

Program Structures & Algorithms

Spring 2021

Assignment No. 2

Task:

Insertion sort algorithm, find out the relationship between time complexity and different sizes of array.

Output:

- Output log
 - `/INFO6205/Python/test/test_util/output.log`
- Data points
 - `/INFO6205/Python/test/test_util/data_full_random.txt`
 - `/INFO6205/Python/test/test_util/data_ordered.txt`
 - `/INFO6205/Python/test/test_util/data_reverse_ordered.txt`
 - `/INFO6205/Python/test/test_util/data_partial_ordered.txt`

Relationship Conclusion:

When the size of array gets larger, the time complexity gets larger. In general, the relationship between array size (n) and time complexity (t) is roughly like $t = n^2$

When the array is ordered, the time complexity is smallest. When the array is reversed from the ordered one, the time complexity is largest.

Evidence to support the conclusion:

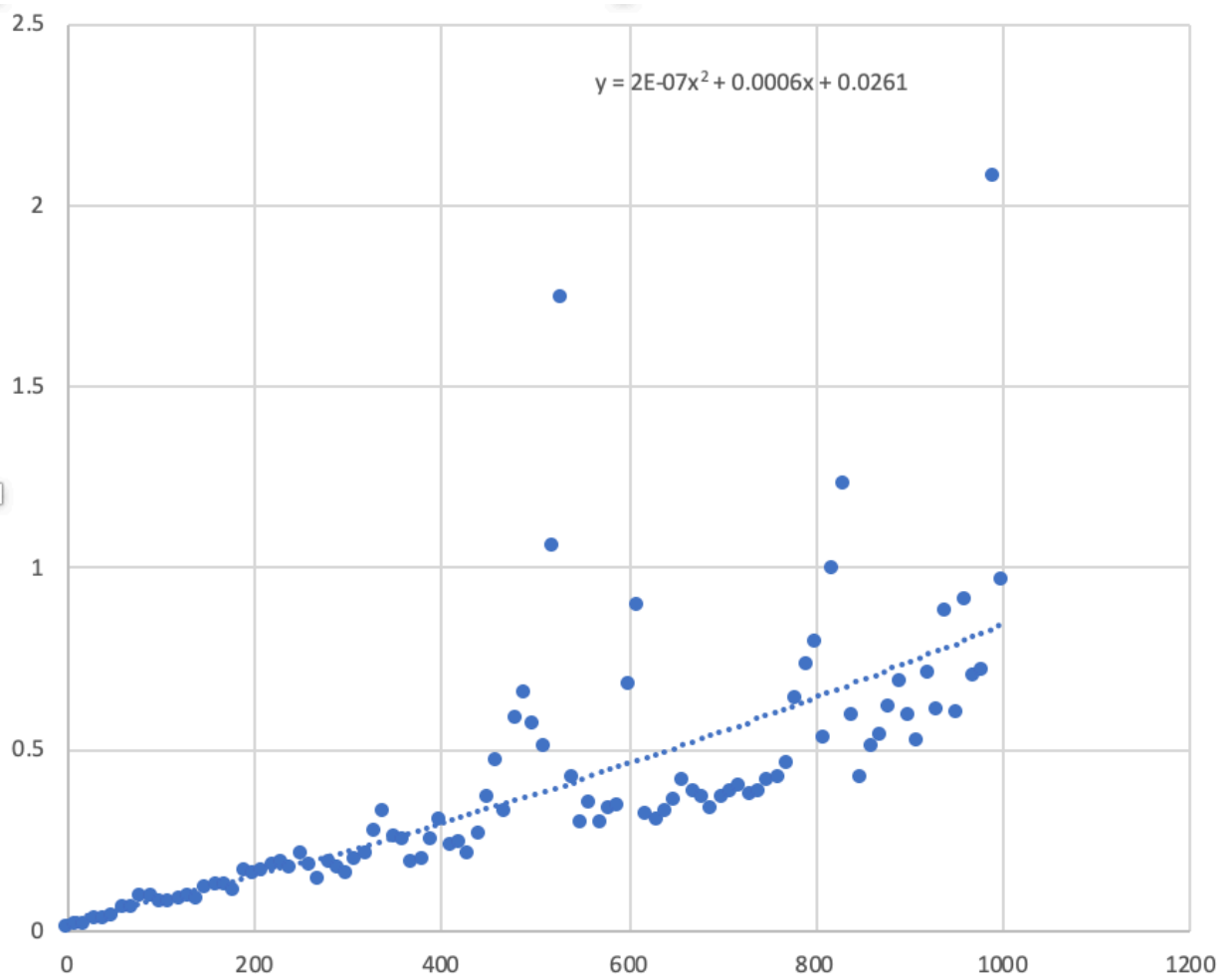
According to the output logging, the count of compares and the count of swaps increases as the size of array grows.

