# ALGORITHMS & DATA STRUCTURES

18th October

## TODAY’S PLAN

- Common mistakes
- Bonus Exercises
- Sorting
- Maximum Submatrix Sum in O(n^3)

## COMMON MISTAKES

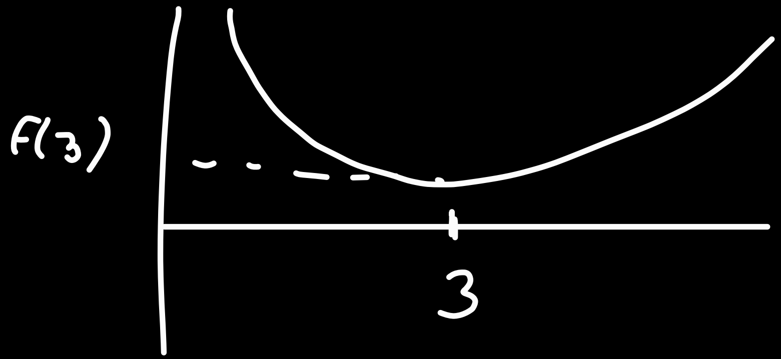
$$\min_{1 \le m \le n} f(m)$$

$$f(1) \quad f(2) \quad f(3) \quad \dots \quad f(m)$$

$$f(3) < f(i) \quad \forall i \neq 3$$

$$\Rightarrow \min_{1 \le m \le n} f(m) = f(3)$$

$$\min_{1 \le m \le n} f(m) = f(m)$$



$$\left[ \operatorname*{argmin}_{1 \le m \le n} f(m) \right]$$

## EXERCISE 3.3

**Algorithm 2**

```
for $j = 1, \ldots, n$ do
    for $k = 1, \ldots, n$ do
        for $m = 1, \ldots, n$ do
            $f()$
```

$$\sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{m=1}^{n} 1 = n^3$$

$$n^3 \leq c \cdot n^3$$

$$n^3 \geq c' \cdot n^3$$

$\Rightarrow$ # calls $\leq O(n^3)$

# calls $\geq \Omega(n^3)$

$\Longrightarrow \Theta(n^3)$ calls of $f()$

**Algorithm 3**

```
for $j = 1, \dots, n$ do
    $k \leftarrow \min(j, 100)$
    for $l = 1, \dots, k$ do
        $f()$
```

$$\sum_{j=1}^{n} \sum_{e=1}^{\min(j,100)} 1 = \sum_{j=1}^{n} \min(j, 100) \leq \sum_{j=1}^{n} 100 = 100n$$

$$\sum_{j=1}^{n} \sum_{e=1}^{\min(j,100)} 1 = \sum_{j=1}^{n} \min(j, 100) \geq \sum_{j=1}^{n} 1 = n$$

$\Rightarrow$ # calls $= \Theta(n)$

**Algorithm 4**

```
for $j = 1, \dots, n$ do
    if $j^2 \le n$ then
        for $k = j, \dots, n$ do
            $f()$
            $f()$
            $f()$
```

$j \le \lfloor \sqrt{n} \rfloor$

$n = 3$

$\sqrt{n} = 1.7\dots$

***Hint:*** *You may use the following fact without proof: For every $n \ge 2$, we have $\lfloor \sqrt{n} \rfloor \ge \frac{\sqrt{n}}{2}$ and $n - \lfloor \sqrt{n} \rfloor + 1 \ge \frac{n}{2}$, where $\lfloor x \rfloor$ is the largest integer satisfying $\lfloor x \rfloor \le x$.*

$$\sum_{j=1}^{\lfloor \sqrt{n} \rfloor} \sum_{k=j}^{n} 3 = \sum_{j=1}^{\lfloor \sqrt{n} \rfloor} 3(n-j+1) \le \sum_{j=1}^{\lfloor \sqrt{n} \rfloor} 3n = 3n \lfloor \sqrt{n} \rfloor \le 3n\sqrt{n}$$

