University of Stavanger

PREBEN ANDERSEN, LISAN KURUPPU AND SEBASTIAN GRØNBECH
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

DAT600 Assignmnet 1 - Group 19

Technical Report - Data Science - February 2026

I, **Preben Andersen, Lisan Kuruppu and Sebastian Grønbech**, declare that this thesis titled, “DAT600 Assignmnet 1 - Group 19” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master’s degree at the University of Stavanger.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

https://github.com/SebastianGronbech/dat600-assignments

Contents

Chapter 1

Introduction

Page budget for Introduction: 3-5 pages.

1.1 Background and Motivation

Sorting algorithms are a fundamental topic in computer science and are widely used in applications such as databases, search engines, and data processing systems. Because sorting often serves as a building block for more complex algorithms, its efficiency has a direct impact on overall system performance.

From an educational perspective, sorting algorithms provide clear examples for studying algorithm design and runtime analysis. Classical algorithms such as insertion sort, merge sort, heap sort, and quicksort illustrate different algorithmic paradigms and exhibit well-known asymptotic time complexities. These properties make them suitable for both theoretical analysis and empirical evaluation.

While asymptotic analysis describes how algorithms scale with input size, it abstracts away constant factors and implementation details. In practice, these factors can significantly affect actual runtime performance. Therefore, combining theoretical analysis with experimental evaluation is essential for understanding algorithm behavior on real hardware and in different programming languages.

This assignment integrates implementation, empirical measurement, and formal analysis to provide a comprehensive study of sorting algorithms and divide-and-conquer recurrences.

1.2 Objectives

The objectives of this study are:

- To implement classical sorting algorithms and analyze their asymptotic behavior.
- To empirically validate theoretical runtime complexities using step counting.
- To compare actual execution times across different programming languages.
- To apply formal proof techniques to analyze asymptotic growth and recursive relations.

1.3 Approach and Contributions

This study combines implementation, experimentation, and theoretical analysis to investigate algorithm performance.

First, four sorting algorithms are implemented in Python and instrumented to count fundamental operations. Step counts are recorded for increasing input sizes and compared with expected worst-case time complexities.

Second, insertion sort is implemented in both Python and Go, and execution times are measured for varying input sizes. This comparison highlights the effect of programming language choice and runtime overhead on practical performance.

Finally, formal asymptotic proofs are presented, including little-o proofs and divide-and-conquer runtime analysis using the Master Theorem and the substitution method.

The main contributions of this work are:

- Python implementations of classical sorting algorithms with step-count analysis.
- An empirical comparison of execution time between Python and Go.
- Formal proofs of asymptotic relationships and recursive runtime bounds.

1.4 Outline

The remainder of this report is organized as follows:

- Task 1-2 presents the implementation of sorting algorithms and empirical results from step-count analysis, followed by a comparison of real execution times across programming languages and formal asymptotic proofs
- Task 3 focuses on basic asymptotic proofs including demonstrations of little-o relationship
- Task 4 analyzes divide-and-conquer recurrences using the Master Theorem
- Task 5 applies the substitution method to a recursive relation and evaluates which inductive hypotheses hold.

Together, these chapters provide a structured progression from empirical analysis to formal mathematical reasoning.

Chapter 2

Tasks

2.1 Task 1 - Counting the steps

2.1.1 Description

The goal of this task is to implement and empirically evaluate the asymptotic running time of four sorting algorithms (Insertion Sort, Merge Sort, Heap Sort, and Quick Sort) by counting their executed steps for increasing input sizes and comparing the results with their expected worst-case complexities.

2.1.2 Method

Each algorithm was implemented in Python. For each input size, we counted the number of comparisons and swaps as steps. Input collections were randomly generated. The step count were plotted as a function of input size.

2.1.3 Results

As shown in Figure 2.1, the number of steps increases roughly logarithmically with input size for heap sort and merge sort, consistent with their $n \log(n)$ theoretical complexity. While for the quick sort and insertion sort, the theoretical complexity is $\Theta(n)$. Quick sort show deviations due to pivot selection.

