Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

CHAI
Humanities-Centered AI

Transformer-based Algorithmics

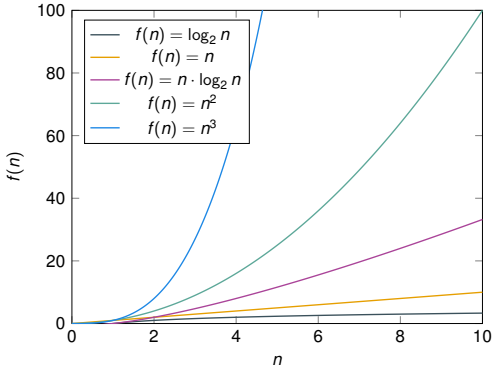
Malte Luttermann – Institute for Humanities-Centered Artificial Intelligence (CHAI)

April 15, 2026

Version from April 20, 2026

Topics for Today

(1) Runtime comparison and *O*-notation
(2) Insertion sort
(3) Selection sort



Acknowledgement

- Some of the upcoming content is taken from the following lectures and corresponding exercises and has been translated and partially modified:
  - Prof. Dr. Ralf Möller: »Algorithmen und Datenstrukturen – Vorlesung«
  - Dr. Magnus Bender: »Algorithmen und Datenstrukturen – Übung«

Exercise 1 – Runtime Comparison

Given the two program runtimes $T_1(n) = n^2 + 20n$ and $T_2(n) = 200n$.

(1) Classify both using O-notation.
(2) For which input sizes is each function the better choice?

Exercise 1 – Solution

Classification:

- $T_1(n) = n^2 + 20n \in O(n^2)$

Exercise 1 – Solution

Classification:

- $T_1(n) = n^2 + 20n \in O(n^2)$
- $T_2(n) = 200n \in O(n)$

Exercise 1 – Solution

Classification:

- $T_1(n) = n^2 + 20n \in O(n^2)$
- $T_2(n) = 200n \in O(n)$

Crossover point:

$$n^2 + 20n = 200n$$
$$n^2 + 20n - 200n = 0$$
$$n^2 - 180n = 0$$
$$n(n - 180) = 0$$

Exercise 1 – Solution

Classification:

- $T_1(n) = n^2 + 20n \in O(n^2)$
- $T_2(n) = 200n \in O(n)$

Crossover point:

$$n^2 + 20n = 200n$$
$$n^2 + 20n - 200n = 0$$
$$n^2 - 180n = 0$$
$$n(n - 180) = 0$$

- $n < 180$: T_1 faster
- $n > 180$: T_2 faster

Exercise 1 – Solution

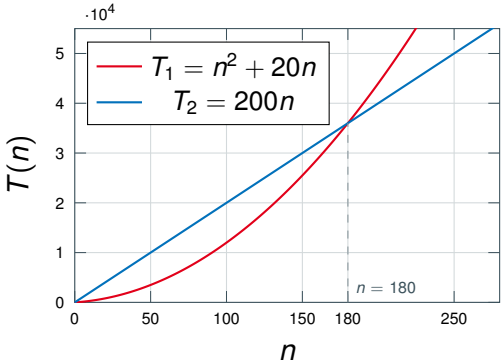
Classification:

- $T_1(n) = n^2 + 20n \in O(n^2)$
- $T_2(n) = 200n \in O(n)$

Crossover point:

$$n^2 + 20n = 200n$$
$$n^2 + 20n - 200n = 0$$
$$n^2 - 180n = 0$$
$$n(n - 180) = 0$$

- $n < 180$: T_1 faster
- $n > 180$: T_2 faster



Large constant factors matter for small n;
asymptotic class dominates for large n.

Exercise 2 – Asymptotic Complexity

Given the following functions, classify them using O-notation.