$$\sum_{j=1}^{\lfloor \sqrt{n} \rfloor} \sum_{k=j}^{n} 3 = \sum_{j=1}^{\lfloor \sqrt{n} \rfloor} 3(n-j+1) \ge \sum_{j=1}^{\lfloor \sqrt{n} \rfloor} 3(n - \lfloor \sqrt{n} \rfloor + 1)$$

$$\ge \sum_{j=1}^{\lfloor \sqrt{n} \rfloor} 3 \cdot \frac{n}{2} = \frac{3}{2} n \lfloor \sqrt{n} \rfloor$$

$$\ge \frac{3}{4} n \sqrt{n}$$

$\Rightarrow$ # calls $= \Theta(n\sqrt{n})$

## EXERCISE 3.4

| $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $A_i$ | 150 | 150 | 200 | -100 |
| $I_i$ | 150 | 350 | 500 | 400 |

a) Let $1 \leq i \leq n-1$. Suppose that you had invested 1 CHF on the $i$-th day and sold the entire investment on the $(i+1)$-th day. Show that you would have got $\dfrac{I_{i+1} - A_{i+1}}{I_i}$ CHF in return.

$I_i$ : value of portfolio on day $i$

$I_{i+1} - A_{i+1}$ : new value on day $i+1$ of what we had on day on day $i$

$$\frac{I_{i+1} - A_i}{I_i}$$

b) Let $1 \leq i \leq j \leq n$. Suppose that you had invested 100 CHF on the $i$-th day and sold the entire investment on the $j$-th day. Show that your profit is equal to

$$100 \cdot \prod_{k=i}^{j-1} \frac{I_{k+1} - A_{k+1}}{I_k} - 100.$$

Note that in the above equation, we adopt the convention that if $i = j$, then $\prod_{k=i}^{j-1} \frac{I_{k+1} - A_{k+1}}{I_k} = 1$.

You are interested in finding the maximum profit that you could have made in a single buy-sell operation by investing 100 CHF. Here, you would buy 100 CHF worth of the stock on some day $i$ where $1 \leq i \leq n$ and then sell the entire investment another day $j$ where $i \leq j \leq n$.

We first assume that all $A_1, \ldots, A_n$ and $I_1, \ldots, I_n$ are positive, and that $I_k > A_k$ for every $k$.

c) Describe how you can use the maximum subarray-sum algorithm that you learned in class in order to devise an algorithm that computes the maximum profit in $O(n)$ time. You can assume that arithmetic operations (such as addition, subtraction, multiplication and division) as well as logarithms and exponentials are elementary. This means that the computation of log and exp take one unit of time each.

***Hint:*** *You can use the fact that the logarithm is a strictly increasing function that turns products into sums.*

$$\max_{1 \leq i \leq j \leq n} 100 \prod_{k=i}^{j-1} \frac{I_{k+1} - A_{k+1}}{I_k} - 100 \quad (1)$$

Wir machen zuerst $\max\limits_{1 \leq i \leq j \leq n} \prod\limits_{k=i}^{j-1} \frac{I_{k+1} - A_{k+1}}{I_k}$ und dann setzen wir das Ergebnis in (1).

Wir maximisieren $\max\limits_{1 \leq i \leq j \leq n} \log\left[ \prod\limits_{k=i}^{j-1} \frac{I_{k+1} - A_{k+1}}{I_k} \right] = \max\limits_{1 \leq i \leq j \leq n} \sum\limits_{k=i}^{j-1} \underbrace{\log\left(\frac{I_{k+1} - A_{k+1}}{I_k}\right)}_{G_k}$