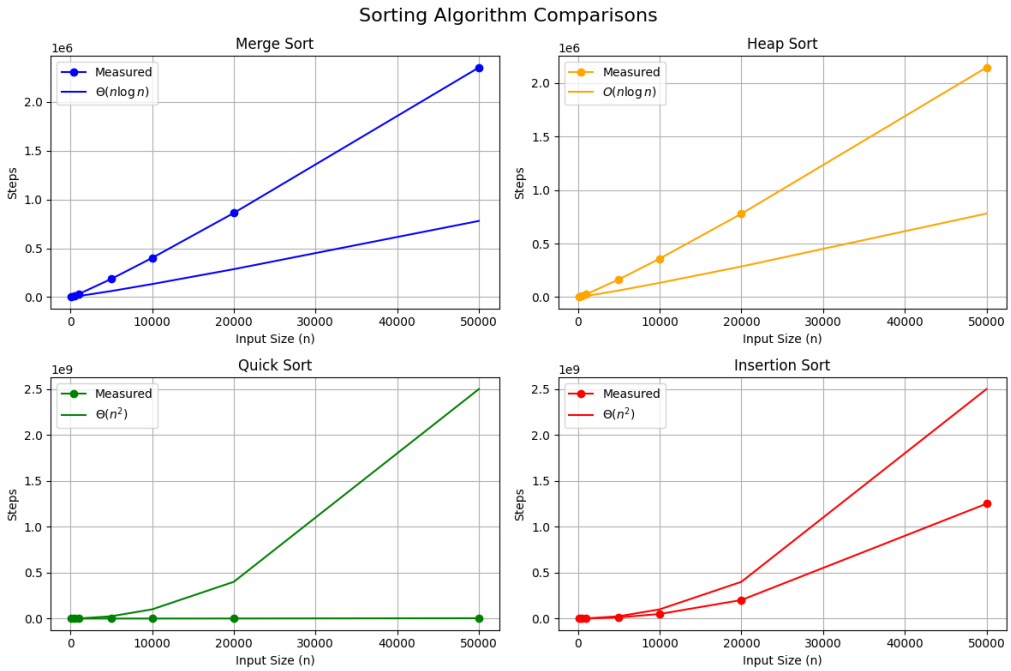

Figure 2.1: Step count vs input size for the sorting algorithms.

2.2 Task 2 - Compare true execution time

2.2.1 Description

In this task, we compare the real execution time of insertion sort implemented in both Python and Go. By running the algorithms on inputs of increasing size, we investigate how language and implementation affect practical performance.

2.2.2 Method

For each input size, an array of integers was generated with values drawn uniformly from the range $[0, 1000]$. The insertion sort algorithm was then applied to each array, and its execution time was measured. In Python, the execution time was recorded using `time.perf_counter_ns()`, while in Go, benchmarking was performed using `testing.B`.

Input size (n)	Python (ns)	Go (ns)	Python / Go
100	310 541	1 435	216.41
500	7 063 625	29 865	236.52
1000	22 924 666	117 853	194.52
5000	456 371 458	2 704 282	168.76
10000	1 785 091 167	10 827 533	164.87
20000	7 134 526 417	4 366 4641	163.39
50000	45 739 860 916	283 365 521	161.42

Table 2.1: Insertion Sort runtime comparison between Python (`time.perf_counter_ns()`) and Go (`go test -bench`) on random arrays

2.2.3 Results

As shown in Table 2.1 results how that Go outperforms Python for all input sizes, and Go becomes faster as the input size increases.

2.3 Task 3: Basic Proofs

2.3.1 Equation 1: $\frac{n^2}{\log n} = o(n^2)$

We use the definition of little-o:

$$f(n) = o(g(n)) \quad \iff \quad \forall c > 0 \; \exists n_0 \text{ such that } 0 \leq f(n) < c\, g(n) \text{ for all } n \geq n_0.$$

A common equivalent criterion is the limit test:

$$f(n) = o(g(n)) \quad \iff \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0,$$

provided $g(n) > 0$ for sufficiently large n.

Let

$$f(n) = \frac{n^2}{\log n}, \qquad g(n) = n^2,$$

where $\log n$ denotes a logarithm of any fixed base > 1 (the base does not matter for asymptotic classes).

Compute the ratio:

$$\frac{f(n)}{g(n)} = \frac{\frac{n^2}{\log n}}{n^2}.$$

Cancel n^2:

$$\frac{f(n)}{g(n)} = \frac{1}{\log n}.$$

Now evaluate the limit:

$$\lim_{n\to\infty} \frac{1}{\log n}.$$

Since $\log n \to \infty$ as $n \to \infty$, its reciprocal goes to 0:

$$\log n \to \infty \quad \Longrightarrow \quad \frac{1}{\log n} \to 0.$$

Therefore,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \quad \Longrightarrow \quad \frac{n^2}{\log n} = o(n^2).$$