(1) $T_1(n) = 3n^3 + 2n^2 + 5$
(2) $T_2(n) = 1000n \cdot \log_2 n + 50n$
(3) $T_3(n) = n^2 + n^3 + n$
(4) $T_4(n) = n \cdot \log_2 n + 1000n^2$
(5) $T_5(n) = 10n^2 + 100n \cdot \log_2 n + 1000$

Exercise 2 – Solution

(1) $T_1(n) = 3n^3 + 2n^2 + 5$
(2) $T_2(n) = 1000n \cdot \log_2 n + 50n$
(3) $T_3(n) = n^2 + n^3 + n$
(4) $T_4(n) = n \cdot \log_2 n + 1000n^2$
(5) $T_5(n) = 10n^2 + 100n \cdot \log_2 n + 1000$

O-notation classification:

(1) $T_1(n) \in O(n^3)$

Exercise 2 – Solution

(1) $T_1(n) = 3n^3 + 2n^2 + 5$
(2) $T_2(n) = 1000n \cdot \log_2 n + 50n$
(3) $T_3(n) = n^2 + n^3 + n$
(4) $T_4(n) = n \cdot \log_2 n + 1000n^2$
(5) $T_5(n) = 10n^2 + 100n \cdot \log_2 n + 1000$

O-notation classification:

(1) $T_1(n) \in O(n^3)$
(2) $T_2(n) \in O(n \cdot \log_2 n)$

Exercise 2 – Solution

(1) $T_1(n) = 3n^3 + 2n^2 + 5$
(2) $T_2(n) = 1000n \cdot \log_2 n + 50n$
(3) $T_3(n) = n^2 + n^3 + n$
(4) $T_4(n) = n \cdot \log_2 n + 1000n^2$
(5) $T_5(n) = 10n^2 + 100n \cdot \log_2 n + 1000$

O-notation classification:

(1) $T_1(n) \in O(n^3)$
(2) $T_2(n) \in O(n \cdot \log_2 n)$
(3) $T_3(n) \in O(n^3)$

Exercise 2 – Solution

(1) $T_1(n) = 3n^3 + 2n^2 + 5$
(2) $T_2(n) = 1000n \cdot \log_2 n + 50n$
(3) $T_3(n) = n^2 + n^3 + n$
(4) $T_4(n) = n \cdot \log_2 n + 1000n^2$
(5) $T_5(n) = 10n^2 + 100n \cdot \log_2 n + 1000$

O-notation classification:

(1) $T_1(n) \in O(n^3)$
(2) $T_2(n) \in O(n \cdot \log_2 n)$
(3) $T_3(n) \in O(n^3)$
(4) $T_4(n) \in O(n^2)$

Exercise 2 – Solution

(1) $T_1(n) = 3n^3 + 2n^2 + 5$
(2) $T_2(n) = 1000n \cdot \log_2 n + 50n$
(3) $T_3(n) = n^2 + n^3 + n$
(4) $T_4(n) = n \cdot \log_2 n + 1000n^2$
(5) $T_5(n) = 10n^2 + 100n \cdot \log_2 n + 1000$

O-notation classification:

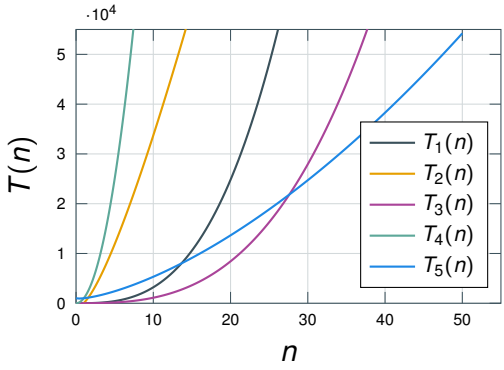
(1) $T_1(n) \in O(n^3)$
(2) $T_2(n) \in O(n \cdot \log_2 n)$
(3) $T_3(n) \in O(n^3)$
(4) $T_4(n) \in O(n^2)$
(5) $T_5(n) \in O(n^2)$

Exercise 2 – Solution

(1) $T_1(n) = 3n^3 + 2n^2 + 5$
(2) $T_2(n) = 1000n \cdot \log_2 n + 50n$
(3) $T_3(n) = n^2 + n^3 + n$
(4) $T_4(n) = n \cdot \log_2 n + 1000n^2$
(5) $T_5(n) = 10n^2 + 100n \cdot \log_2 n + 1000$