Wenn wir $\max_{1 \leq i \leq j \leq n} \sum_{k=i}^{j-1} G_k =: r$, dann ist

$$\max_{1 \leq i \leq j \leq n} \log\left( \prod_{k=i}^{j-1} \frac{I_{k+1} - A_{k+1}}{I_k} \right) = r$$

$$\max_{1 \leq i \leq j \leq n} \prod_{k=i}^{j-1} \frac{I_{k+1} - A_{k+1}}{I_k} = e^r := r'$$

$$\max_{1 \leq i \leq j \leq n} 100 \prod_{k=i}^{j-1} \frac{I_{k+1} - A_{k+1}}{I_k} - 100 = 100 r' - 100$$

---

Berechne $G_1 \dots G_n$ $\Theta(n)$

Max. subarray sum auf $G_1 \dots G_n$ $\Theta(n)$

$100 e^r - 100$ $\Theta(1)$

$\Rightarrow$ Alles geht in $\Theta(n)$

d) Now assume that the logarithm and exponential operations are expensive so that we would like to avoid using them. Explain how you can modify the maximum subarray sum algorithm in order to solve the problem using only elementary arithmetic operations such as addition, subtraction, multiplication and division. The running time of the algorithm should remain in $O(n)$.

$$\max_{1 \leq i \leq j \leq n} \prod_{k=i}^{j-1} \underbrace{\frac{I_{k+1} - A_{k+1}}{I_k}}_{a_k}$$

Und wir setzen res

in (1)

```
res ← 0
rand_max ← 0

for i = 1 ... n
    rand_max += a_i
    res ← max(
                res,
                rand_max)
    if rand_max < 0
        randmax ← 0
```

```
res ← 1
rand_max ← 1

for i = 1 ... n
    rand_max *= a_i
    res ← max( res,
               rand_max)
    if rand_max < 1
        rand_max ← 1
```

## SORTING AND SEARCHING

## SEARCHING: TWO ALGORITHMS

**LINEAR SEARCH**

- Check all elements
- Array does not need to be sorted
- O(n)

**BINARY SEARCH**

- “Dictionary” idea
- Array needs to be sorted
- O(log n)

## SEARCHING: WHEN USING WHICH?

**LINEAR SEARCH O(n)**

- If the array is not sorted and you need to search just  “a few” elements

**BINARY SEARCH O(log n)**

- If the array is already sorted
- If you have to search “many” elements (in depth analysis next week)

## SORTING: THREE BASIC ALGORITHMS IN O(n^2)

- https://www.youtube.com/watch?v=xli_FI7CuzA
- https://www.youtube.com/watch?v=g-PGLbMth_g
- https://www.youtube.com/watch?v=JU767SDMDvA

