Herzlich Willkommen bei Inf-Einf-B Block 3.

Heute geht es um **Algorithmen (Suchen und Sortieren) und Laufzeitkomplexität. Außerdem schauen wir uns C structs an.**

Diese Woche gibt es auch eine **Check-Aufgabe** für die eigenständige Lernziel-kontrolle (anders als bei Übungen veröffentlichen wir dazu keine Lösungen).

Wenn es zu schnell geht:
Section Video / Notes anschauen.

Kurs-Website mit Wochenplan, Videos, Notizen, Aufgaben, FAQ, uvm.: **inf.zone**

Die Vorlesungen werden aufgezeichnet und im Internet veröffentlicht. Die Videos werden allerdings meist erst 3 Wochen später online sein; Folien und Notizen am Tag nach der Vorlesung. Machen Sie sich Notizen!

Ihre Fragen sind in der Aufzeichnung in der Regel nicht zu hören (ich wiederhole sie).

Wir machen heute eine Pause.

Dies ist Inf-Einf-B.

lecture-1 (Woche 1) · 103 Teilnehmende

Du: **2/3** (67%) Ganzer Kurs: **1.9/3** (65%) +2 PP

Verteilung im Kurs:

0% ≤1/3 ≤2/3 <90% ≥90%

lecture-2 (Woche 2) · 104 Teilnehmende

Du: **2/3** (67%) Ganzer Kurs: **1.5/3** (51%) +16 PP

Verteilung im Kurs:

0% ≤1/3 ≤2/3 <90% ≥90%

block25 (Woche 2.5) · 123 Teilnehmende

Du: **13/20** (65%) Ganzer Kurs: **9.9/20** (50%) +15 PP

Verteilung im Kurs:

0% ≤1/3 ≤2/3 <90% ≥90%

Gesamtstatistik

Deine Leistung	Ganzer Kurs	Differenz
17/26	13.4/26	+14 Prozentpunkte
65%	51%	

Aktuelle Einschätzung

Einschätzung deiner Situation

Du liegst zwar über dem Kursdurchschnitt, aber die meisten Fragen sind grundlegend – du solltest eigentlich **mindestens 85%** schaffen. Schau dir die Aufgaben an, die du nicht lösen konntest (siehe unten), und arbeite gezielt diese Themen nach. Besuche **regelmäßig ein Tutorium**, um konkrete Lücken zu schließen, und löse die **Übungsaufgaben vollständig**, nicht nur teilweise. Die Klausur setzt solides Verständnis der Basics voraus – "über dem Durchschnitt" reicht nicht für eine gute Note!

Lernfortschritt für ausgewählte Kompetenzen

Diese Analyse zeigt deine Entwicklung bei grundlegenden Programmieraufgaben über mehrere Wochen hinweg.

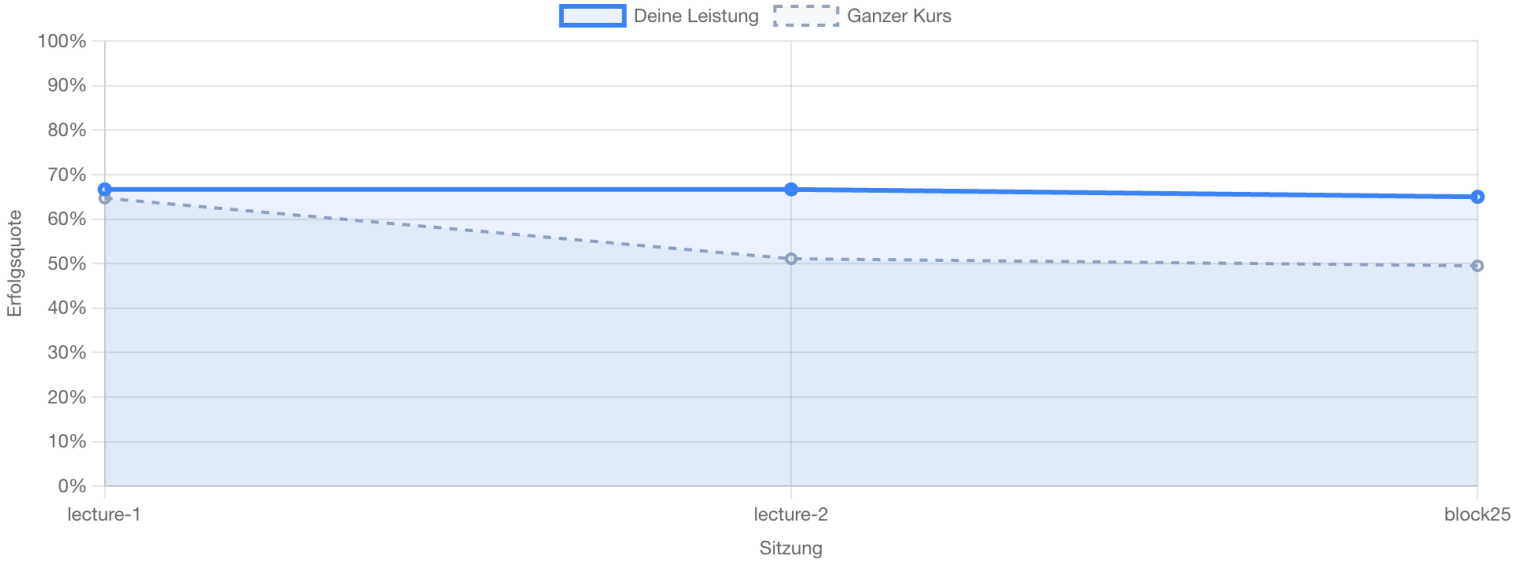
Lernfortschritt: Grundlegendes Schleifen-Tracing

lecture-1	lecture-2	block25
✓	✓	✓
Tracing-Problem 2	Tracing Problem	Hi
Ganzer Kurs: 56%	Ganzer Kurs: 77%	Ganzer Kurs: 79%

Exzellent! Du beherrschst grundlegendes Schleifen-Tracing durchgehend. Diese Kompetenz ist fundamental für alle weiteren Programmieraufgaben.

Dein Lernverlauf im Vergleich

Die Prozentwerte zeigen, welcher Anteil der Antwortenden angab, die Aufgabe lösen zu können.



Hochgeladene Übungsaufgaben

Woche 0 Scratch	Woche 1 C	Woche 2 Arrays
88	130 Endstand	112+

```c
#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    string word = "cabbage";
    int counts[26] = {0}; // alle auf 0 setzen

    for (int i = 0; i < strlen(word); i++)
    {
        int index = word[i] - 'a';
        counts[index]++;
    }

    for (int i = 0; i < 25; i++)
    {
        if ( ??? )
        {
            printf("Match: %c and %c appear %d times\n", ?, ?, ?);
            return 0;
        }
    }

    printf("No match\n");
}
```

PROBLEM 1

Ergänze die Lücken so, dass das Programm zwei benachbarte Buchstaben im Alphabet findet, die im Wort gleich oft vorkommen.

Im Beispiel soll ausgegeben werden:
Match: a and b appear 2 times.

if-Bedingung:
_____

printf Argumente: _____, _____, _____

```c
#include <cs50.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    string word = "cabbage";
    int counts[26] = {0}; // alle auf 0 setzen

    for (int i = 0; i < strlen(word); i++)
    {
        int index = word[i] - 'a';
        counts[index]++;
    }

    for (int i = 0; i < 25; i++)
    {
        if (counts[i] == counts[i + 1] && counts[i] > 0)
        {
            printf("Match: %c and %c appear %d times\n", 'a' + i, 'a' + i + 1, counts[i]);
            return 0;
        }
    }

    printf("No match\n");
}
```

PROBLEM 1

Ergänze die Lücken so, dass das Programm zwei benachbarte Buchstaben im Alphabet findet, die im Wort gleich oft vorkommen.

Im Beispiel soll ausgegeben werden:
Match: a and b appear 2 times.

```
A  #include <stdio.h>
B  arr[i + 1] = arr[i];
C  #include <cs50.h>
D  arr[i] = arr[i + 1];
E  for (int i = 0; i < 5; i++) {
F  printf("%d ", arr[i]);
G  printf("\n");
H  }
I  }
J  arr[4] = first;
K  }
L  int arr[] = {10, 20, 30, 40, 50};
M  int first = arr[0];
N  for (int i = 0; i < 6; i++) {
O  int main(void) {
P  for (int i = 0; i < 4; i++) {
```

PROBLEM 2

Ordne die Zeilen so, dass das Programm das erste Element ans Ende verschiebt und alle anderen nach links.

Beispiel

[10, 20, 30, 40, 50]

