Problem: Find Pairs

Given a list of **unique** integers and a target integer, find all pairs in the list that add up to the target.

Example

Given a list of integers `[1, 2, 3, 4, 5]` and a target integer `5`, the function should return `{(1, 4), (2, 3)}` (a set of tuples):

```
>>> find_pairs_naive([1, 2, 3, 4, 5], 5)
{(1, 4), (2, 3)}
```

Part 1: Implementation

Create a file `hw3.py`. In this file, you need to write two functions that behave as desribed above: `find_pairs_naive()` and `find_pairs_optimized()`.

Use test-driven development as you go - write unittests (using the unittest module) in a file `TestHw3.py`. As always, write your own tests - don't use any of the examples shown in this assignment.

- Test the correctness of the two functions:
  - Test the functions with different inputs and check if the output is correct.
  - Test the function with edge cases and different inputs to check the correctness of the functions. For example:
    - test case with empty list
    - test case with target = 0
    - test case with expected duplicate values as the target. For example if input list is `[1, 2, 3, 4, 5]` and target is `10`, the output should be an empty set `set()`
    - test case with target that equals to double number. For example, For example if input list is `[1, 2, 3, 4, 5]` and target is `6`, the output should be `{(2, 4), (1, 5)}`. Pair `(3,3)` should not be included.

`find_pairs_naive(lst, target)`

This function should iterate over the entire list using two nested loops to check for pairs that add up to the target.

- Input:
  - `lst`: a list of integers
  - `target`: an integer
- Output:
  - a set of tuples, each tuple containing two integers that add up to the target

Example:

```
>>> find_pairs_naive([6, 5, 2, 8, 9, 1], 7)
{(6, 1), (5, 2)}
```

`find_pairs_optimized(lst, target)`

This function should use a data structure to improve the time complexity of the algorithm.

- Input:
  - `lst`: a list of integers
  - `target`: an integer
- Output:
  - a set of tuples, each tuple containing two integers that add up to the target

Example:

```
>>> find_pairs_optimized([6, 5, 2, 8, 9, 1], 7)
{(6, 1), (5, 2)}
```

Part 2: Measure Time

`measure_min_time(fn, args)`

In the same file `hw3.py`, write a function `measure_min_time` that measures the minimum time of 10 runs for both functions using the `time` module.

- Input:
  - `fn`: a function (e.g. `find_pairs_naive` or `find_pairs_optimized`)
  - `args`: input for `fn`
- Output:
  - It calls the function 10 times and return the minimum time taken.

Test the function on different input size to check the performance of the function and how well it scales.

Measure running times

Add a short loop that uses `measure_min_time` to print out a table showing the time taken for both functions above as `n` increases. Round the time to 4 decimals, e.g. using `round(number, digits)` or the appropriate string formatting options (`f"{VARIABLE:.4f}"`). Format the table nicely, for example:

```
       n               naive           optimized
**************************************************
      10                   -                   -
      50                   -                   -
     100                   -                   -
     150                   -                   -
     200                   -                   -
     300                   -                   -
     500                   -                   -
--------------------------------------------------
```

Part 3: Time Complexity

We can often calculate the running time of a function by adding up the running times of every line. For instance, the below function has a O(n) running time to add up all the items in a list:

```
def sum(L):
    total = 0           # 1
    for item in L:      # n
        total += item   # 2 (add, then assign)
    return total        # 1
                        #--------------
                        # 1 + 2n + 1 = O(n)
```

- Add comments to `find_pairs_naive` and `find_pairs_optimized` that show line-by-line running times and the total sum, as in the example above
- Include a brief explanation of the results in comments under the function.

Submission

At a minimum, submit the following. you should also submit any other code required for your solution to run (e.g. any modules you write yourself and import in your solution)

- `hw3.py`
- `TestHw3.py`