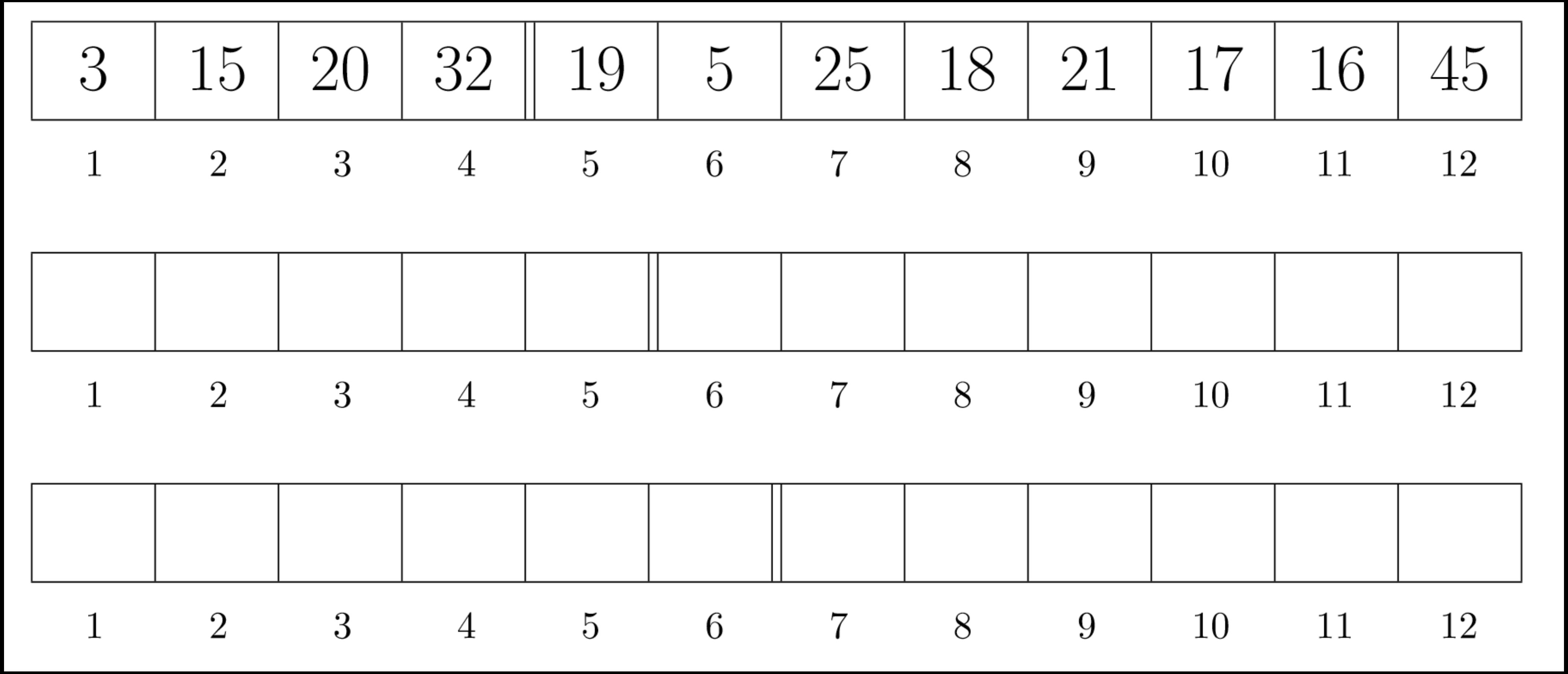
Führen Sie auf dem folgenden Array zwei Iterationen des Sortieralgorithmus *Sortieren durc* *Einfügen* aus. Das zu sortierende Array ist durch vorherige Iterationen dieses Algorithmu bereits bis zum Doppelstrich sortiert worden.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 15 | 20 | 32 | 19 | 5 | 25 | 18 | 21 | 17 | 16 | 45 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

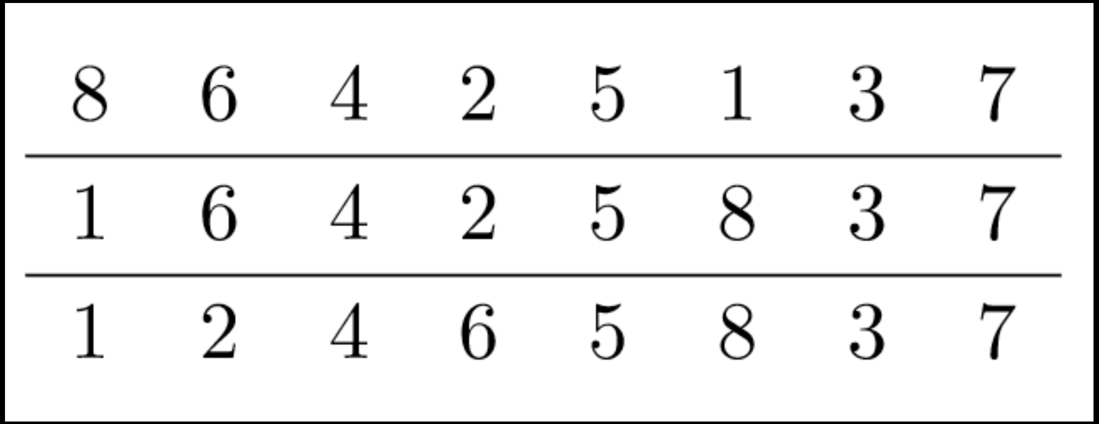
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 15 | 19 | 20 | 32 | 5 | 25 | 18 | 21 | 17 | 16 | 45 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 5 | 15 | 19 | 2 | 32 | 25 | 18 | 21 | 17 | 16 | 45 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