→

[20, 30, 40, 50, 10

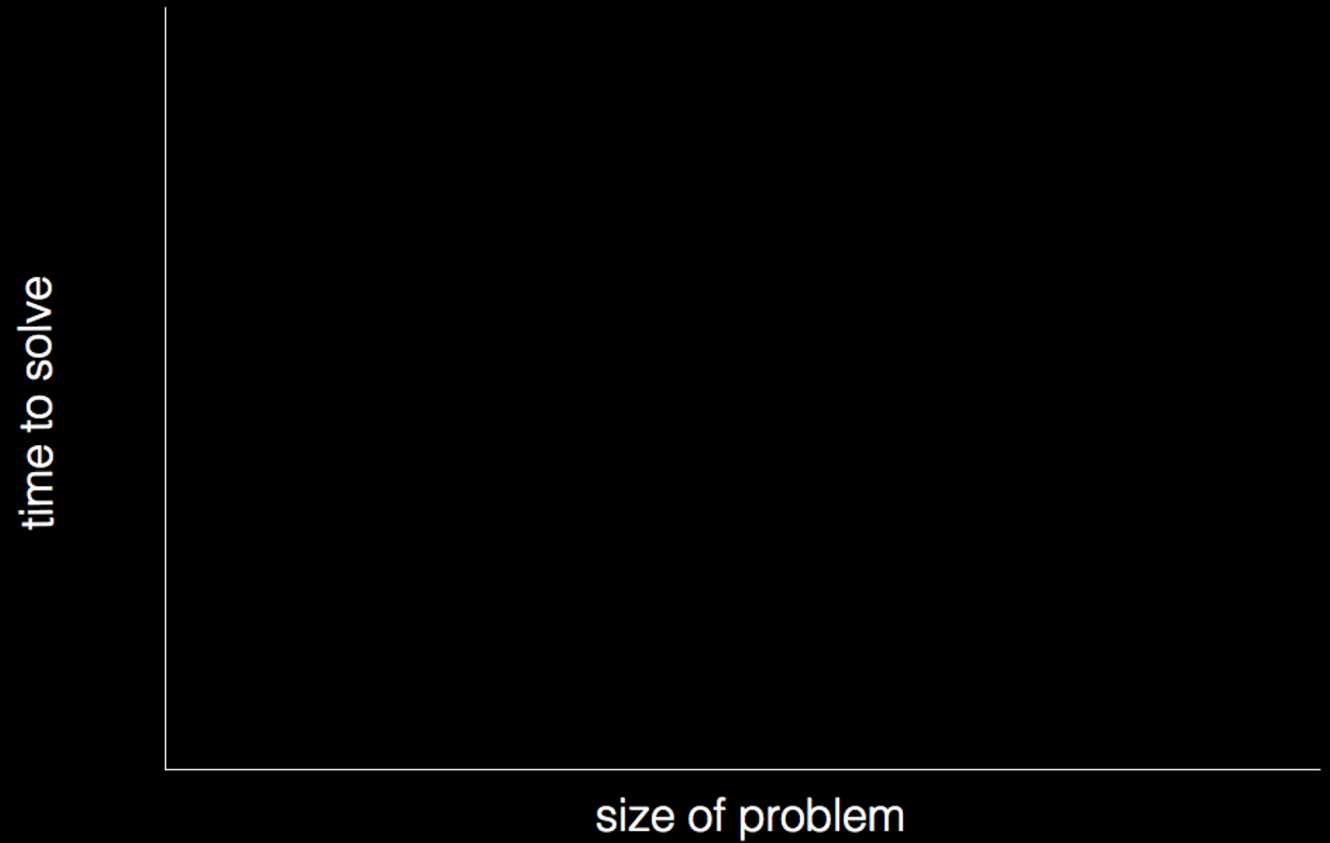
```c
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    int arr[] = {10, 20, 30, 40, 50};
    int first = arr[0];
    for (int i = 0; i < 4; i++) {
        arr[i] = arr[i + 1];
    }
    arr[4] = first;
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

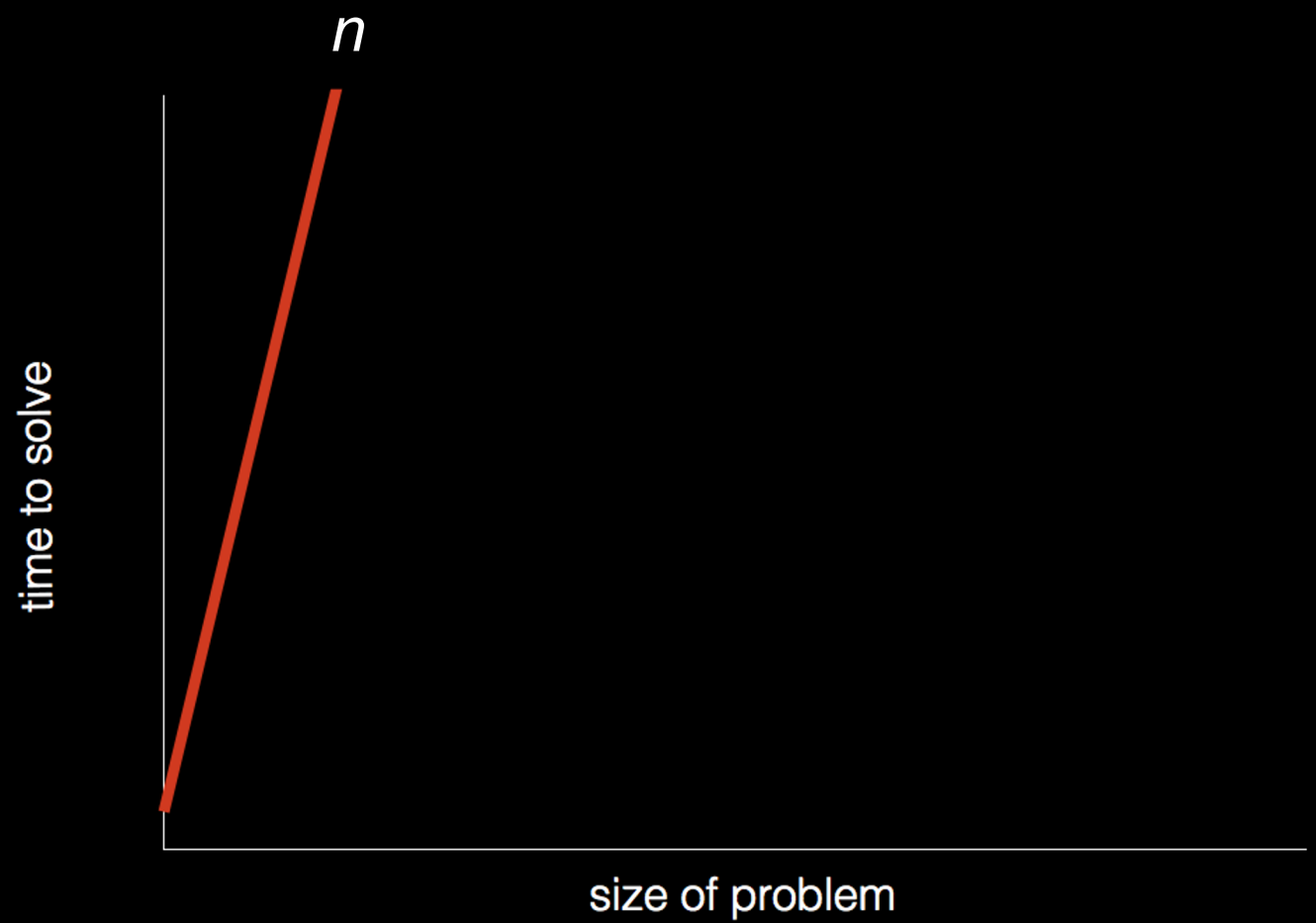
PROBLEM 2

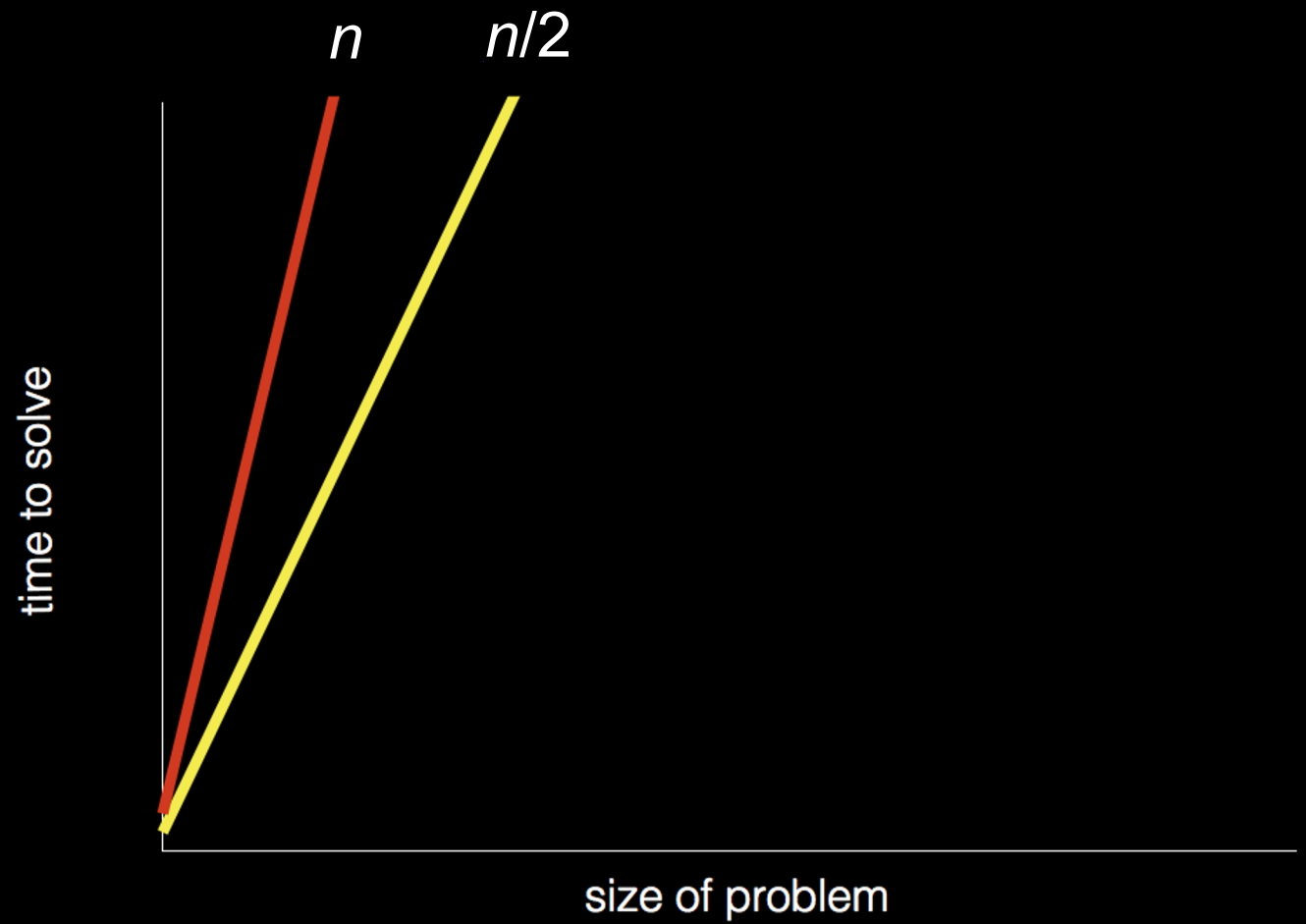
Ordne die Zeilen so, dass das Programm das erste Element ans Ende verschiebt und alle anderen nach links.

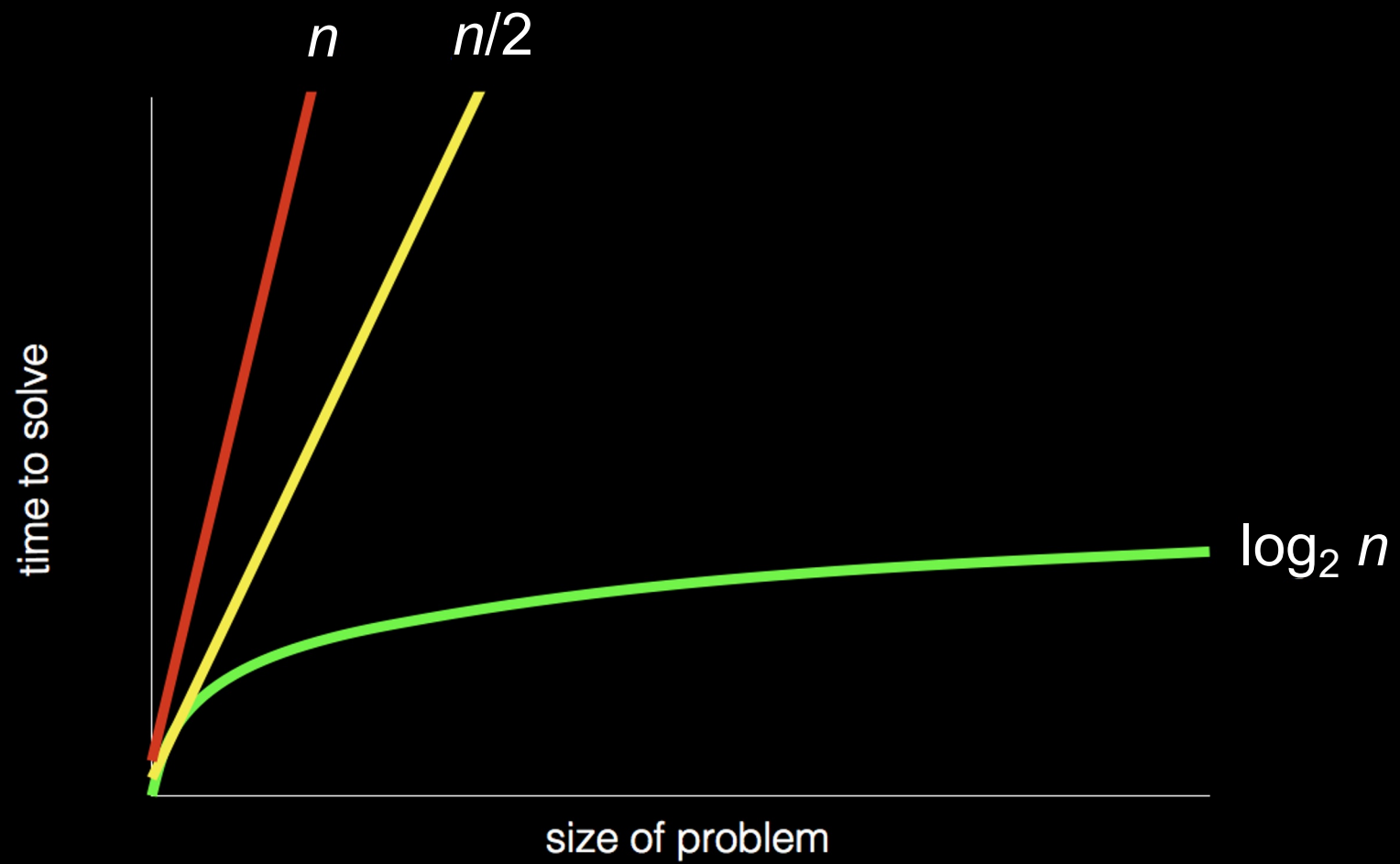
Beispiel
[10, 20, 30, 40, 50]
→
[20, 30, 40, 50, 10

lernen wie man algorithmisch denkt

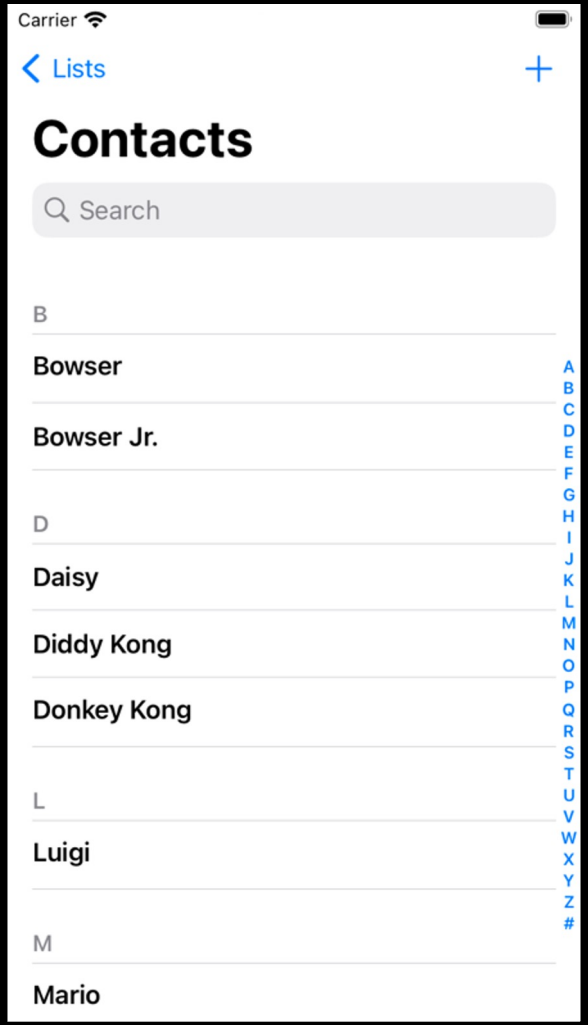

time to solve