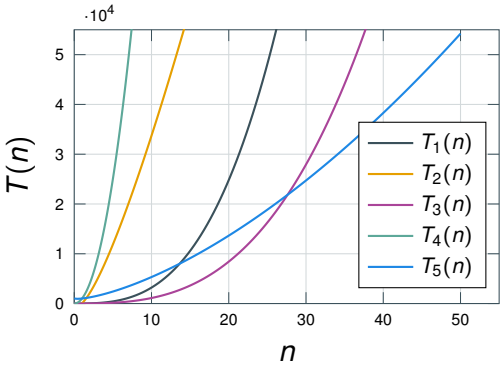
O-notation classification:

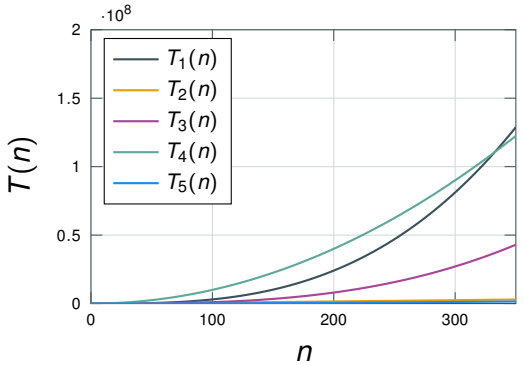
(1) $T_1(n) \in O(n^3)$
(2) $T_2(n) \in O(n \cdot \log_2 n)$
(3) $T_3(n) \in O(n^3)$
(4) $T_4(n) \in O(n^2)$
(5) $T_5(n) \in O(n^2)$



Exercise 2 – Solution

Visual comparison of the functions:





Exercise 3 – Complexity of Functions

Give the complexity in *O*-notation for

(1) lines 4-6,
(2) lines 3-7,
(3) lines 2-8.

```
1 function adder(A)
2   for i = length(A):-1:1
3     for j = 1:2:length(A)
4       if i != j
5         A[i] = A[i] + A[j]
6       end
7     end
8   end
9 end
```

Exercise 3 – Solution

(1) Lines 4-6: A comparison, an addition, and an assignment: all operations do not depend on the input size n. $\Rightarrow O(1)$

```
1 function adder(A)
2   for i = length(A):-1:1
3     for j = 1:2:length(A)
4       if i != j
5         A[i] = A[i] + A[j]
6       end
7     end
8   end
9 end
```

Exercise 3 – Solution

(1) Lines 4-6: A comparison, an addition, and an assignment: all operations do not depend on the input size n. $\Rightarrow O(1)$

(2) Lines 3-7: The inner loop runs $n / 2$ times, and each iteration performs $O(1)$ work. $\Rightarrow O(n)$

```
1 function adder(A)
2   for i = length(A):-1:1
3     for j = 1:2:length(A)
4       if i != j
5         A[i] = A[i] + A[j]
6       end
7     end
8   end
9 end
```

Exercise 3 – Solution

(1) Lines 4-6: A comparison, an addition, and an assignment: all operations do not depend on the input size n. $\Rightarrow O(1)$
(2) Lines 3-7: The inner loop runs $n / 2$ times, and each iteration performs $O(1)$ work. $\Rightarrow O(n)$
(3) Lines 2-8: The outer loop runs n times, and each iteration performs $O(n)$ work. $\Rightarrow O(n^2)$

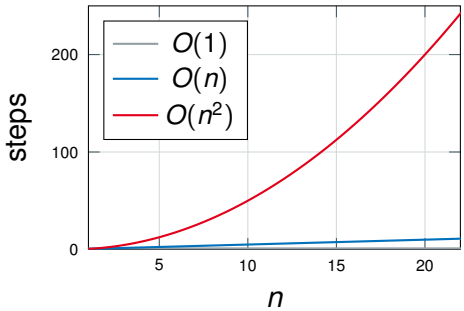
```
1 function adder(A)
2   for i = length(A):-1:1
3     for j = 1:2:length(A)
4       if i != j
5         A[i] = A[i] + A[j]
6       end
7     end
8   end
9 end
```