2.3.2 Equation 2: $n^2 \neq o(n^2)$

Again, using the limit characterization:

$$f(n) = o(g(n)) \Longleftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

Set $f(n) = n^2$ and $g(n) = n^2$. Then:

$$\frac{f(n)}{g(n)} = \frac{n^2}{n^2} = 1 \quad \text{for all } n \neq 0.$$

Hence the limit is:

$$\lim_{n\to\infty} \frac{n^2}{n^2} = \lim_{n\to\infty} 1 = 1 \neq 0.$$

So n^2 is *not* little-o of n^2:

$$n^2 \neq o(n^2).$$

Little-o means "grows strictly slower than." But n^2 and n^2 grow at the same rate, so it cannot be strictly slower. In fact, $n^2 = \Theta(n^2)$.

2.4 Task 4: Divide and Conquer Analysis

2.4.1 Master Theorem: $T(n) = 16T(n/4) + n$

We match the recurrence to the Master theorem form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

where here

$$a = 16, \quad b = 4, \quad f(n) = n.$$

Compute the critical exponent:

$$n^{\log_b a} = n^{\log_4 16}.$$

Since $16 = 4^2$, we have $\log_4 16 = 2$, so:

$$n^{\log_4 16} = n^2.$$

Now compare $f(n)$ with $n^{\log_b a}$:

$$f(n) = n, \qquad n^{\log_b a} = n^2.$$

We can write:

$$n = O(n^{2-\varepsilon}) \quad \text{with } \varepsilon = 1.$$

So this is Master Theorem **Case 1**, therefore:

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^2).$$

2.4.2 Substitution method: $T(n) = 4T(n/2) + n$

We now analyze:

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

using induction with different hypotheses. Assume n is a power of 2 for simplicity (standard in these proofs), and assume a constant base case $T(1) = d$ for some $d > 0$.

Hypothesis 1: $T(n) \leq cn^2$ **for some** $c > 0$

Inductive attempt. Assume (inductive hypothesis) that for all smaller inputs:

$$T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)^2 = \frac{cn^2}{4}.$$

Plug into the recurrence:

$$T(n) = 4T\left(\frac{n}{2}\right) + n \leq 4 \cdot \frac{cn^2}{4} + n = cn^2 + n.$$

But to conclude $T(n) \leq cn^2$, we would need:

$$cn^2 + n \leq cn^2,$$

which implies $n \leq 0$, false for $n > 0$.

Conclusion. This hypothesis **fails** because the extra $+n$ term prevents the inequality from closing. So **Hypothesis 1 is not valid** as stated.

Hypothesis 2: $T(n) \geq cn^2$ **for some** $c > 0$

Inductive proof: Assume:

$$T\left(\frac{n}{2}\right) \geq c\left(\frac{n}{2}\right)^2 = \frac{cn^2}{4}.$$

Then:

$$T(n) = 4T\left(\frac{n}{2}\right) + n \geq 4 \cdot \frac{cn^2}{4} + n = cn^2 + n \geq cn^2.$$

Base case condition: We must ensure the base case fits:

$$T(1) = d \geq c \cdot 1^2 \quad \iff \quad c \leq d.$$

So if we pick any c with $0 < c \leq d$, the induction holds.

Conclusion. **Hypothesis 2 holds** (with a suitable choice of c consistent with the base case).

Hypothesis 3: $T(n) \leq cn^2 - bn$ **for** $c > 0$**,** $b > 0$

This is a common trick: we strengthen the upper bound by subtracting a linear term so the $+n$ in the recurrence can be "absorbed."

Inductive attempt. Assume:

$$T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)^2 - b\left(\frac{n}{2}\right) = \frac{cn^2}{4} - \frac{bn}{2}.$$

Substitute into the recurrence:

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n \\ &\leq 4\left(\frac{cn^2}{4} - \frac{bn}{2}\right) + n \\ &= cn^2 - 2bn + n \\ &= cn^2 - (2b-1)n. \end{aligned}$$

To conclude

$$T(n) \leq cn^2 - bn,$$

it is sufficient that

$$cn^2 - (2b-1)n \leq cn^2 - bn.$$

Cancel cn^2 from both sides:

$$-(2b-1)n \leq -bn.$$

Multiply by -1 (reverses inequality):

$$(2b-1)n \geq bn.$$

For $n > 0$, divide by n:

$$2b - 1 \geq b \quad \Longleftrightarrow \quad b \geq 1.$$

Base case condition. We also need the base case to satisfy the bound:

$$T(1) = d \leq c \cdot 1^2 - b \cdot 1 = c - b.$$

So we need $c \geq d + b$ (choose c large enough).