# TWO STEPS OF SELECTION SORT



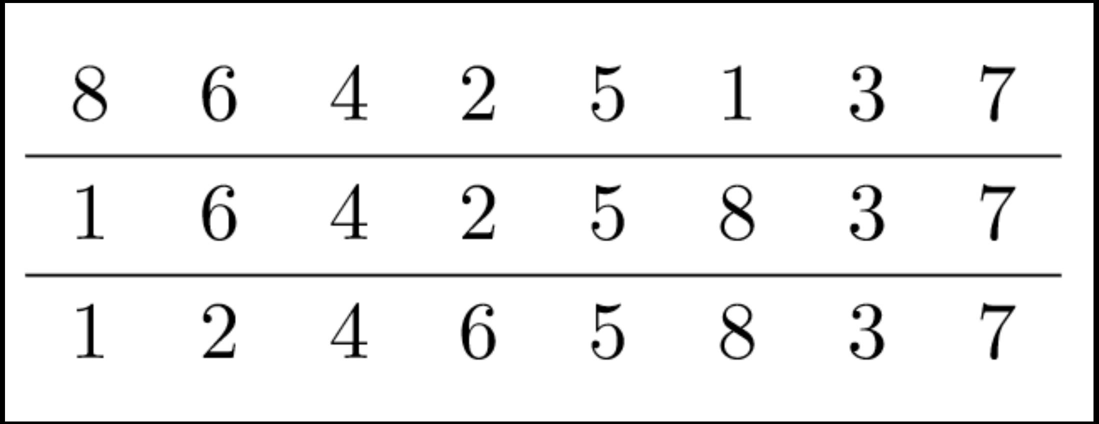
## WHAT ALGORITHM IS THIS?



|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 8 | 6 | 4 | 2 | 5 | 1 | 3 | 7 |
| 1 | 6 | 4 | 2 | 5 | 8 | 3 | 7 |
| 1 | 2 | 4 | 6 | 5 | 8 | 3 | 7 |

A. BUBBLE SORT
B. SELECTION SORT
C. INSERTION SORT

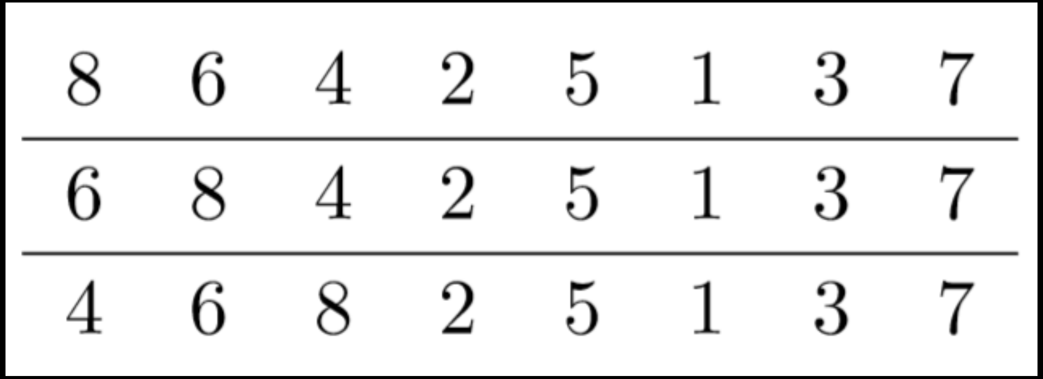
# WHAT ALGORITHM IS THIS?