size of problem

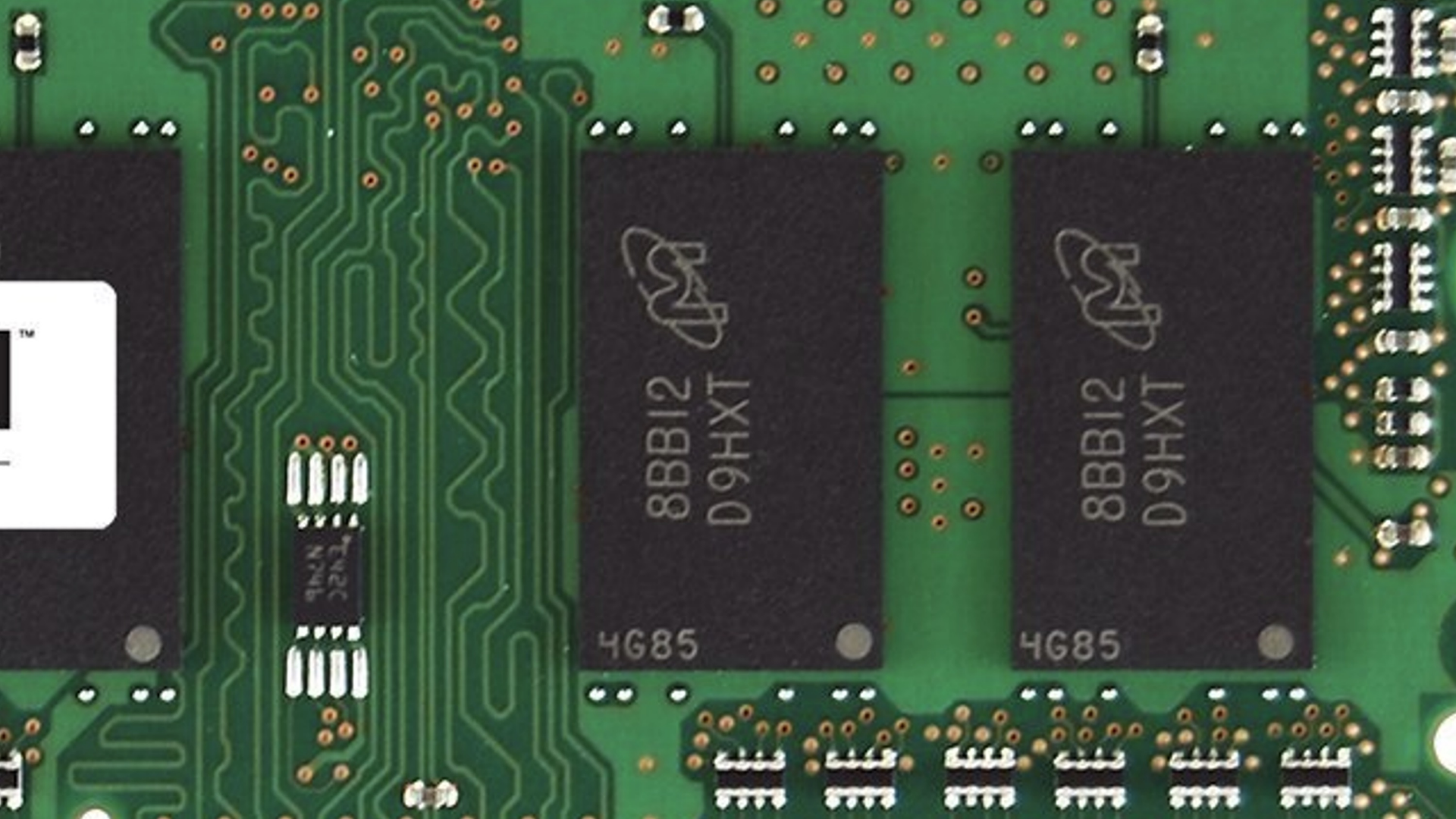

n
time to solve
size of problem

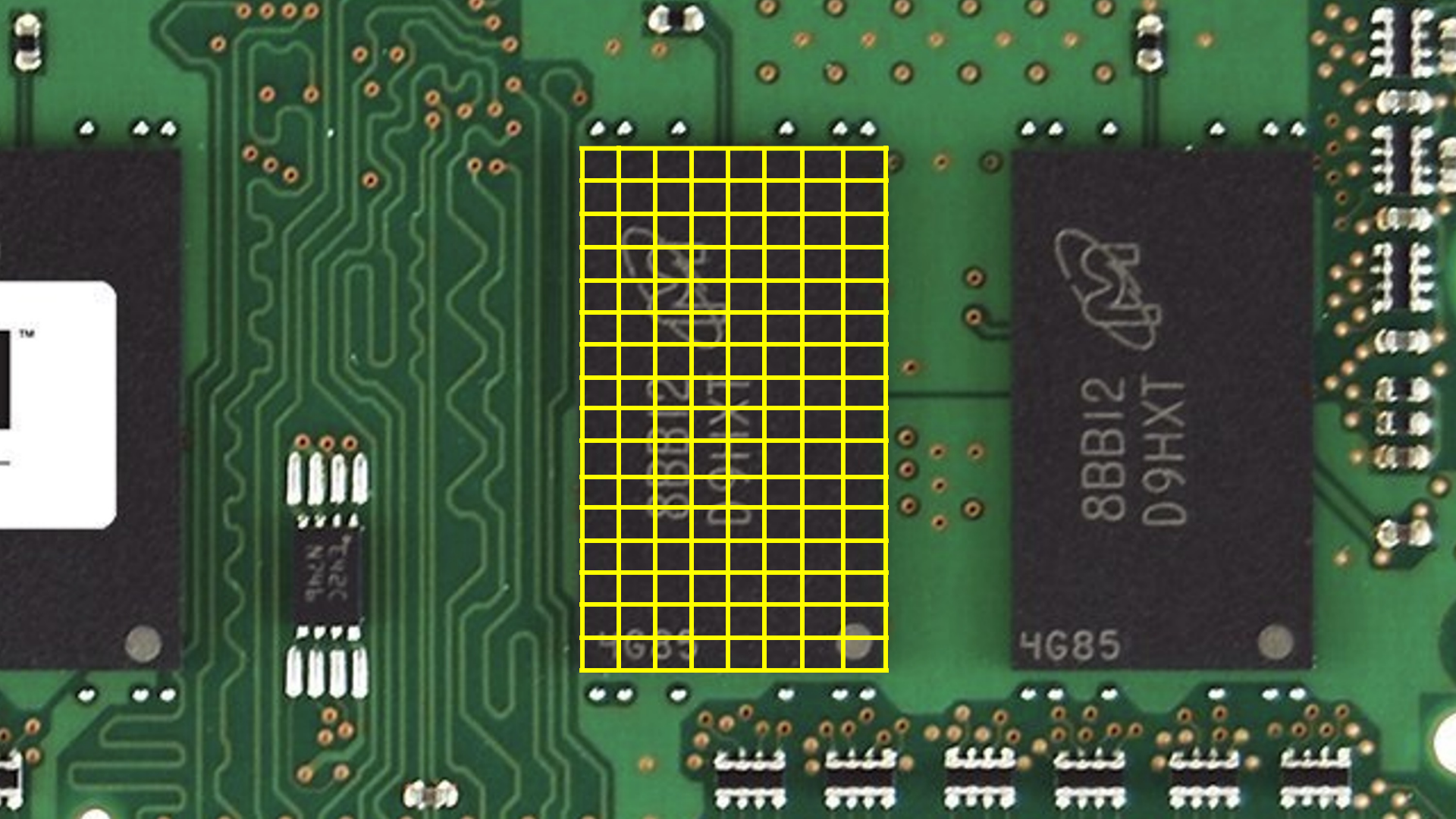

n
n/2
time to solve
size of problem

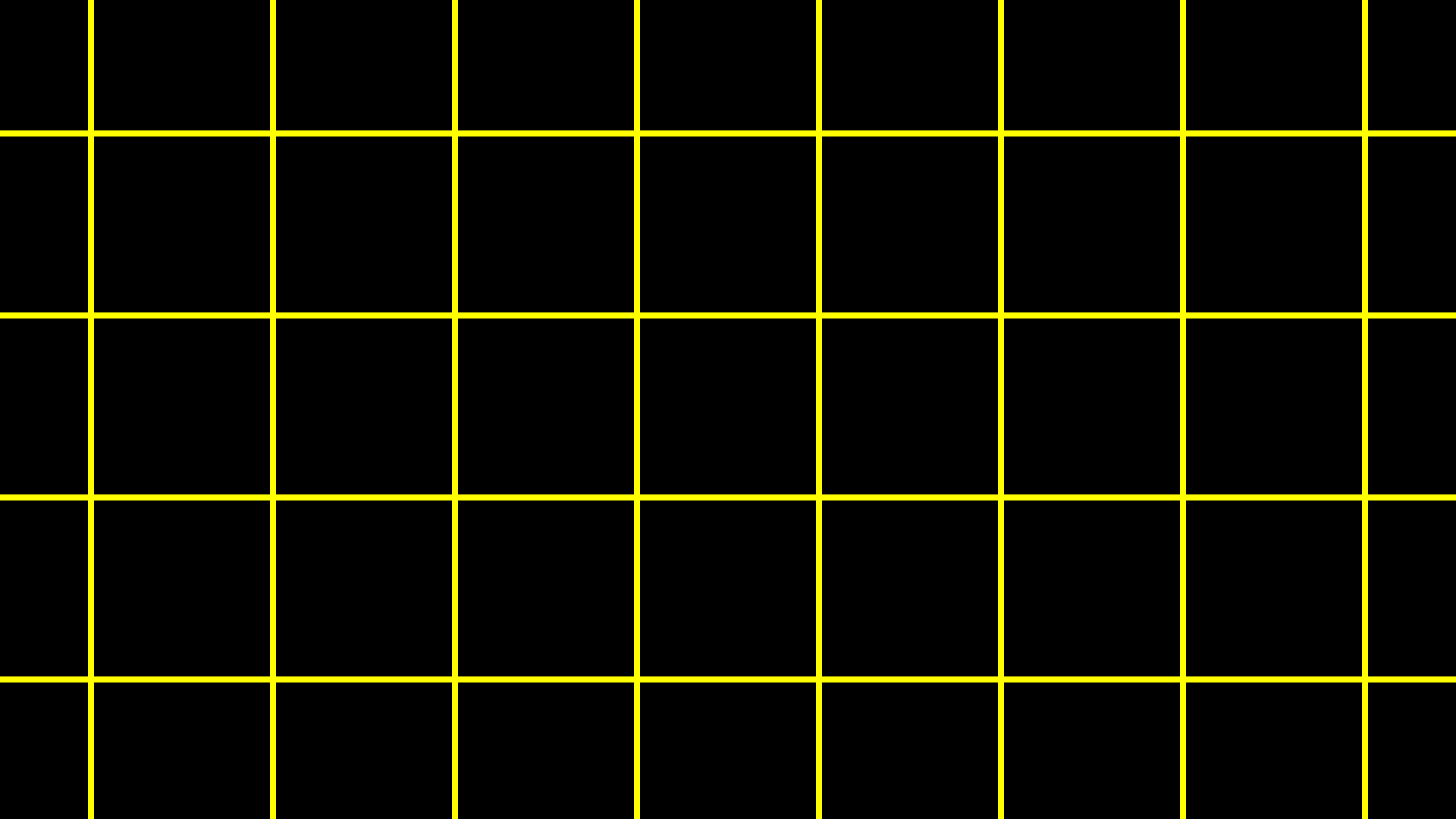

n
n/2
log2 n
time to solve
size of problem

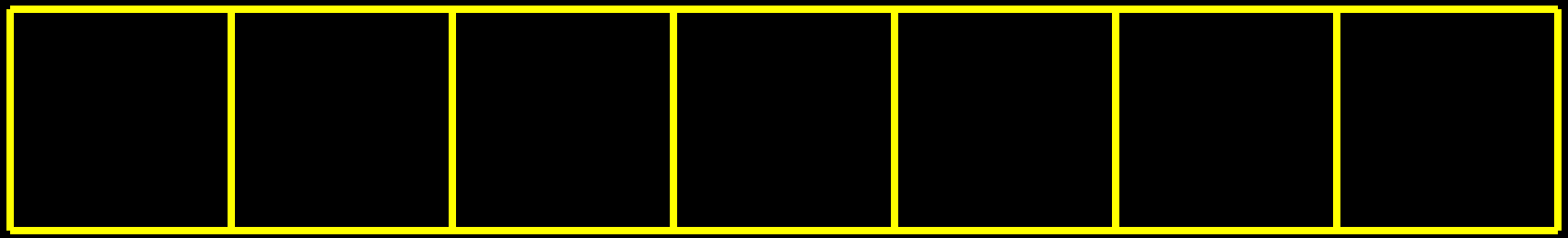
1. Stehen Sie auf und denken Sie an die Zahl 1.
2. Bilden Sie ein Paar mit jemandem, der steht, addieren Sie deren bzw. dessen Zahl zu Ihrer und merken Sie sich die Summe.
3. Eine bzw. einer von Ihnen setzt sich hin.
4. Wenn Sie noch stehen, gehen Sie zurück zu Schritt 2.

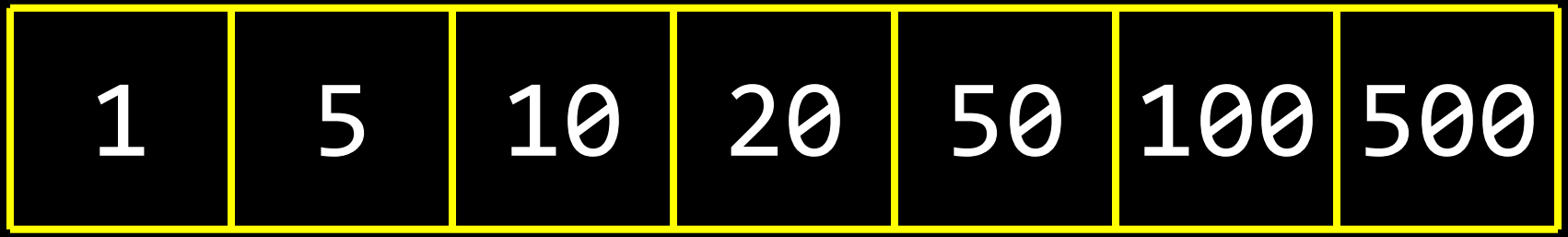

Carrier
Lists
Contacts
Search
B
Bowser
Bowser Jr.
D
Daisy
Diddy Kong
Donkey Kong
L
Luigi
M