Exercise 3 – Solution

(1) Lines 4-6: A comparison, an addition, and an assignment: all operations do not depend on the input size n. $\Rightarrow O(1)$

(2) Lines 3-7: The inner loop runs $n / 2$ times, and each iteration performs $O(1)$ work. $\Rightarrow O(n)$

(3) Lines 2-8: The outer loop runs n times, and each iteration performs $O(n)$ work. $\Rightarrow O(n^2)$



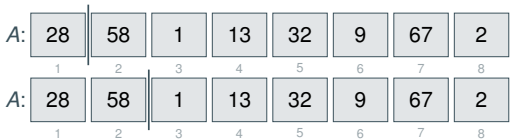
Exercise 4 – Insertion Sort

Sort the given sequence of numbers using *Insertion Sort* (ascending).
After each insertion, show the sorted and unsorted parts.

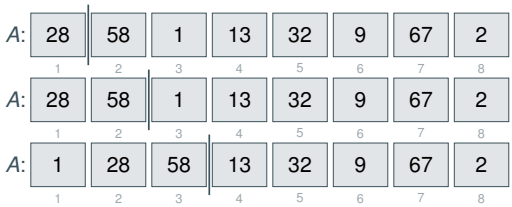
A:	28	58	1	13	32	9	67	2
	1	2	3	4	5	6	7	8