Conclusion. **Hypothesis 3 holds** provided we choose parameters such that:

$$b \geq 1 \quad \text{and} \quad c \geq d + b.$$

Which hypothesis is "correct" and why?

- Hypothesis 1 ($T(n) \leq cn^2$) **fails** because the $+n$ term creates $cn^2 + n$, which cannot be bounded by cn^2.
- Hypothesis 2 ($T(n) \geq cn^2$) **holds** because adding $+n$ only makes the expression larger, so the lower bound closes easily.
- Hypothesis 3 ($T(n) \leq cn^2 - bn$) **holds** (for $b \geq 1$ and sufficiently large c) because the negative linear term provides "slack" that absorbs the $+n$ in the recurrence.

Taken together, Hypothesis 2 and Hypothesis 3 imply:

$$T(n) = \Theta(n^2).$$

This matches the Master theorem result for $T(n) = 4T(n/2) + n$ (since $a = 4$, $b = 2$, and $f(n) = n = O(n^{2-\varepsilon})$).

2.5 Task 5: Substitution Method Analysis

We consider

$$T(n) = 4T(n/2) + n, \tag{2.1}$$

where n is assumed to be the power of 2 for simplification

Runtime Complexity

We expand the recurrence for k levels:

$$T(n) = 4T(n/2) + n \tag{2.2}$$
$$= 4\left(4T(n/4) + \frac{n}{2}\right) + n \tag{2.3}$$
$$= 4^2 T(n/4) + 2n + n \tag{2.4}$$
$$= 4^2 T(n/4) + 3n \tag{2.5}$$
$$= 4^3 T(n/8) + 3n. \tag{2.6}$$

By induction on the expansion, after k steps we obtain

$$T(n) = 4^k\, T\!\left(\frac{n}{2^k}\right) + kn. \tag{2.7}$$

We stop expanding when $n/2^k = 1$, i.e. when $k = \log_2 n$. Substituting gives

$$T(n) = 4^{\log_2 n}\, T(1) + n \log_2 n. \tag{2.8}$$

Since

$$4^{\log_2 n} = (2^2)^{\log_2 n} = 2^{2\log_2 n} = n^2, \tag{2.9}$$

it follows that

$$T(n) = n^2 T(1) + n \log_2 n = \Theta(n^2). \tag{2.10}$$

Which Inductive Hypotheses Hold?

We examine the following hypotheses.

1) $T(n) \le cn^2$ **for** $c > 0$**.** Assume inductively that $T(n/2) \le c(n/2)^2 = cn^2/4$. Then

$$T(n) = 4T(n/2) + n \le 4 \cdot \frac{cn^2}{4} + n = cn^2 + n, \tag{2.11}$$

which is not $\le cn^2$ for $n > 0$. Hence, this hypothesis does not close.

2) $T(n) \geq cn^2$ **for** $c > 0$**.** Assume inductively that $T(n/2) \geq c(n/2)^2 = cn^2/4$. Then

$$T(n) = 4T(n/2) + n \geq 4 \cdot \frac{cn^2}{4} + n = cn^2 + n \geq cn^2. \tag{2.12}$$

Thus, this hypothesis holds (given a suitable base case).

3) $T(n) \leq cn^2 - bn$ **for** $c > 0$ **and** $b > 0$**.** Assume inductively that

$$T(n/2) \leq c(n/2)^2 - b(n/2) = \frac{cn^2}{4} - \frac{bn}{2}. \tag{2.13}$$

Then

$$\begin{aligned} T(n) &= 4T(n/2) + n && (2.14)\\ &\leq 4\Big(\frac{cn^2}{4} - \frac{bn}{2}\Big) + n && (2.15)\\ &= cn^2 - 2bn + n = cn^2 - (2b-1)n. && (2.16) \end{aligned}$$

To ensure $T(n) \leq cn^2 - bn$, it suffices that

$$(2b-1)n \geq bn \quad \Longleftrightarrow \quad 2b - 1 \geq b \quad \Longleftrightarrow \quad b \geq 1. \tag{2.17}$$

Therefore, the hypothesis holds provided $b \geq 1$ (and a suitable base case).

Conclusion: $T(n) = \Theta(n^2)$. Hypothesis (1) does not hold. Hypothesis (2) holds. Hypothesis (3) holds if $b \geq 1$.

University of Stavanger

4036 Stavanger
Tel: +47 51 83 10 00
E-mail: post@uis.no
www.uis.no