Mario
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
#

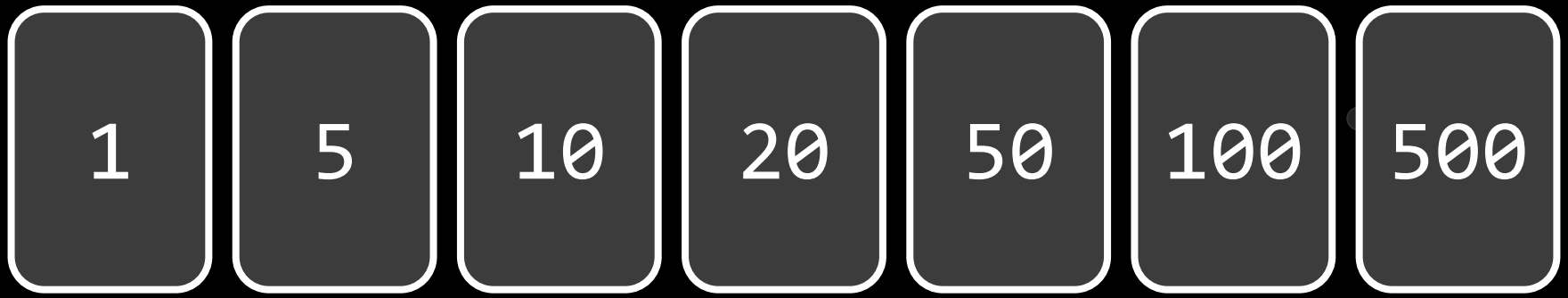

8BBI2
D9HXT
4G85
8BBI2
D9HXT
4G85

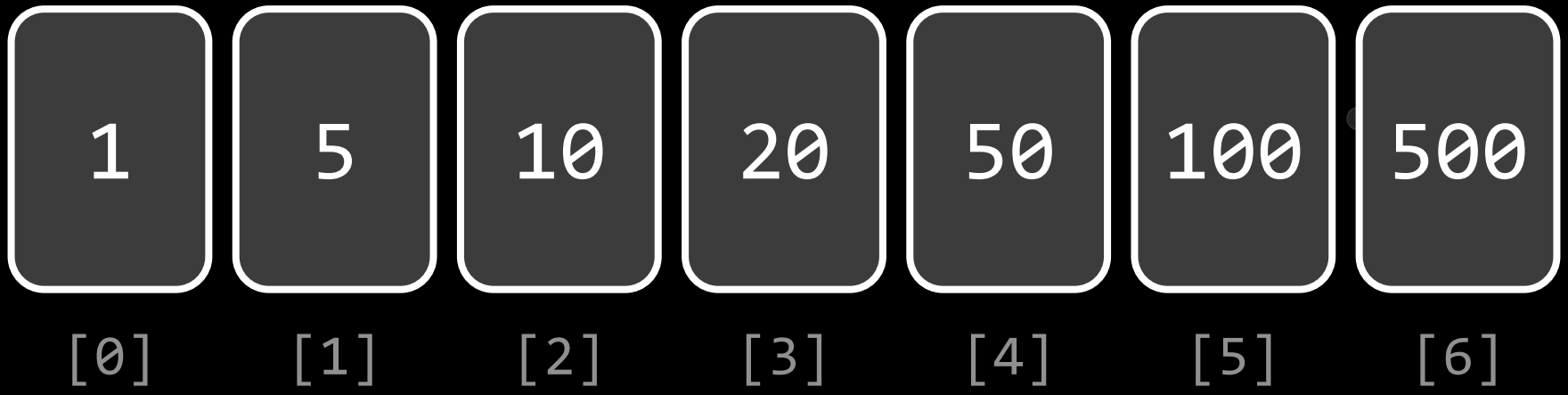

8BBI2
D9HXT
4G85



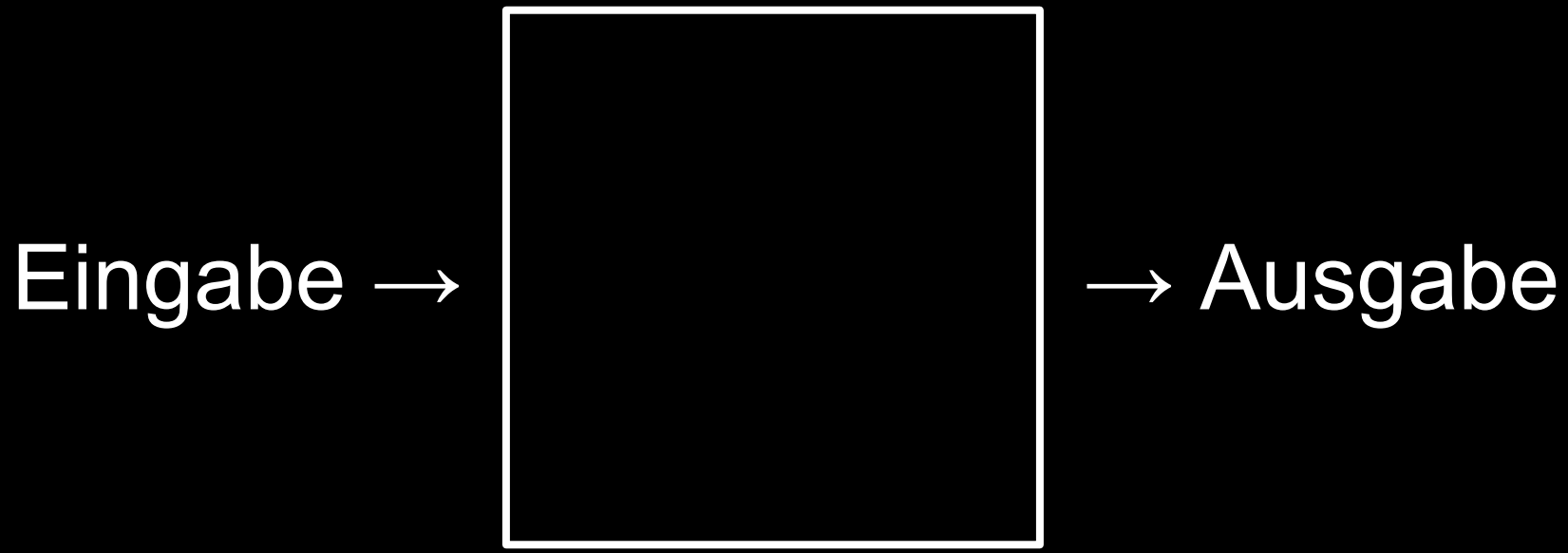


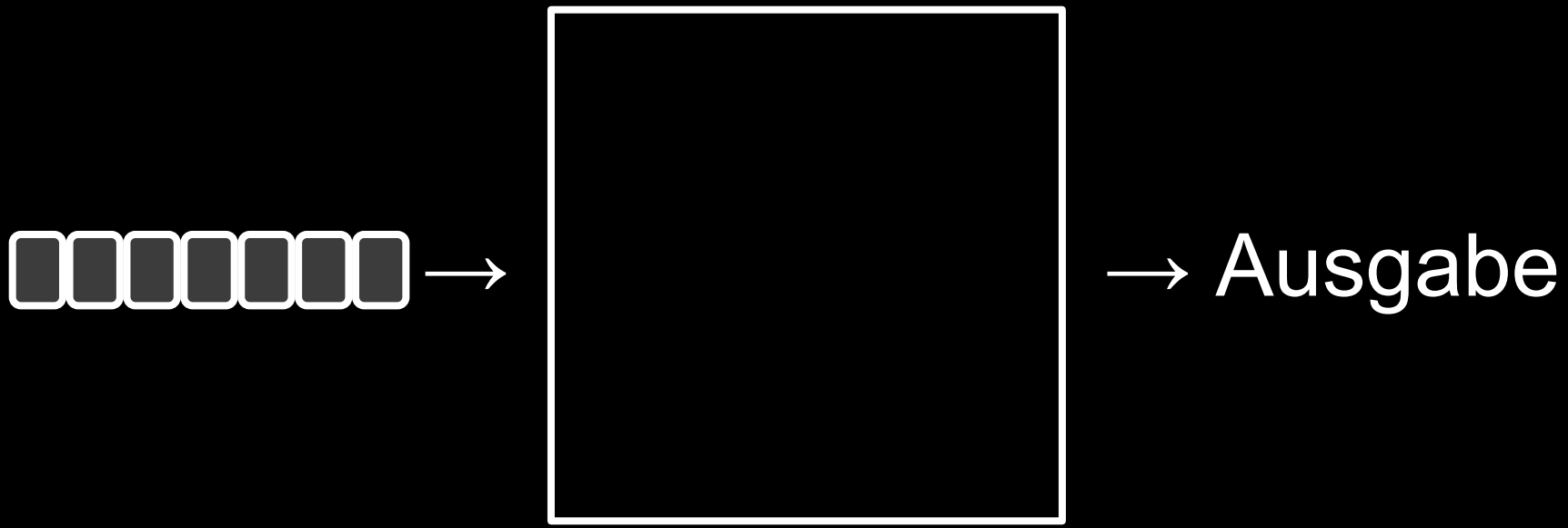

1
5
10
20
50
100
500

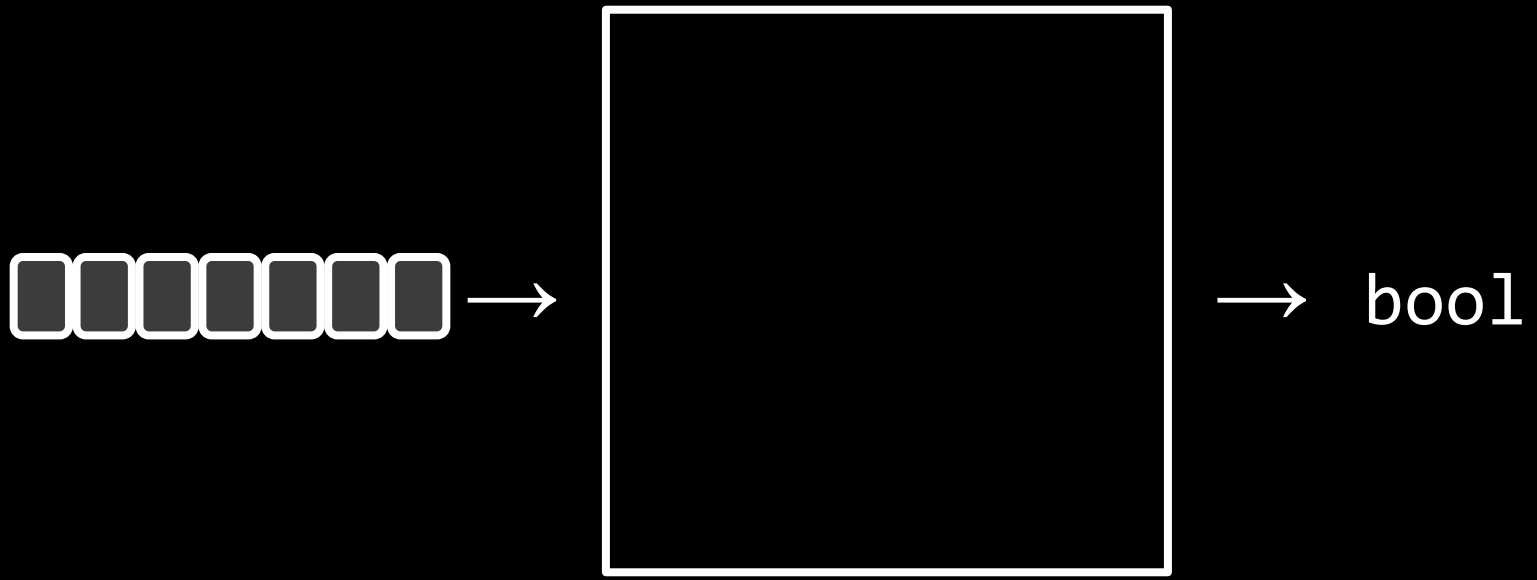

1
5
10
20
50
100
500

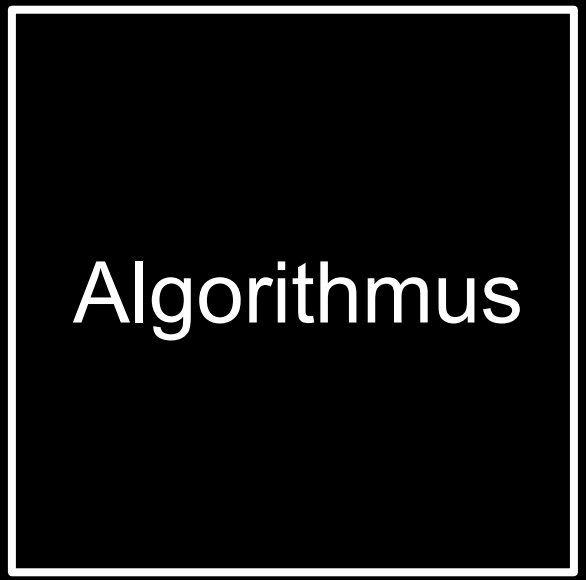

1
5
10
20
50
100
500
[0]
[1]
[2]
[3]
[4]
[5]
[6]

Suche

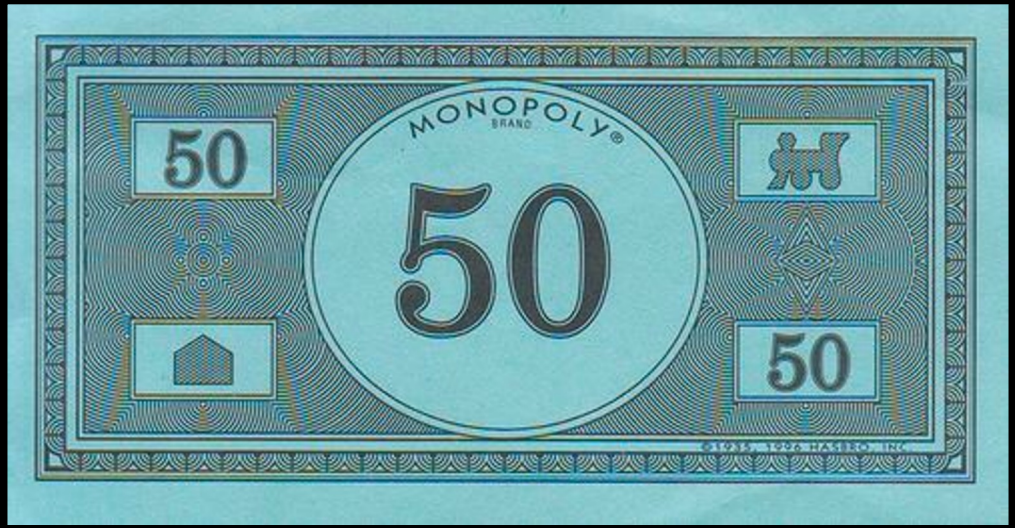

Eingabe →
→ Ausgabe


→
→ Ausgabe


bool


Algorithmus


MONOPOLY
20
50
1
100
500
10
5


MONOPOLY®
BRAND
50
50
50
©1935, 1936 HASBRO, INC.

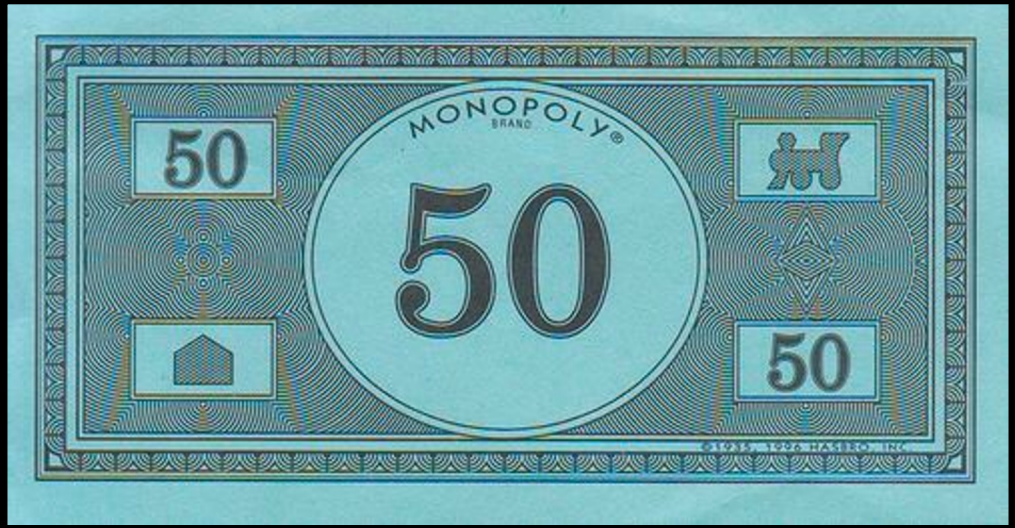
Lineare Suche

