Computational Thinking with Algorithms - Benchmarking in Python

Dominic Carr

Atlantic Technological University

dominic.carr@atu.ie

Overview

- Motivation for benchmarking
- Time in Python
- Benchmarking a single run
- Benchmarking multiple statistical runs

Motivation

- Also known as a posteriori analysis
- Empirical method to compare relative performance of algorithm implementations
- Experimental (e.g. running time) data can be used to validate theoretical / a priori algorithm analysis
- Can effect running time:
    - System Architecture
    - CPU design
    - Operating System
    - Background Processes
    - Energy saving / performance enhancing technologies
- It is prudent to conduct multiple runs of the same experimental setup to ensure you get a representative sample

Time in Python I

- Dates and times in Python are represented as the number of seconds that have elapsed since midnight on January 1st 1970 (the “Unix Epoch”)
- Each second since the Unix Epoch has a specific timestamp
- Can import the time module in Python to work with dates and times
    - e.g. start_time = time.time gets the current time in seconds
    - e.g. **1555001605** is Thursday, 11 April 2019 16:53:25 in GMT

Time in Python II

Listing 1: Benchmarking a single run

```python
import timeit
import time

def functionA():
    print("Function A starts the execution:")
    print("Function A completes the execution:")

start_time = timeit.default_timer()
functionA()
print(timeit.default_timer() - start_time)
```

Time in Python III

Listing 2: Benchmarking multiple runs

```python
num_runs = 10
results = []
for r in range(num_runs):
    start_time = timeit.default_timer()

    ##################################
    # call the function to benchmark #
    ##################################

    end_time = timeit.default_timer()
    time_elapsed = end_time - start_time
    results.append(time_elapsed);

print(results)
```

Time in Python IV

- The function random_array() takes as input a value n and returns an array of n randomly generated integers with values between 0 and 99
- You may use this code to generate random input instances which can be used when benchmarking your chosen sorting algorithms. Note you must import randint from Python's random module [1]

```python
def random_array(n):
    array = []
    for i in range(0, n, 1):
        array.append(randint(0, 100))
    return array
```

[1] https://docs.python.org/3/library/random.html

The End