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | 6 | 4 | 2 | 5 | 1 | 3 | 7 |
| 1 | 6 | 4 | 2 | 5 | 8 | 3 | 7 |
| 1 | 2 | 4 | 6 | 5 | 8 | 3 | 7 |

A. BUBBLE SORT
[✓ B. SELECTION SORT]
C. INSERTION SORT

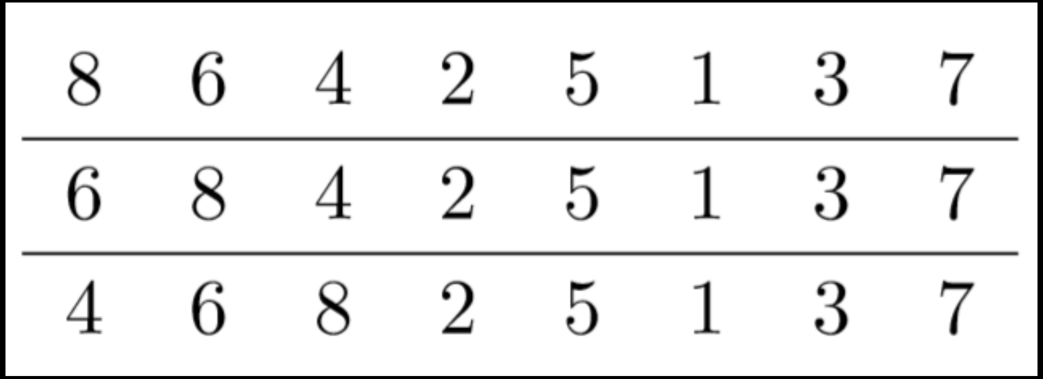
# WHAT ALGORITHM IS THIS?



|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 8 | 6 | 4 | 2 | 5 | 1 | 3 | 7 |
| 6 | 8 | 4 | 2 | 5 | 1 | 3 | 7 |
| 4 | 6 | 8 | 2 | 5 | 1 | 3 | 7 |

A. BUBBLE SORT
B. SELECTION SORT
C. INSERTION SORT

# WHAT ALGORITHM IS THIS?



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | 6 | 4 | 2 | 5 | 1 | 3 | 7 |
| 6 | 8 | 4 | 2 | 5 | 1 | 3 | 7 |
| 4 | 6 | 8 | 2 | 5 | 1 | 3 | 7 |

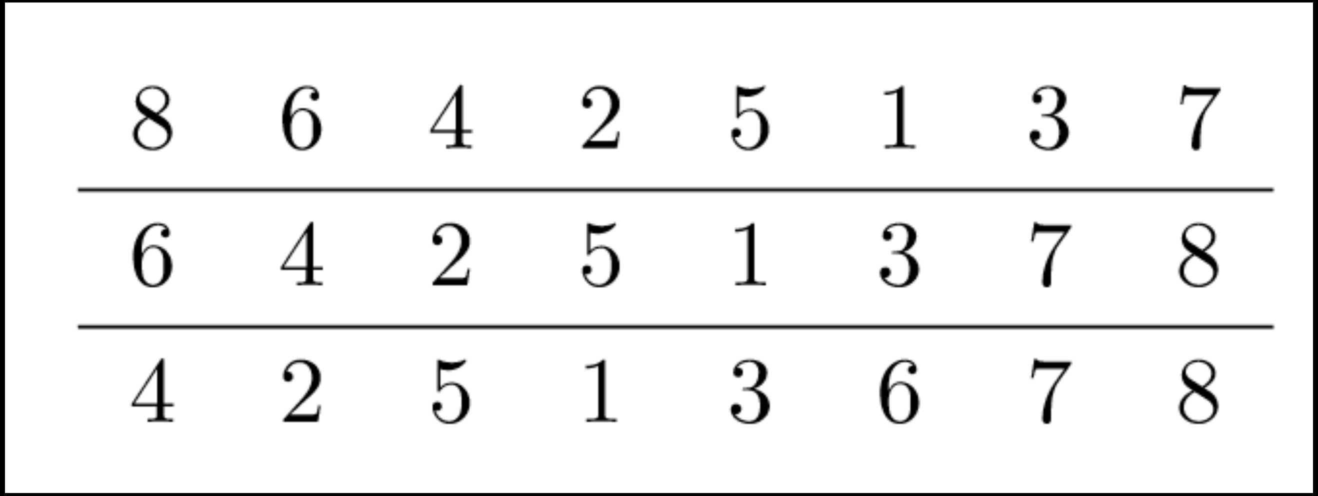
A. BUBBLE SORT
B. SELECTION SORT
C. [✓ INSERTION SORT]