```
For each door from left to right
    If 50 is behind door
        Return true
Return false
```

```
For each door from left to right
    If 50 is behind door
        Return true
    Else
        Return false
```

```
For each door from left to right
    If 50 is behind door
        Return true
Return false
```

```
For i from 0 to n-1
    If 50 is behind doors[i]
        Return true
Return false
```


MONOPOLY®
BRAND
50
50
50
©1935, 1936 HASBRO, INC.

Binäre Suche

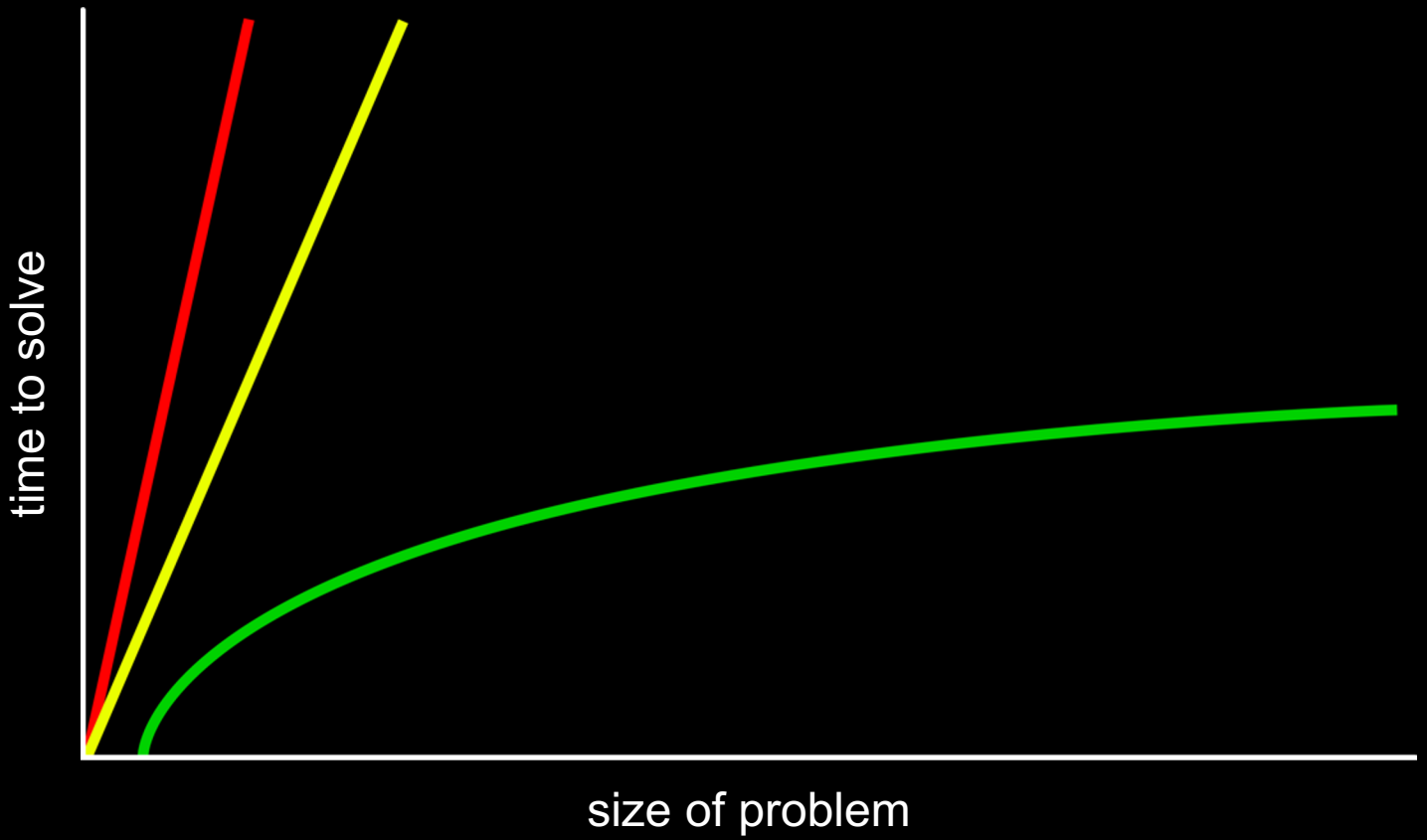
```
If 50 is behind middle door
    Return true
Else if 50 < middle door
    Search left half
Else if 50 > middle door
    Search right half
```

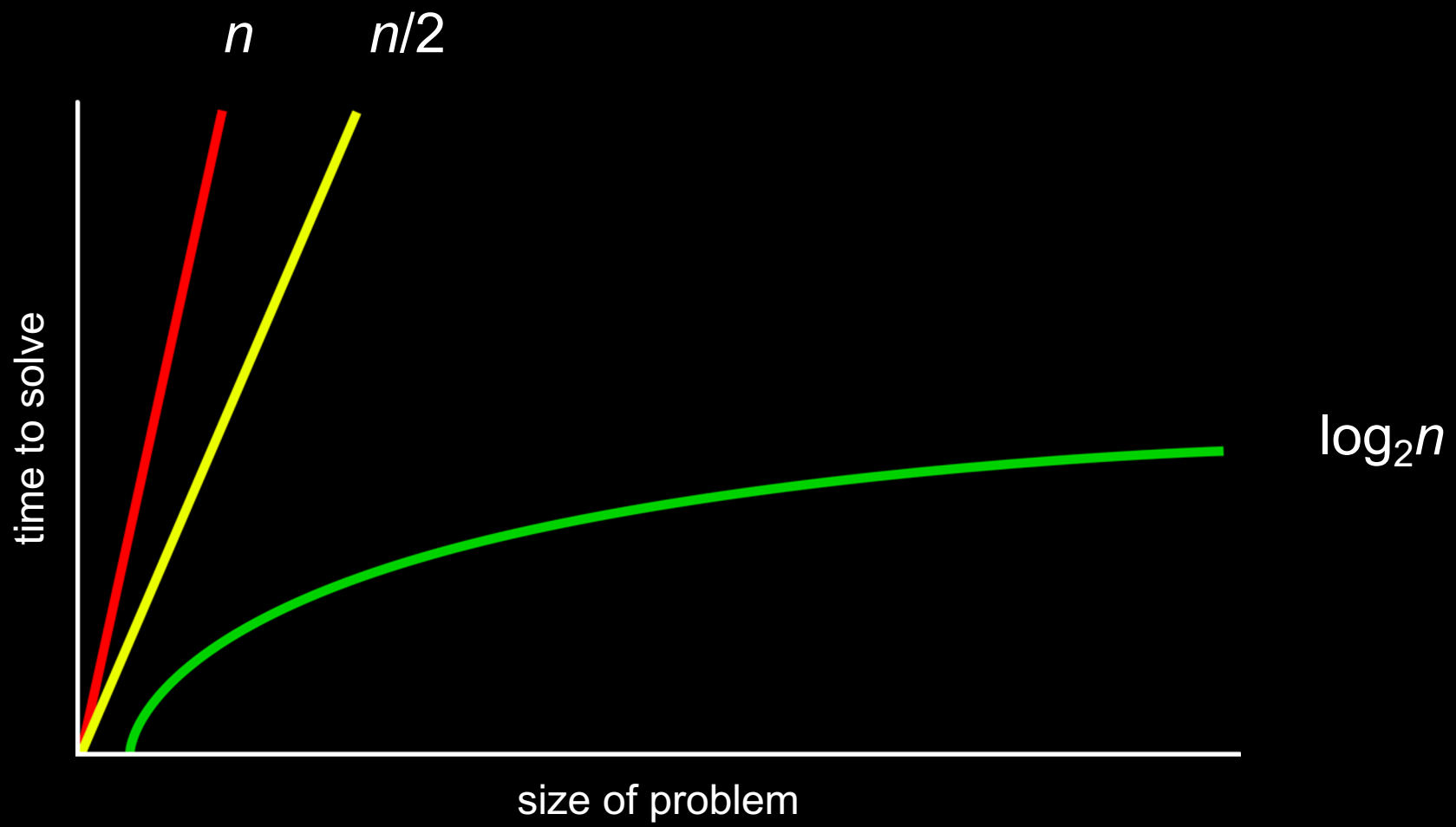
```
If no doors left