In all the types of array ordering, the count of compares only changes with the size of array. The count of swaps changes with both array size and ordering.

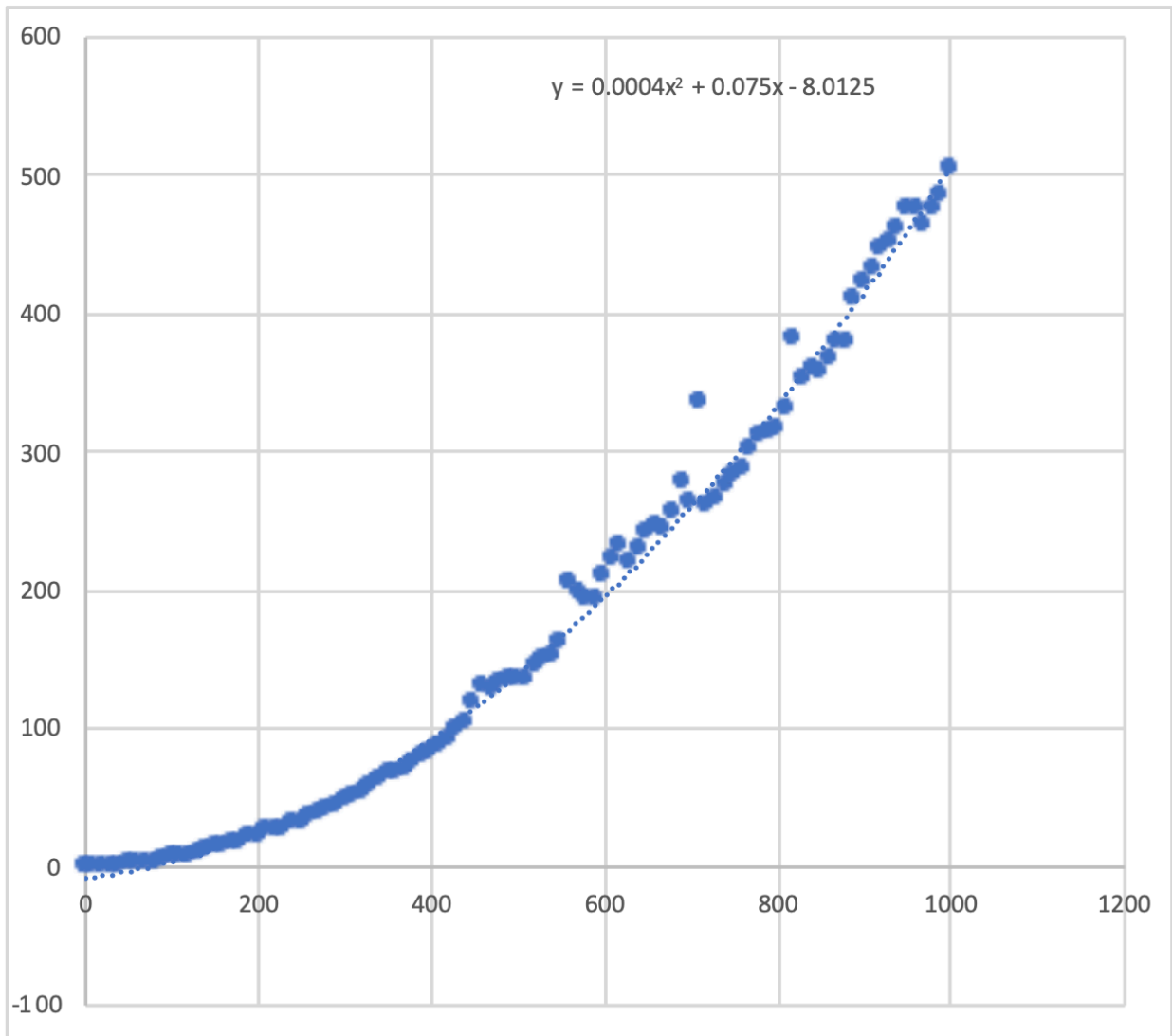
- In the ordered situation, the count of swaps is minimum (zero).
- In the reverse ordered situation, the count of swaps is maximum.