## PERFORM TWO BUBBLE SORT STEPS

Input: 8, 6, 4, 2, 5, 1, 3, 7

# PERFORM TWO BUBBLE SORT STEPS

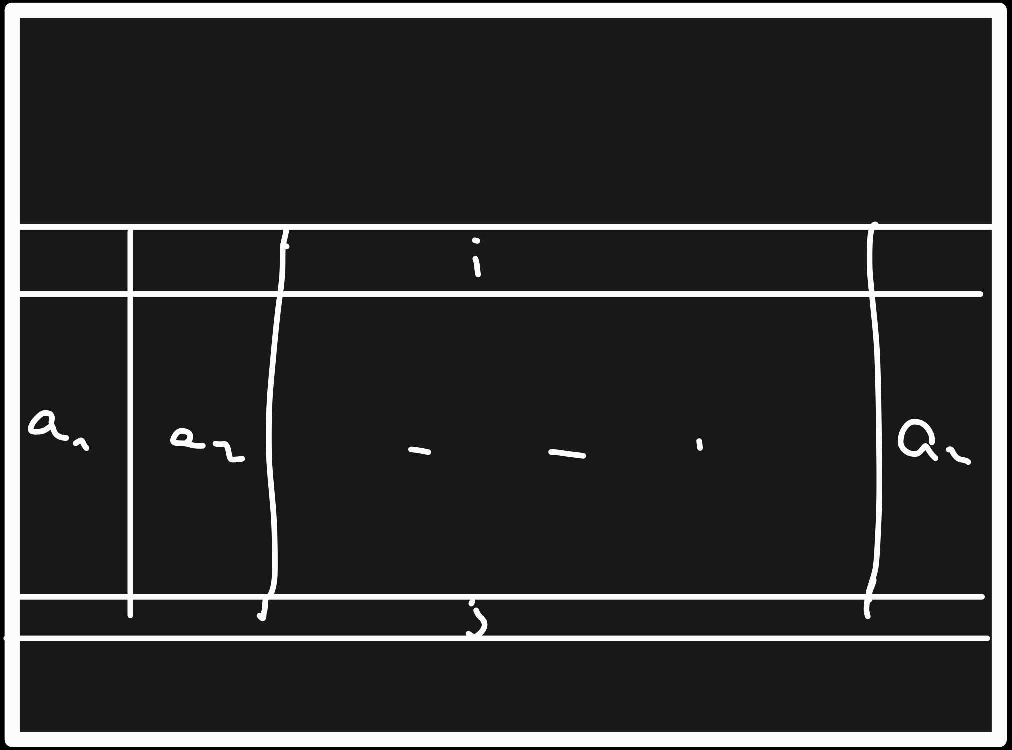
Input: 8, 6, 4, 2, 5, 1, 3, 7



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | 6 | 4 | 2 | 5 | 1 | 3 | 7 |
| 6 | 4 | 2 | 5 | 1 | 3 | 7 | 8 |
| 4 | 2 | 5 | 1 | 3 | 6 | 7 | 8 |

MATRIX

## MAXIMUM SUBARRAY SUM IN O(n^3)



for all i, j

Ich berechne $a_1 \ldots a_n$

MAXIMUM SUBARRAY-sum