If 50 is behind middle door
    Return true
Else if 50 < middle door
    Search left half
Else if 50 > middle door
    Search right half
```

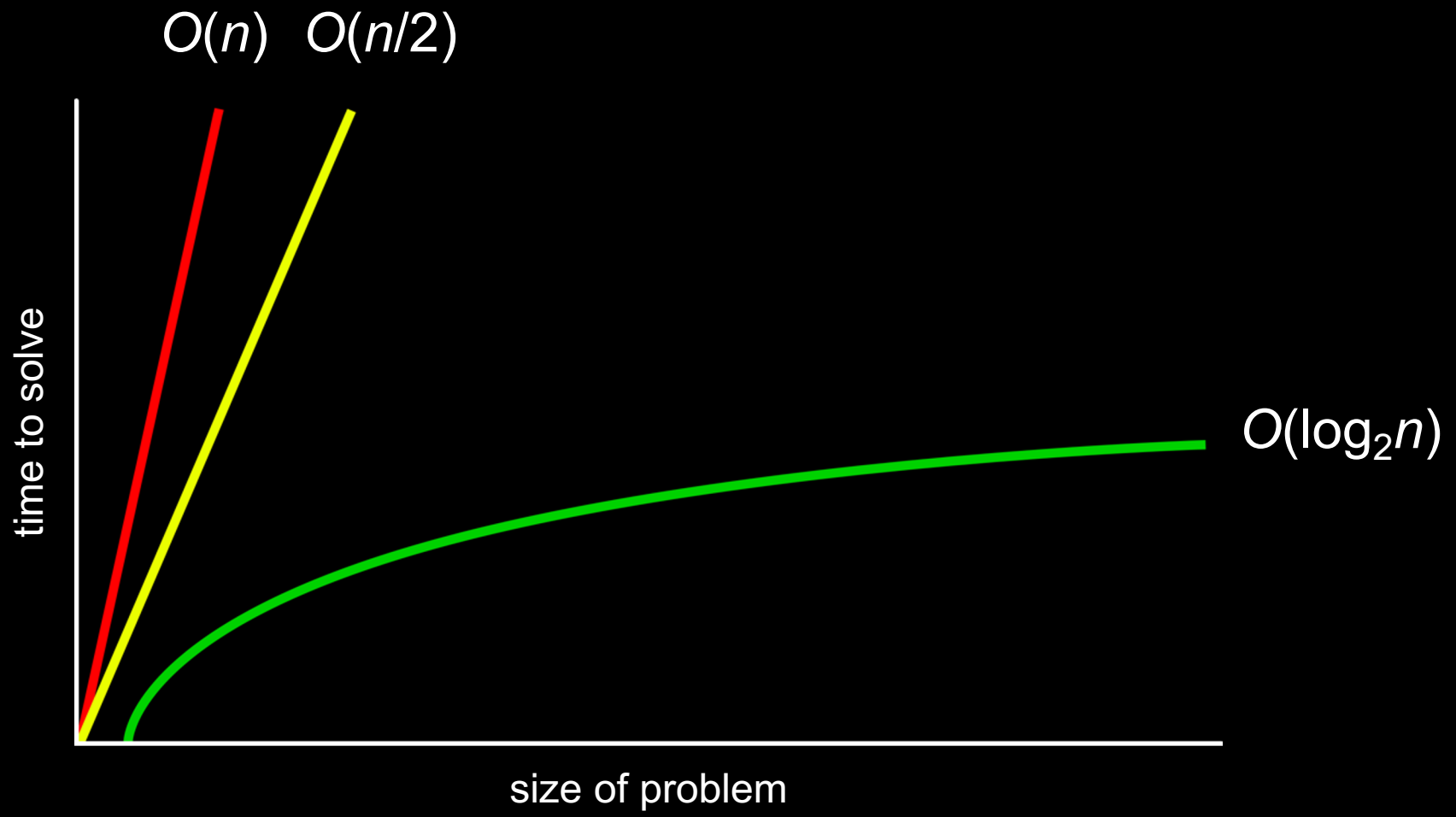
```
If no doors left
    Return false
If 50 is behind middle door
    Return true
Else if 50 < middle door
    Search left half
Else if 50 > middle door
    Search right half
```

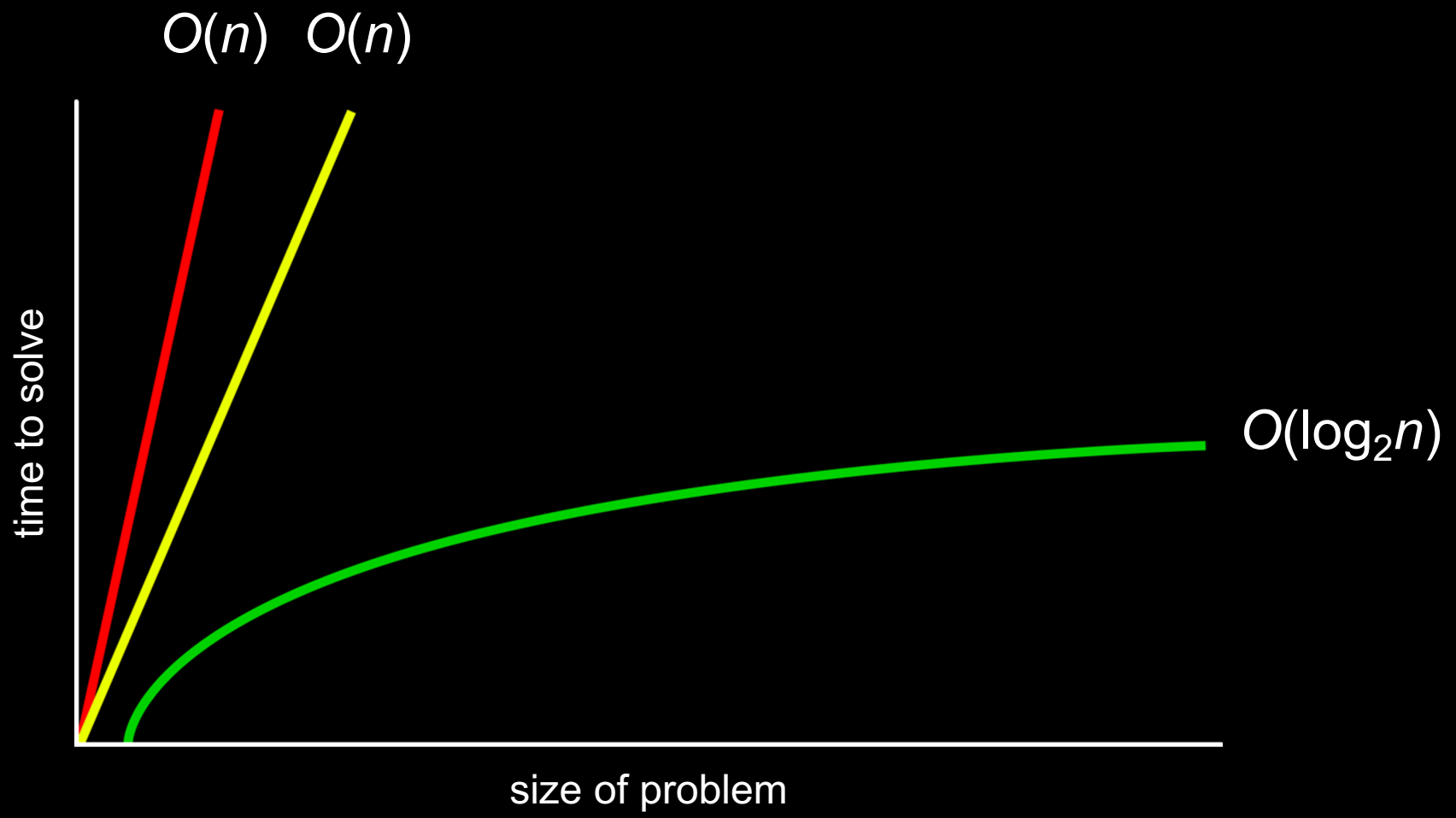
```
If no doors left
    Return false
If 50 is behind doors[middle]
    Return true
Else if 50 < doors[middle]
    Search doors[0] through doors[middle - 1]
Else if 50 > doors[middle]
    Search doors[middle + 1] through doors[n - 1]
```

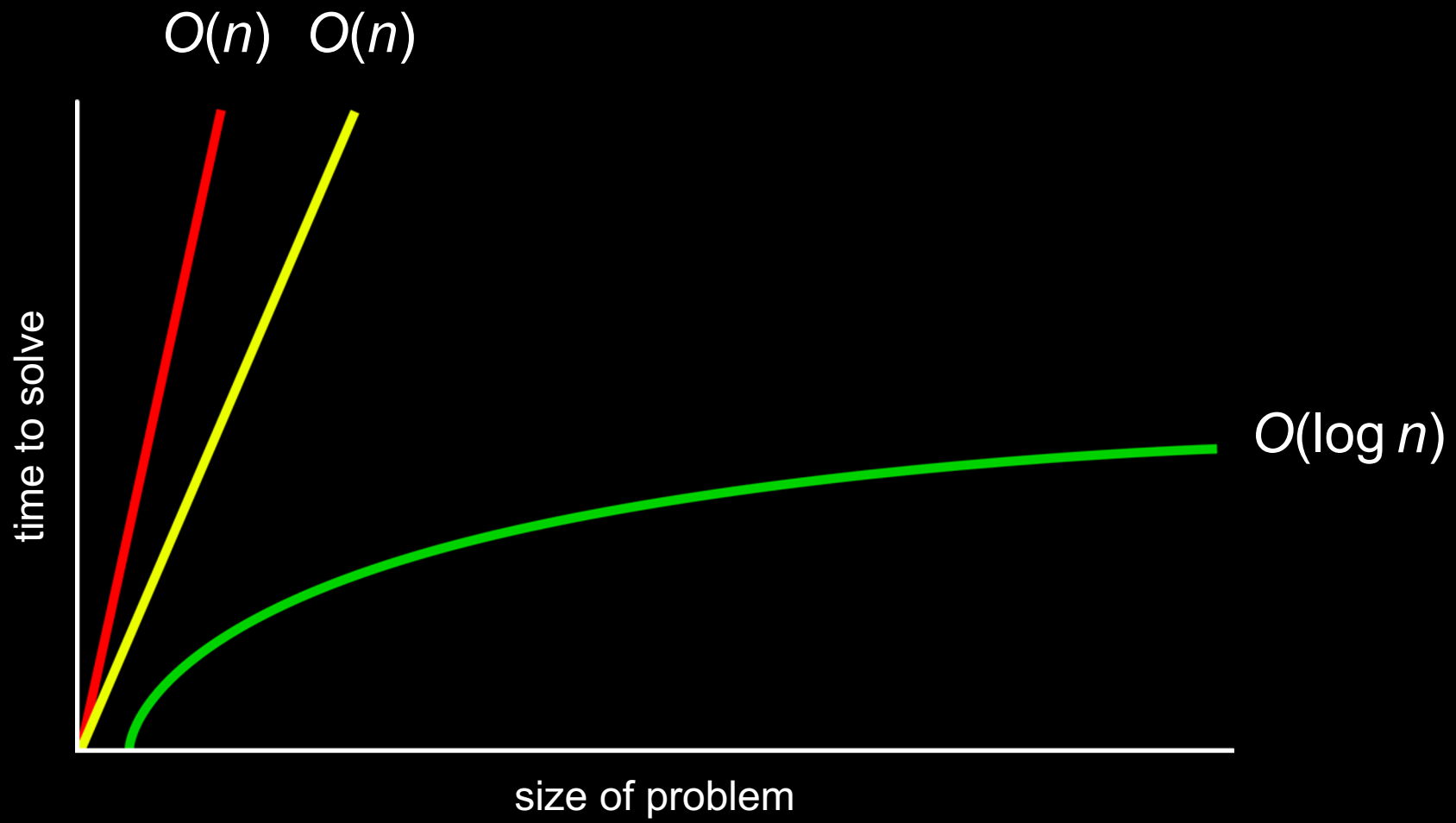
Laufzeit

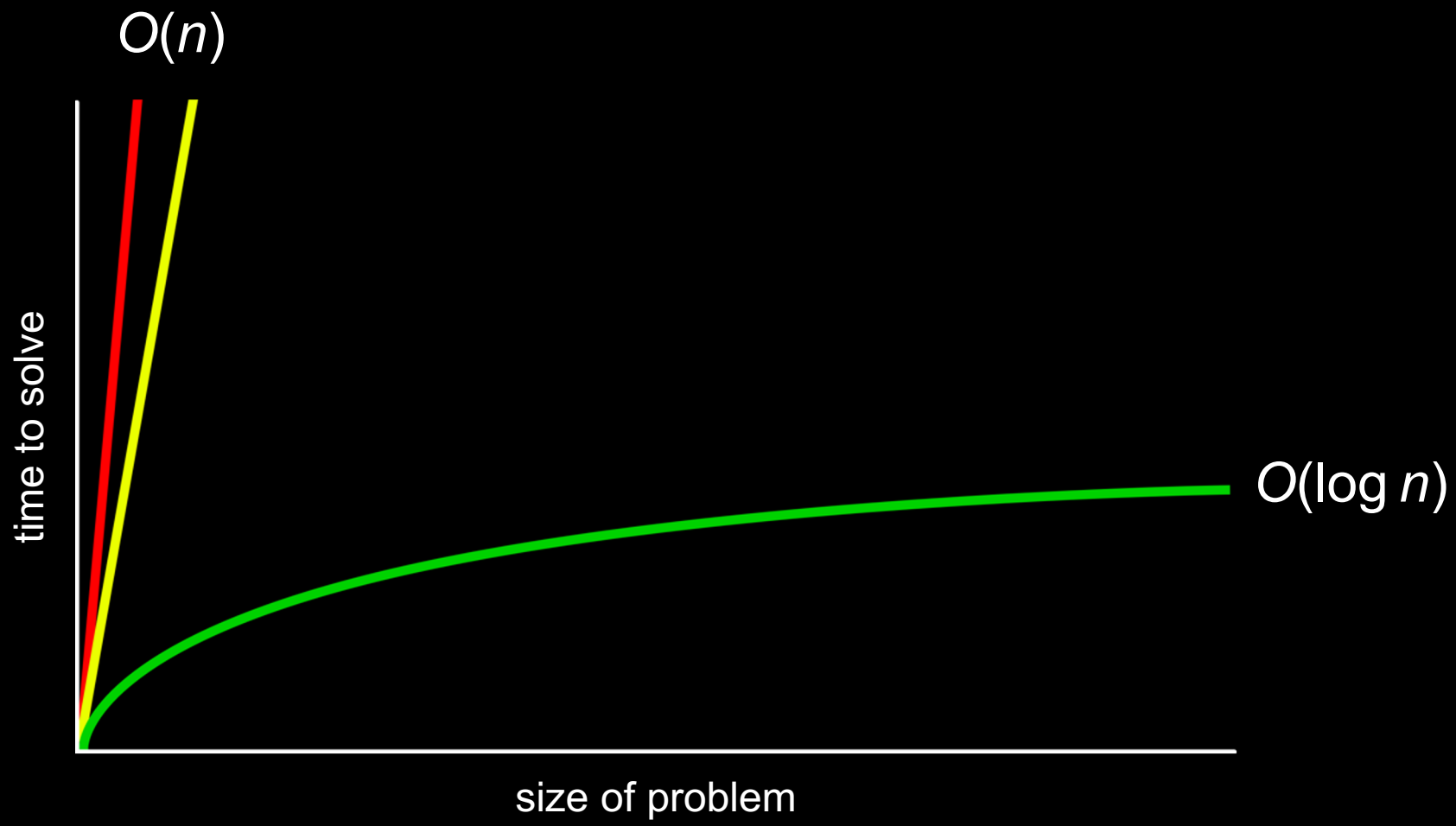

time to solve
size of problem


n
n/2
time to solve
log2n
size of problem


O(n)
O(n/2)
O(log2n)
time to solve
size of problem


O(n)
O(n)
time to solve
O(log2n)
size of problem


O(n)
O(n)
time to solve
O(log n)
size of problem


O(n)
time to solve
O(log n)
size of problem

O

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

$O(n^2)$

$O(n \log n)$

$O(n)$ Lineare Suche

$O(\log n)$

$O(1)$

$O(n^2)$

$O(n \log n)$

$O(n)$ Lineare Suche

$O(\log n)$ Binäre Suche

$O(1)$

$\Omega(n^2)$

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$

$\Omega(n^2)$

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$ Lineare Suche

$\Omega(n^2)$

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$ Lineare Suche, Binäre Suche

Lineare Suche

string.h

manual.cs50.io/#string.h

strcmp

Rückgabewert:
- 0 wenn s1 == s2 (gleich)
- <0 wenn s1 < s2 (alphabetisch früher)
- >0 wenn s1 > s2 (alphabetisch später)

typedef struct

```
person people[]
```

```
string name;
string number;
```

```
typedef struct
{
    string name;
    string number;
}
person;
```

```
typedef struct
{
    string name;
    string number;
} person;
```

```c
#include <cs50.h>
#include <stdio.h>

typedef struct
{
    string titel;
    string kuenstler;
    int jahr;
}
song;

int main(void)
{
    song playlist[2];
    // TODO 1: Erstellen Sie 2 Songs
    // TODO 2: Geben Sie alle Songs ab 2020 aus
}
```

PROBLEM 3

1. Füllen Sie playlist[] mit 2 Songs (Titel, Künstler, Jahr)

2. Schreiben Sie eine Schleife, die alle Song-Titel ausgibt, die nach 2020 veröffentlicht wurden.

```c
#include <cs50.h>
#include <stdio.h>

typedef struct // ...

