TaDa it’s Magic!

Predicting the Performance of Programs through Automated Doubling Experiments

Gregory M. Kapfhammer, Lancaster Wu, Enpu You
CodepaLOUsa 2021
Huh, what is this about?

Key Questions

Can a tool **automatically predict** a program’s performance? Is it possible to automatically estimate the **worst-case time complexity** of a program?

Intended Audience

An **adventuresome** technology enthusiast who wants to explore how a new approach to **performance evaluation** can make their **programs faster**!

Let's learn how to predict a function's performance!
Why focus on Python programming?

Prevalence of Python

Python is consistently ranked as one of the top programming languages for web development, data science, machine learning, and general programming.

Importance of Performance

Programmers who create, say, serverless functions with AWS Lambda need to carefully monitor and improve the performance of these functions.

Challenging about performance evaluation in Python?
Analytical Evaluation

Experimental Evaluation

What are the trade-offs of these two approaches?
Analytical

- Provides a clear means by which to compare programs
- Does not depend on the hardware or software configuration
- Yet, often requires precise mathematical reasoning skills

Experimental

- Must generate inputs to the program subject to experiments
- Must repeatedly run a program and collect performance data
- Only generally accessible to programmers if good tools exist

Analysis characterizes an algorithm as, say, \( O(n) \)

Experiments run program to collect performance data
How to analytically evaluate a program's performance?

- Commonly used growth functions
- Study program's code constructs
"Fast" Order of Growth Functions
"Slow" Order of Growth Functions
Relationship between growth function and program's performance?

- Slow growth functions → fast programs
- Fast growth functions → slow programs
Analyzing the `add_digits` Function

```python
def add_digits(digits: str) -> int:
    value = 0
    for digit in digits:
        value += int(digit)
    return value

sum_digits = add_digits("123")
print(sum_digits)
```

What is worst-case time complexity of `add_digits`?
Analyzing the `factorial` Function

def factorial(x: int) -> int:
    if x == 1:
        return 1
    else:
        return x*factorial(x-1)

factorial_value = factorial(3)
print(factorial_value)

What is worst-case time complexity of `factorial`?
Analyzing the `is_subset` Function

```python
def is_subset(one: List, two: List) -> bool:
    matched = False
    for element_one in one:
        for element_two in two:
            if element_one == element_two:
                matched = True
                break
        if not matched:
            return False
    return True
```

What is worst-case time complexity of
Run an experiment to get likely worst-case time complexity of program?

- Bespoke auto-doubling experiment tool
- TaDa auto-doubling for a Python function
Doubling Experiment: Linear

Double the size of the program’s input

14.98 seconds

31.45 seconds

Doubling ratio is approximately 2

Likely worst-case time complexity is $O(n)$
Doubling Experiment: Quadratic

Double the size of the program’s input

Doubling ratio is approximately 4

Likely worst-case time complexity is $O(n^2)$
Doubling Experiment: Cubic

Double the size of the program’s input

Doubling ratio is approximately 8

Likely worst-case time complexity is $O(n^3)$
What are challenges with running automated doubling experiments?

- Automatically generate inputs to the function
- Determine when to stop running experiments
- Establish a statistical confidence in the prediction
TaDa Runs a Doubling Experiment

Input is a Python function and configuration options

Output is a data table and a performance prediction

See [Tada-Project/tada](https://github.com/Tada-Project/tada) for details
Analyzing the `insertion_sort` Function

```python
def insertion_sort(lst: list[int]) -> list[int]:
    for i in range(1, len(lst)):
        value = lst[i]
        pos = i
        while pos > 0 and value < lst[pos - 1]:
            lst[pos] = lst[pos - 1]
            pos -= 1
        lst[pos] = value
    return lst
```

Can TaDa predict worst-case of `insertion_sort`?
Analyzing the `bubble_sort` Function

```python
def bubble_sort(lst: list[int]) -> list[int]:
    for num in range(len(lst) - 1, 0, -1):
        for i in range(num):
            if lst[i] > lst[i + 1]:
                temp = lst[i]
                lst[i] = lst[i + 1]
                lst[i + 1] = temp

    return lst
```

Can TaDa predict worst-case of `bubble_sort`?
How to automatically generate function inputs during experiments?

- Hypothesis: Property-based testing tool
- JSON Schema: Describe format of input
Hypothesis and JSON Schema for Data

```json
[
{
  "type": "array",
  "items": {
    "type": "integer"
  },
  "uniqueItems": true,
  "maxItems": 0,
  "minItems": 0
},
...
```

Describe structure to support automated data generation
TaDa’s Automated Analysis of Insertion Sort

<table>
<thead>
<tr>
<th>Size</th>
<th>Mean</th>
<th>Median</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>3.644364811706543e-06</td>
<td>3.498709533691405e-06</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>6.535123836263021e-06</td>
<td>6.483351989746092e-06</td>
<td>1.793213405878218</td>
</tr>
<tr>
<td>100</td>
<td>1.2902192108154296e-05</td>
<td>1.2540842590332028e-05</td>
<td>1.9742842571032526</td>
</tr>
<tr>
<td>200</td>
<td>2.5023900944010416e-05</td>
<td>2.4608139038085928e-05</td>
<td>1.9395077002608803</td>
</tr>
<tr>
<td>400</td>
<td>5.526396857910156e-05</td>
<td>5.35152070731250005e-05</td>
<td>2.2084473840729952</td>
</tr>
<tr>
<td>800</td>
<td>0.00011801120257161459</td>
<td>0.00011251379296875</td>
<td>2.1354094829925283</td>
</tr>
</tbody>
</table>

Interpreting TaDa’s output:
- Ran multiple threads for multiple input sizes
- Doubled the input size and recorded time
- Used ratio to correctly predict worst-case
### TaDa’s Comparison of Sorting Functions

**bubble_sort: O(n^2) quadratic**

<table>
<thead>
<tr>
<th>Size</th>
<th>Mean</th>
<th>Median</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>2.877628824869792e-05</td>
<td>2.846207250976562e-05</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>0.00010703222574869792</td>
<td>0.00010308191601562499</td>
<td>3.7194796562140504</td>
</tr>
<tr>
<td>100</td>
<td>0.0004109644687825521</td>
<td>0.00039437410449218743</td>
<td>3.8396330255474633</td>
</tr>
<tr>
<td>200</td>
<td>0.0015730586140625</td>
<td>0.0015326660937500002</td>
<td>3.8277241308051635</td>
</tr>
<tr>
<td>400</td>
<td>0.00632440301875</td>
<td>0.006229572156249999</td>
<td>4.020449690947576</td>
</tr>
<tr>
<td>800</td>
<td>0.02929213468333335</td>
<td>0.028519337000000006</td>
<td>4.631604690038055</td>
</tr>
</tbody>
</table>

At the greatest common size 800:

Mean: insertion_sort is 99.60% faster than bubble_sort

Median: insertion_sort is 99.61% faster than bubble_sort

---

Correct worst-case predictions and empirical insights
Performance Evaluation

TaDa tool bridges the experimental and analytical!

- Analytical study of performance is challenging
- Experimental study requires data and tooling
- TaDa runs doubling experiments and predicts
Tool Development with Python

TaDa makes it easy to run doubling experiments!

See [Tada-Project/tada](https://www.gregorykapfhammer.com/) for details.

https://www.gregorykapfhammer.com/

gkapfham/codepalousa2021-presentation-tada