Exercise 4 – Solution

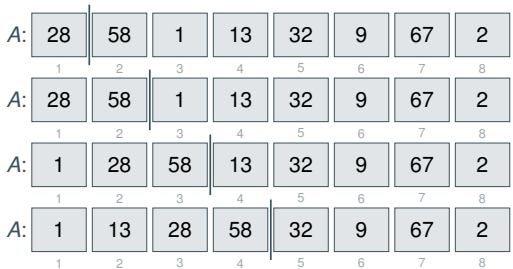
Exercise 4 – Solution



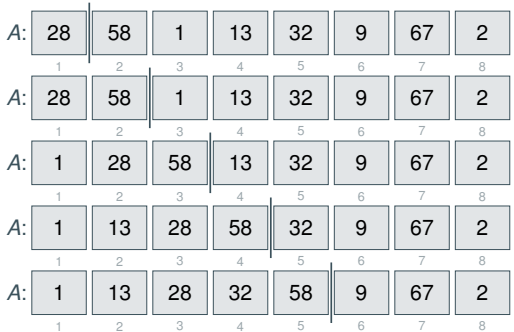
Exercise 4 – Solution



Exercise 4 – Solution

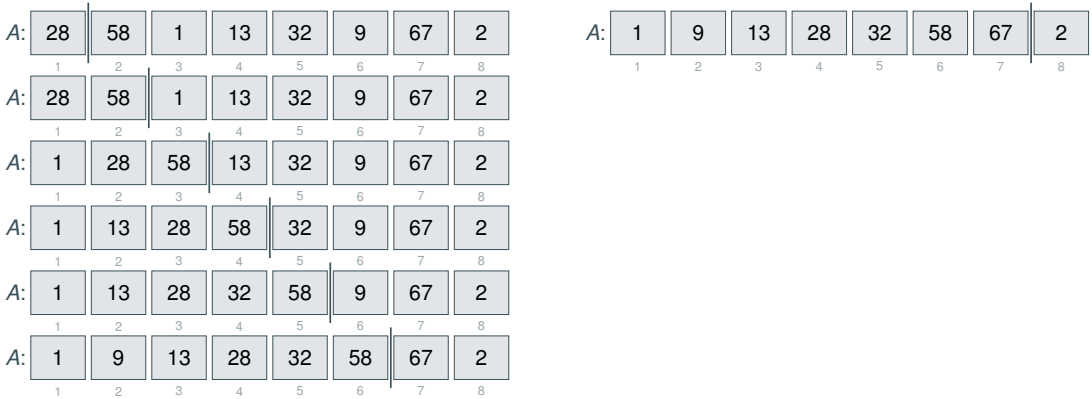


Exercise 4 – Solution



Exercise 4 – Solution

Exercise 4 – Solution



Exercise 4 – Solution

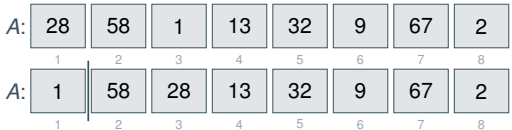
Exercise 5 – Selection Sort

Sort the given sequence of numbers using *Selection Sort* (ascending).
After each swap, show the sorted and unsorted parts.