int main(void)
{
    song playlist[2];
    playlist[0].titel = "Song1";
    playlist[0].kuenstler = "Artist1";
    playlist[0].jahr = 2023;
    // ... repeat for [1]
    for (int i = 0; i < 2; i++)
    {
        if (playlist[i].jahr > 2020) {
            printf("%s\n", playlist[i].titel);
        }
    }
}
```

PROBLEM 3

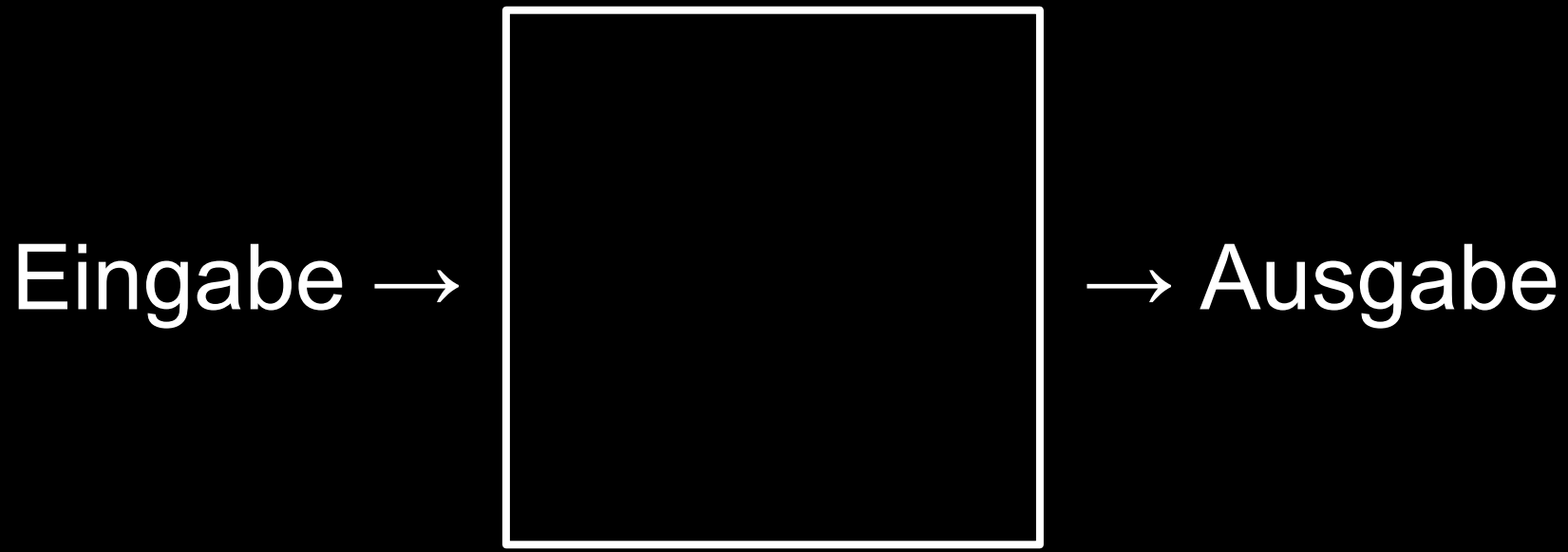
1. Füllen Sie playlist[] mit 2 Songs (Titel, Künstler, Jahr)

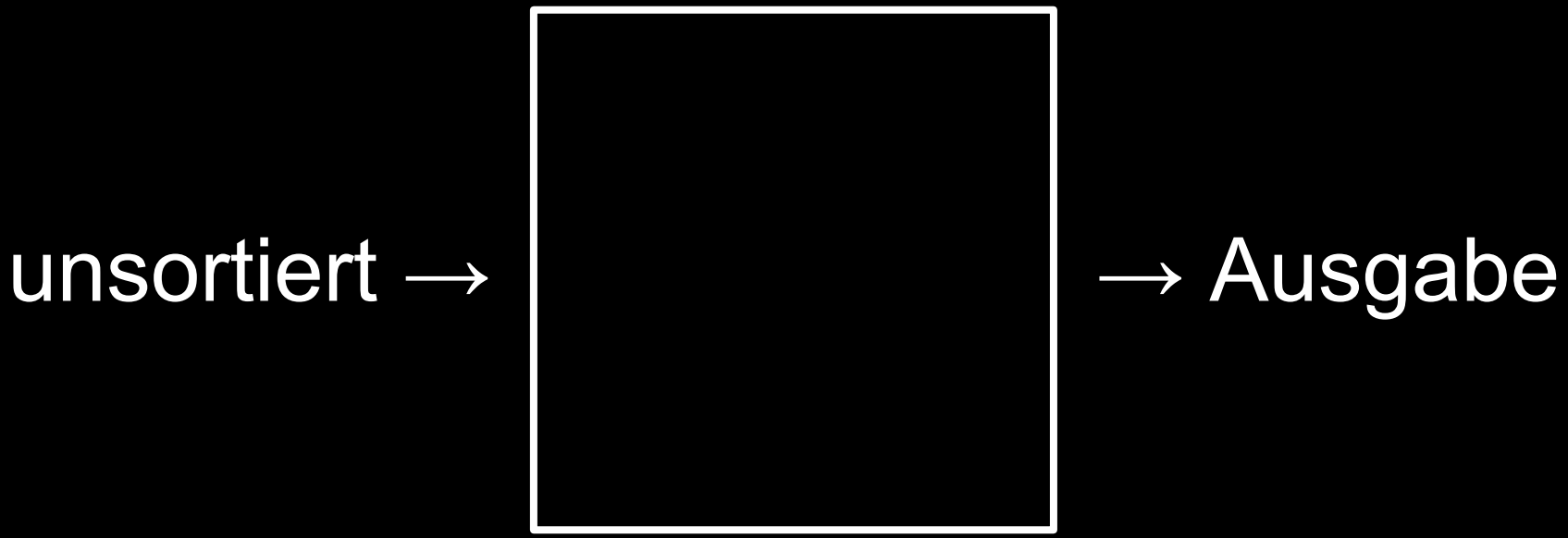
2. Schreiben Sie eine Schleife, die alle Song-Titel ausgibt, die nach 2020 veröffentlicht wurden.

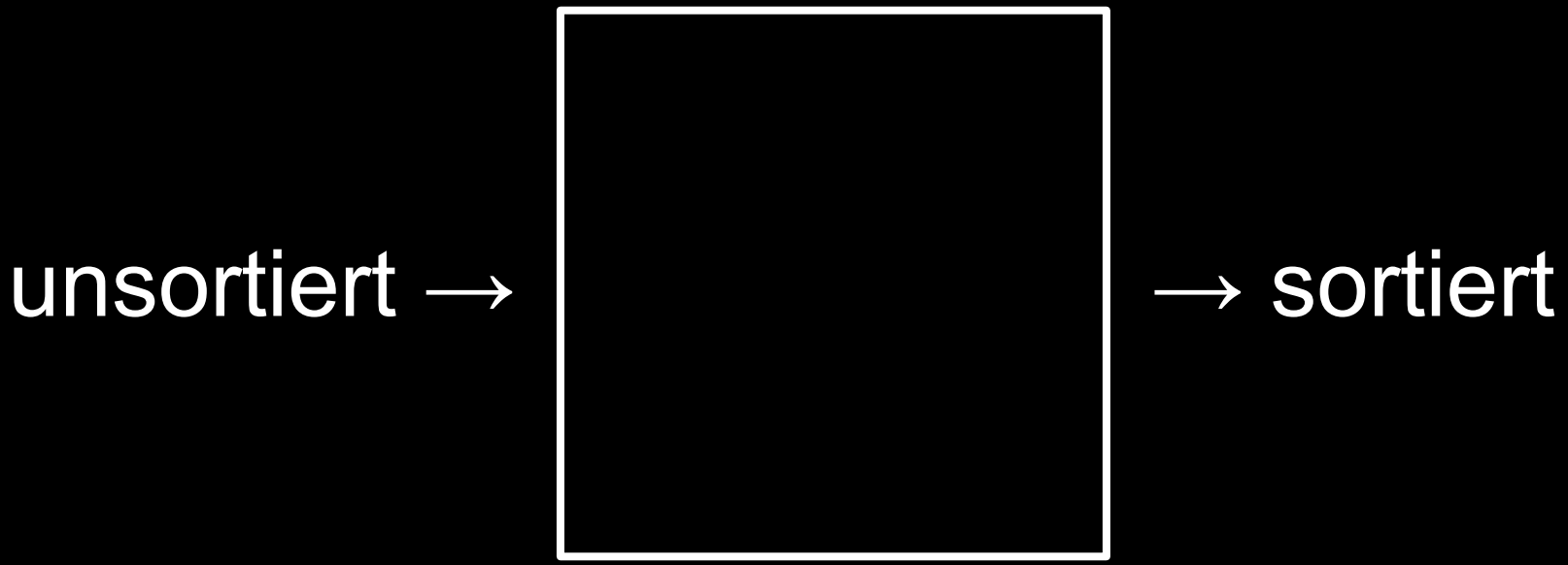
PAUSE

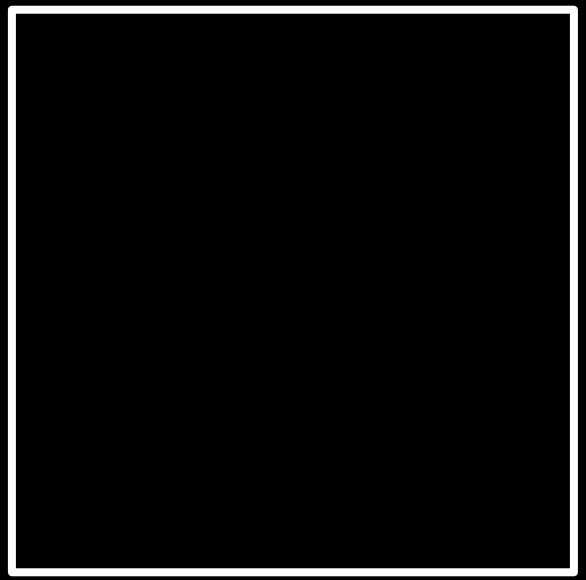
20 min

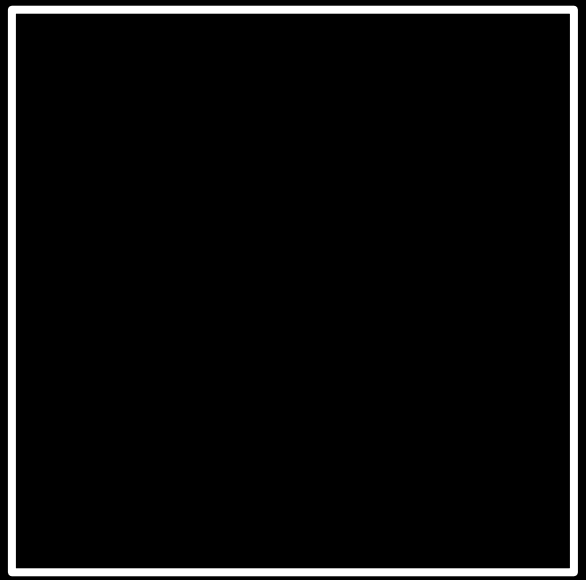
Sortieren


Eingabe →
→ Ausgabe


unsortiert →
→ Ausgabe


unsortiert →
→ sortiert

7 2 5 4 1 6 0 3 →  → sortiert

7 2 5 4 1 6 0 3 →  → 0 1 2 3 4 5 6 7

Selection Sort

```
For i from 0 to n-1
    Finde kleinste Zahl zwischen numbers[i] und numbers[n-1]
    Vertausche kleinste Zahl und numbers[i]
```

$(n - 1)$

$(n - 1) + (n - 2)$

$(n - 1) + (n - 2) + (n - 3)$

$(n - 1) + (n - 2) + (n - 3) + \ldots + 1$

$(n - 1) + (n - 2) + (n - 3) + \ldots + 1$

$n(n - 1)/2$

$(n - 1) + (n - 2) + (n - 3) + ... + 1$

$n(n - 1)/2$

$(n^2 - n)/2$

$(n - 1) + (n - 2) + (n - 3) + ... + 1$

$n(n - 1)/2$

$(n^2 - n)/2$

$n^2/2 - n/2$

$(n - 1) + (n - 2) + (n - 3) + ... + 1$

$n(n - 1)/2$

$(n^2 - n)/2$

$n^2/2 - n/2$

$O(n^2)$

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

$O(n^2)$	Selection Sort
$O(n \log n)$	
$O(n)$	
$O(\log n)$	
$O(1)$	

```
For i from 0 to n-1
    Finde kleinste Zahl zwischen numbers[i] und numbers[n-1]
    Vertausche kleinste Zahl und numbers[i]
```

$\Omega(n^2)$

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$

$\Omega(n^2)$ Selection Sort

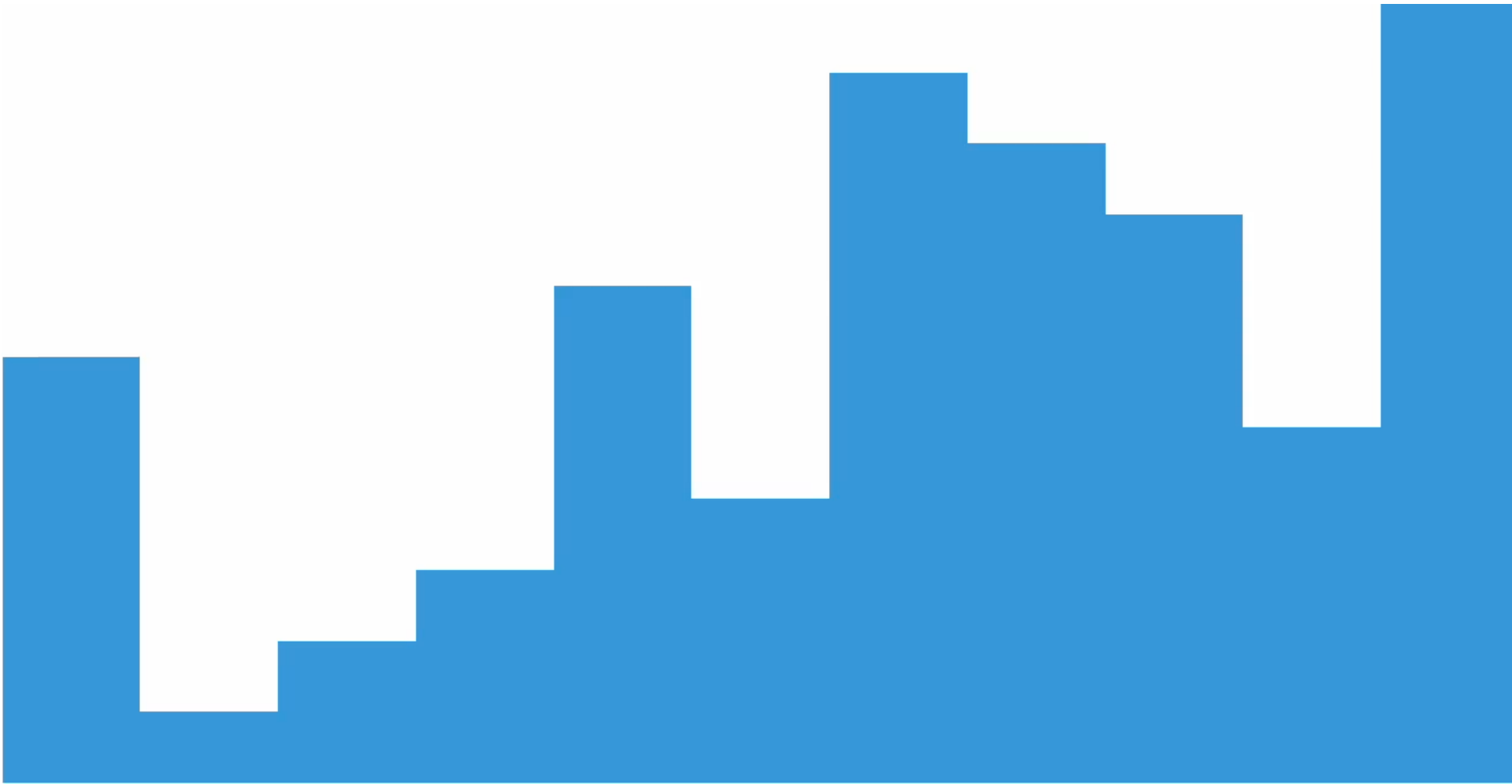
$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$

Bubble Sort



```
Wiederhole n mal
    For i from 0 to n-2
        If numbers[i] and numbers[i+1] in falscher Reihenfolge
            Vertausche sie
```