Graphical representation:

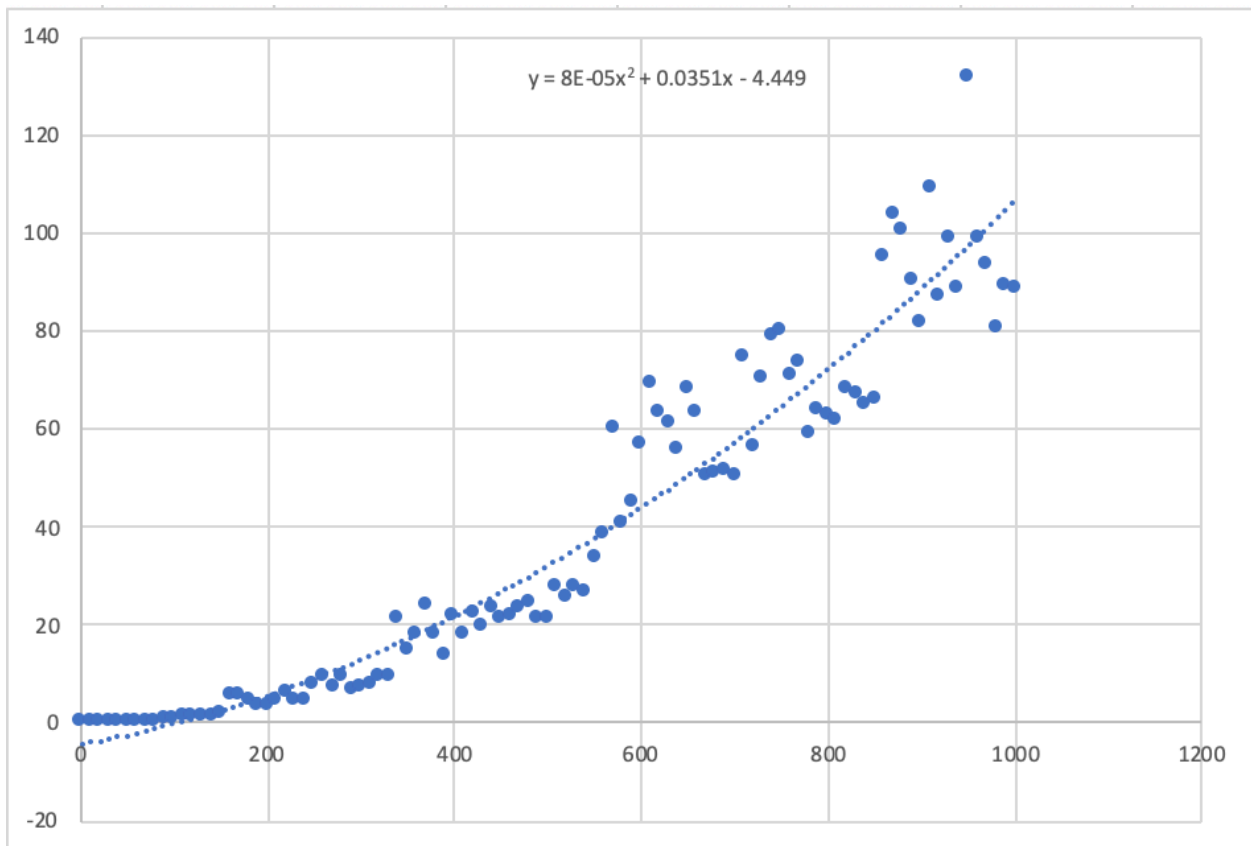
1. Ordered:



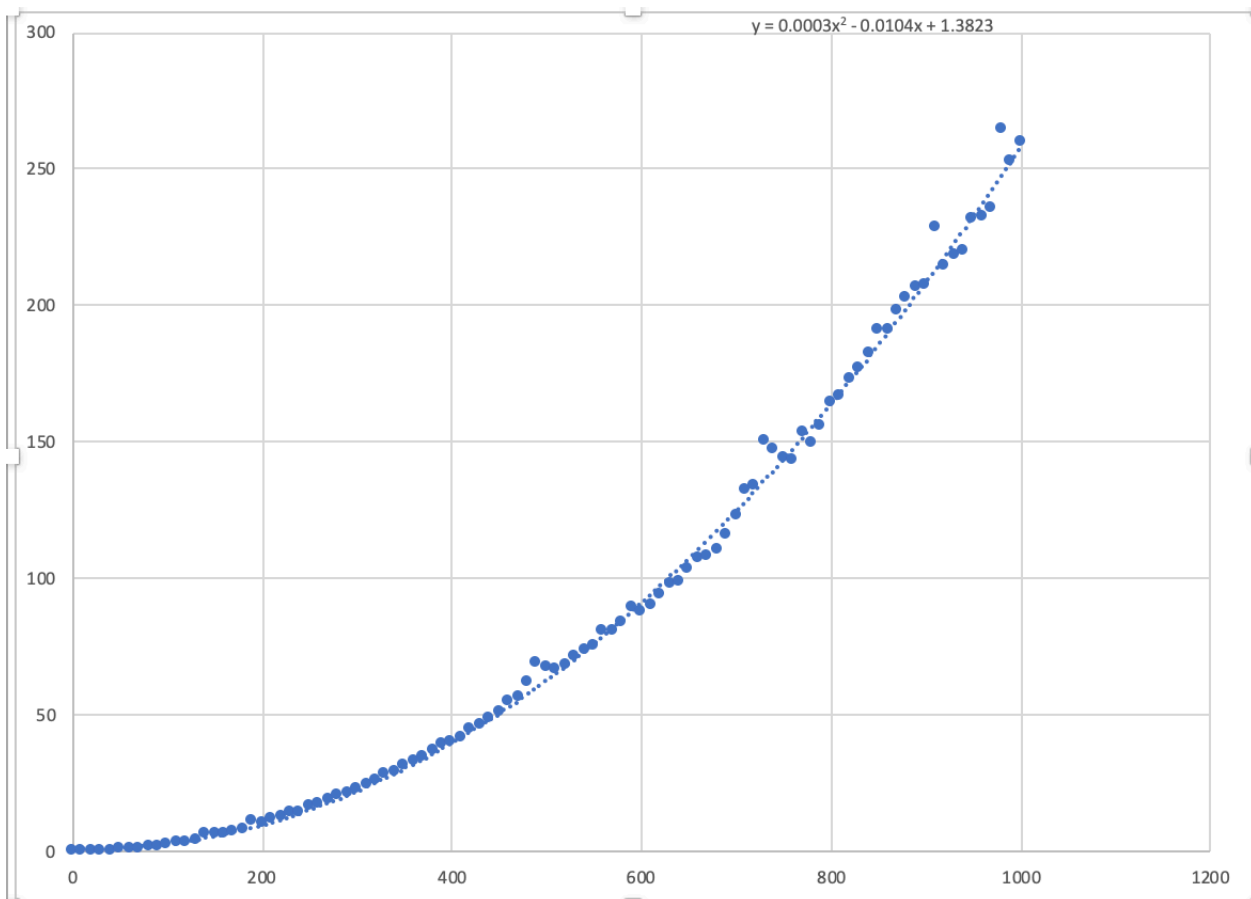
2. Reverse ordered:



3. Partial ordered:



4. Full random:



Unit tests result:

The screenshot displays an IDE with two panels. The top panel shows the execution of a benchmark script, and the bottom panel shows the results of unit tests for the same code.

Top Panel: Benchmark Results

```
INFO6205 Python src util benchmark.py
Project
  selection_sort.py
  sort.py
  __init__.py
  > union_find
  > util
    benchmark.py
    generic_type.py
  > test
    > test_bqs
    > test_functions
    > test_randomwalk
    > test_search

Run: benchmark
Helper for InsertionSort with 1000 elements: compares=241947, swaps=240954
Helper for InsertionSort with 1000 elements: compares=241947, swaps=240954
Helper for InsertionSort with 1000 elements: compares=241947, swaps=240954
Helper for InsertionSort with 1000 elements: compares=241947, swaps=240954
Helper for InsertionSort with 1000 elements: compares=241947, swaps=240954
Helper for InsertionSort with 1000 elements: compares=241947, swaps=240954
Helper for InsertionSort with 1000 elements: compares=241947, swaps=240954
Helper for InsertionSort with 1000 elements: compares=241947, swaps=240954
Helper for InsertionSort with 1000 elements: compares=241947, swaps=240954
Helper for InsertionSort with 1000 elements: compares=241947, swaps=240954
InsertionSort: 1000: 235.12794 millisecs
Process finished with exit code 0
```

Bottom Panel: Unit Test Results

```
INFO6205 Python test test_util test_benchmark.py
Project
  test_util
    assignment 2.xlsx
    data_full_random.txt
    data_ordered.txt
    data_partial_ordered.txt
    data_reverse_ordered.txt
    test_benchmark.py
    __init__.py
    poetry.lock
    pyproject.toml
    README.md
  > src
    > target

Run: Unittests in test_benchmark.py
Tests passed: 4 of 4 tests - 2 h 44 m 40 s 790 ms
Test Results 2 h 44 m 40 s 790 ms
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Helper for InsertionSort with 440 elements: compares=96580, swaps=96580
  Tests passed: 4
  Tests passed: 4 (moments ago)
```

The code in the background shows the implementation of the InsertionSort algorithm and the test functions used to verify its correctness.

```
1 from random import Random
2 from typing import Callable, List, Generic, Union
3
4 import sys
5 import time
6
7 from sort.simple.insertion_sort import InsertionSort
8 from sort.simple.sort import Sort
9 from util.generic_type import C
10
11 class Timer:
12     def __init__(self) -> None:
13         if __name__ == "__main__": for k in range(5)
```

```
27 xs = list(range(n))
28 xs.reverse()
29 return xs
30
31 gen_data(fn_xs, 'data_reverse_ordered')
32
33 def test_partial_ordered(self)->None:
34     def fn_xs(n: int) -> List[int]:
35         xs_1 = list(range(n//2))
36         random = Random()
37         xs_2 = [random.randint(0, int(1e5)) for _ in range(n - n//2)]
38         xs = xs_1 + xs_2
```