A:	28	58	1	13	32	9	67	2
	1	2	3	4	5	6	7	8

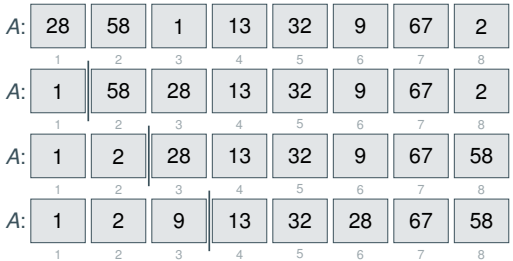
Exercise 5 – Solution

Exercise 5 – Solution

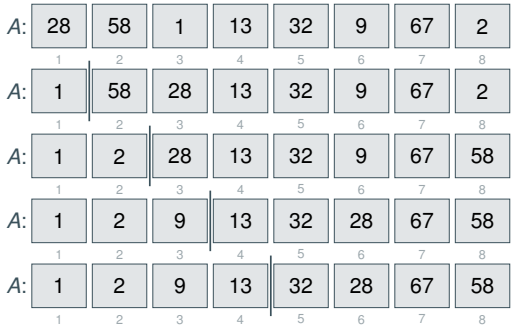


Exercise 5 – Solution

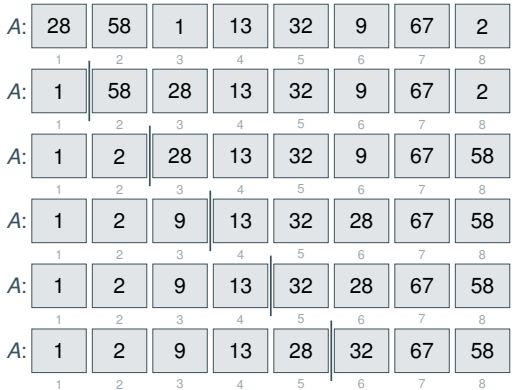
Exercise 5 – Solution



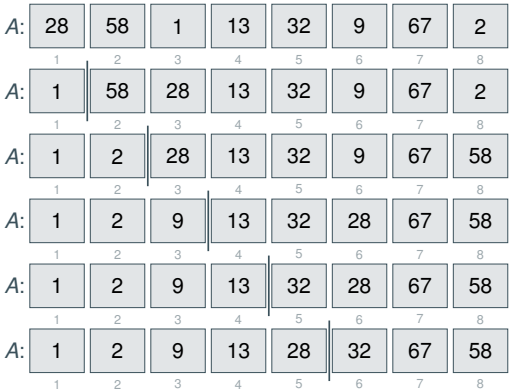
Exercise 5 – Solution



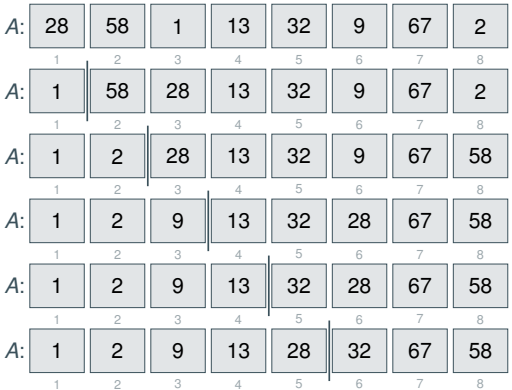
Exercise 5 – Solution

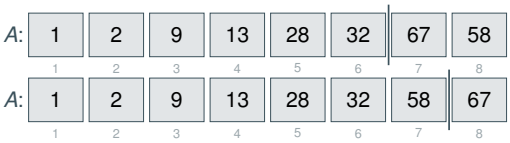


Exercise 5 – Solution



Exercise 5 – Solution





Exercise 6 – Comparison of Insertion Sort and Selection Sort

(1) What are the best and worst-case time complexities of Insertion Sort and Selection Sort?
(2) How many shifts does Insertion Sort perform in the worst case?
(3) How many swaps does Selection Sort perform in the worst case?
(4) Which sorting algorithm should be preferred?

Exercise 6 – Comparison of Insertion Sort and Selection Sort

Overview of Insertion Sort and Selection Sort:

```
function insertion_sort(A)
  for j = 2:length(A)
    key = A[j]
    i = j - 1
    while i > 0 && A[i] > key
      A[i + 1] = A[i]
      i -= 1
    end
    A[i + 1] = key
  end
end
```

```
function selection_sort(A)
  for i = 1:length(A)
    min = i
    for j = (i + 1):length(A)
      if A[j] < A[min]
        min = j
      end
    end
    x = A[i]
    A[i] = A[min]
    A[min] = x
  end
end
```

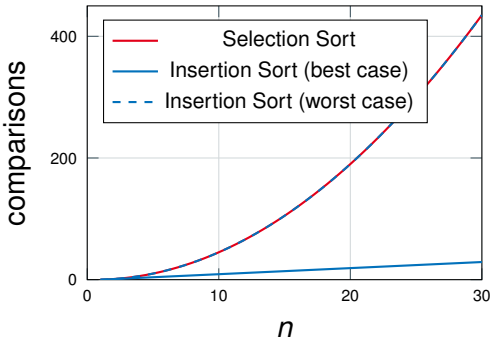
Exercise 6 – Solution

Insertion Sort

- Best case: $O(n)$
- Worst case: $O(n^2)$

Selection Sort

- Always: $O(n^2)$



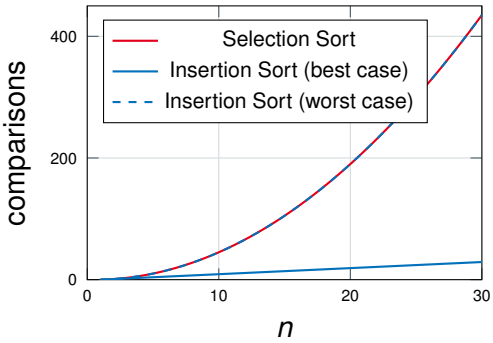
Exercise 6 – Solution

Insertion Sort

- Best case: $O(n)$
- Worst case: $O(n^2)$
- Up to $O(n^2)$ *shifts*

Selection Sort

- Always: $O(n^2)$
- At most $n - 1$ *swaps*



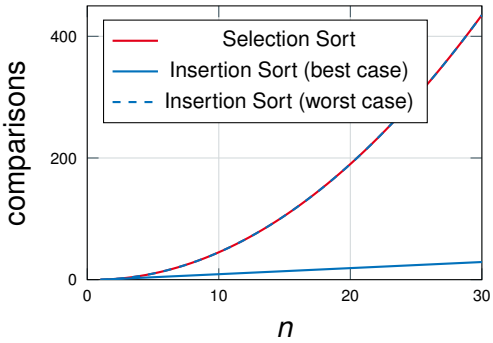
Exercise 6 – Solution

Insertion Sort

- Best case: $O(n)$
- Worst case: $O(n^2)$
- Up to $O(n^2)$ *shifts*

Selection Sort

- Always: $O(n^2)$
- At most $n - 1$ *swaps*



Selection Sort is preferred in scenarios where shifts are computationally expensive (e.g., due to large numbers)