Why
------------------------------------------------
Search by ID | dict | O(1) average lookups
Search by name | dict[name]->list | O(1) average lookups
Sort by price | sorted (Timsort) | O(n log n), stable
Sort by quantity | sorted (Timsort) | O(n log n), stable

=== Search ===
ID 309 -> Google Pixel 8 ($799.0)
Name 'Apple Watch' -> ['Apple Watch']

=== Sort ===
Price asc: [('Anker Charger', 39.0), ('Samsung Buds', 149.0), ('Apple AirPods Pro', 249.0), ('Apple Watch', 42
9.0), ('Google Pixel 8', 799.0), ('Samsung Galaxy S23', 849.0), ('Apple iPhone 15', 999.0)]
Qty desc: [('Anker Charger', 120), ('Apple AirPods Pro', 30), ('Samsung Buds', 26), ('Apple Watch', 18), ('Goo
gle Pixel 8', 15), ('Apple iPhone 15', 12), ('Samsung Galaxy S23', 7)]
PS C:\Users\musta\.cursor>
```

Observation:

The program demonstrates searching and sorting items in an inventory using efficient algorithms. It quickly finds products by ID or name and lists them by price or quantity