```
Wiederhole n-1 mal
    For i from 0 to n-2
        If numbers[i] and numbers[i+1] in falscher Reihenfolge
            Vertausche sie
```

$(n - 1) \times (n - 1)$

$(n - 1) \times (n - 1)$

$n^2 - 1n - 1n + 1$

$(n - 1) \times (n - 1)$

$n^2 - 1n - 1n + 1$

$n^2 - 2n + 1$

$(n - 1) \times (n - 1)$

$n^2 - 1n - 1n + 1$

$n^2 - 2n + 1$

$O(n^2)$

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

$O(n^2)$ Bubble Sort

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

```
Wiederhole n-1 mal
    For i from 0 to n-2
        If numbers[i] and numbers[i+1] in falscher Reihenfolge
            Vertausche sie
```

```
Wiederhole n-1 mal
    For i from 0 to n-2
        If numbers[i] and numbers[i+1] in falscher Reihenfolge
            Vertausche sie
    Wenn nichts getauscht wurde
        Ende
```

$\Omega(n^2)$

$\Omega(n \log n)$

$\Omega(n)$

$\Omega(\log n)$

$\Omega(1)$

$\Omega(n^2)$

$\Omega(n \log n)$

$\Omega(n)$ Bubble Sort

$\Omega(\log n)$

$\Omega(1)$

Rekursion

```
Wenn keine Schachtel übrig
    Gib false zurück
Wenn Zahl in mittlerer Schachtel
    Gib true zurück
Oder wenn Zahl < als Zahl in mittlerer Schachtel
    Durchsuche linke Hälfte
Oder wenn Zahl > als Zahl in mittlerer Schachtel
    Durchsuche rechte Hälfte
```

```
Wenn keine Schachtel übrig
    Gib false zurück
Wenn Zahl in mittlerer Schachtel
    Gib true zurück
Oder wenn Zahl < als Zahl in mittlerer Schachtel
    Durchsuche linke Hälfte
Oder wenn Zahl > als Zahl in mittlerer Schachtel
    Durchsuche rechte Hälfte
```

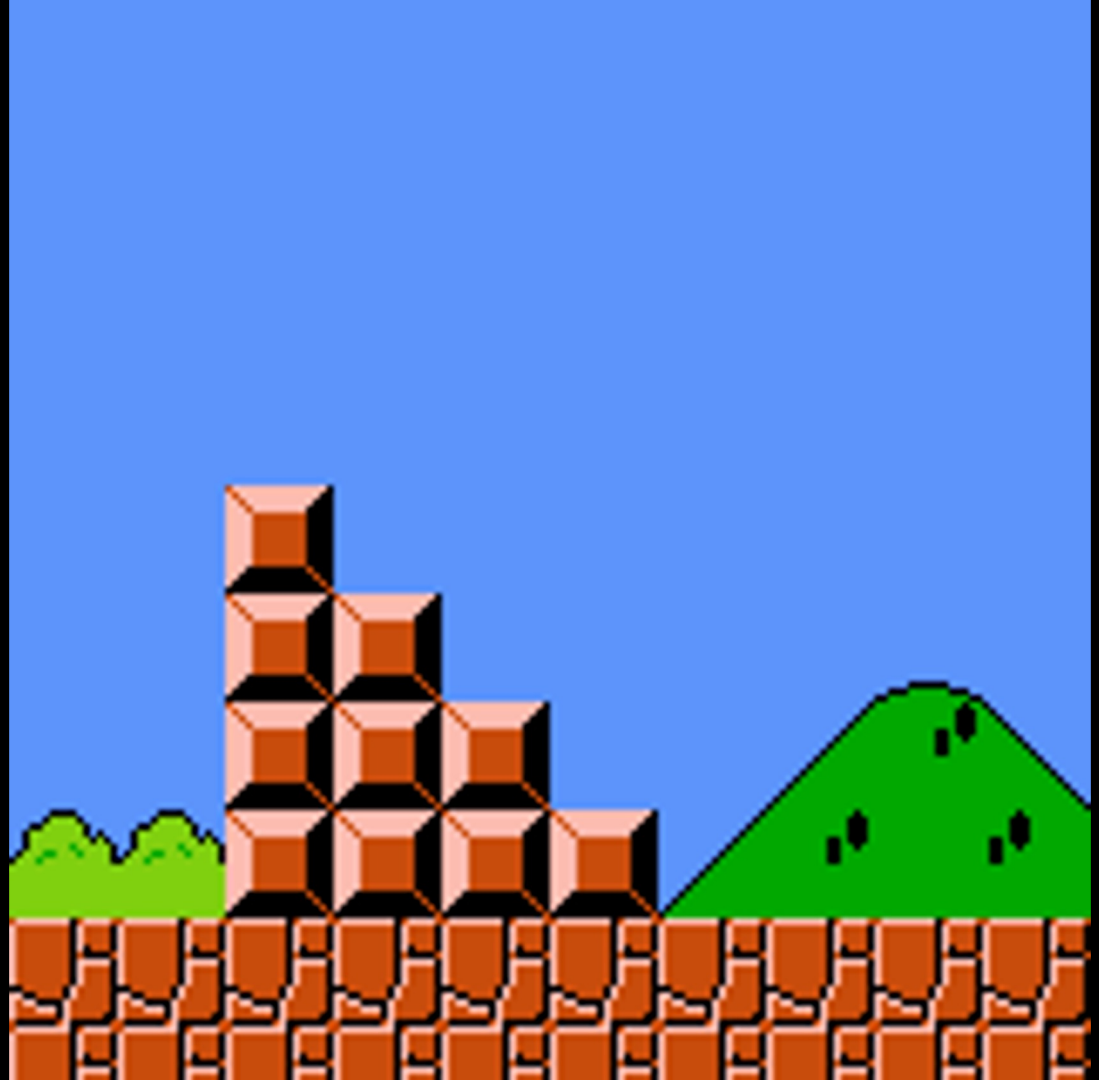
```
1   Nimm Telefonbuch
2   Öffne das Telefonbuch in der Mitte
3   Schau dir die Seite an.
4   Wenn die Person auf der Seite ist
5       Rufe die Person an.
6   Oder wenn die Person weiter vorne steht
7       Gehe zur Mitte der linken Hälfte
8       Gehe zurück zu Zeile 3
9   Oder wenn die Person weiter hinten steht
10      Gehe zur Mitte der rechten Hälfte
11      Gehe zurück zu Zeile 3
12  Sonst
13      Ende
```

```
1   Nimm Telefonbuch
2   Öffne das Telefonbuch in der Mitte
3   Schau dir die Seite an.
4   Wenn die Person auf der Seite ist
5       Rufe die Person an.
6   Oder wenn die Person weiter vorne steht
7       Gehe zur Mitte der linken Hälfte
8       Gehe zurück zu Zeile 3
9   Oder wenn die Person weiter hinten steht
10      Gehe zur Mitte der rechten Hälfte
11      Gehe zurück zu Zeile 3
12  Sonst
13      Ende
```

```
1   Nimm Telefonbuch
2   Öffne das Telefonbuch in der Mitte
3   Schau dir die Seite an.
4   Wenn die Person auf der Seite ist
5       Rufe die Person an.
6   Oder wenn die Person weiter vorne steht
7       Gehe zur Mitte der linken Hälfte
8       Gehe zurück zu Zeile 3
9   Oder wenn die Person weiter hinten steht
10      Gehe zur Mitte der rechten Hälfte
11      Gehe zurück zu Zeile 3
12  Sonst
13      Ende
```

```
1   Nimm Telefonbuch
2   Öffne das Telefonbuch in der Mitte
3   Schau dir die Seite an.
4   Wenn die Person auf der Seite ist
5       Rufe die Person an.
6   Oder wenn die Person weiter vorne steht
7       Durchsuche die linke Hälfte.
8
9   Oder wenn die Person weiter hinten steht
10      Durchsuche die rechte Hälfte.
11
12  Sonst
13      Ende
```

```
1   Nimm Telefonbuch
2   Öffne das Telefonbuch in der Mitte
3   Schau dir die Seite an.
4   Wenn die Person auf der Seite ist
5       Rufe die Person an.
6   Oder wenn die Person weiter vorne steht
7       Durchsuche die linke Hälfte.
8   Oder wenn die Person weiter hinten steht
9       Durchsuche die rechte Hälfte.
10  Sonst
11      Ende
```



google.com/search?q=recursion

Merge Sort

Sortiere linke Hälfte der Zahlen.
Sortiere rechte Hälfte der Zahlen.
Führe beide hälften zusammen.

```
Wenn es nur eine Zahl ist
    Ende
Sonst
    Sortiere linke Hälfte der Zahlen.
    Sortiere rechte Hälfte der Zahlen.
    Führe beide hälften zusammen.
```

```
Wenn es nur eine Zahl ist
    Ende
Sonst
    Sortiere linke Hälfte der Zahlen.
    Sortiere rechte Hälfte der Zahlen.
    Führe beide hälften zusammen (= merge)
```

1 3 4 6 0 2 5 7

```
Wenn es nur eine Zahl ist
    Ende
Sonst
    Sortiere linke Hälfte der Zahlen.
    Sortiere rechte Hälfte der Zahlen.
    Führe beide hälften zusammen.
```

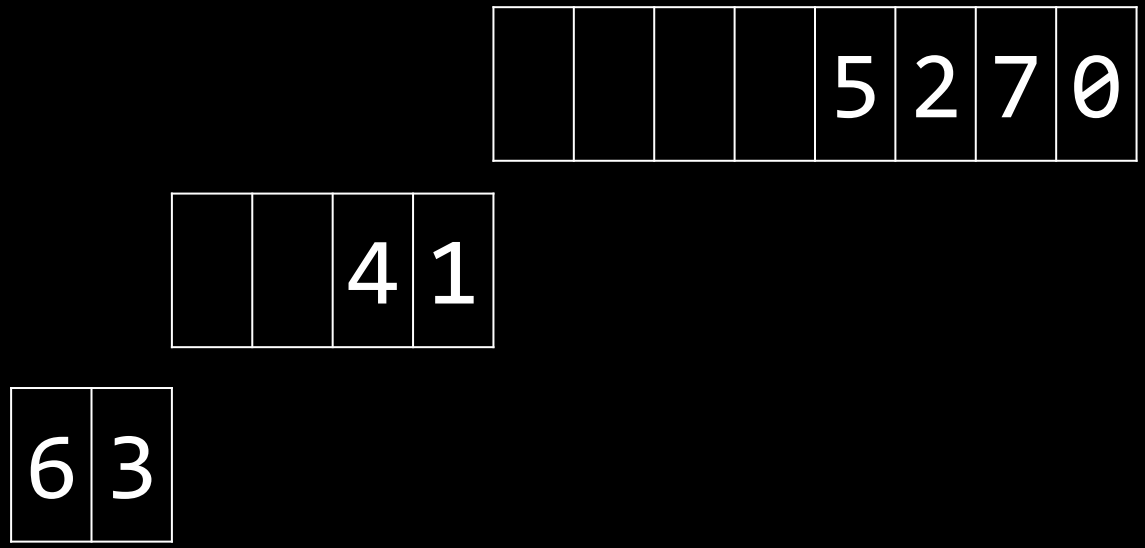
6 3 4 1 5 2 7 0

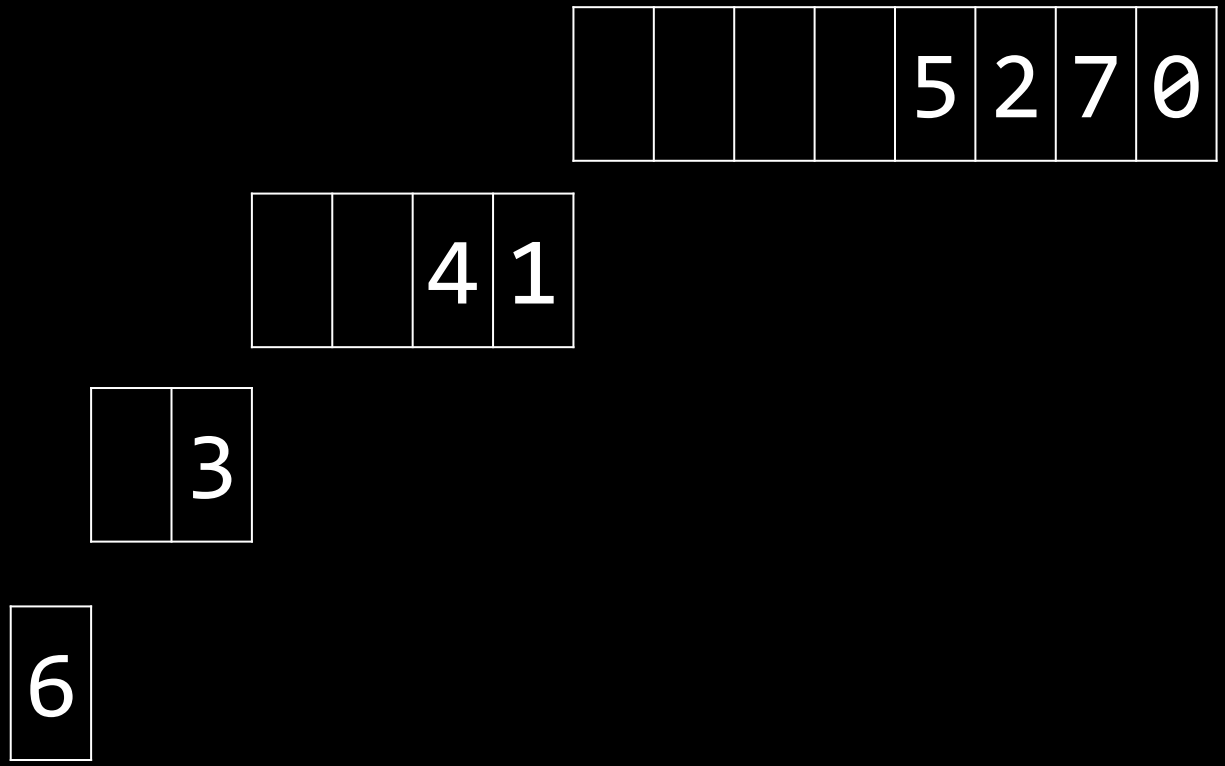
6 3 4 1 5 2 7 0

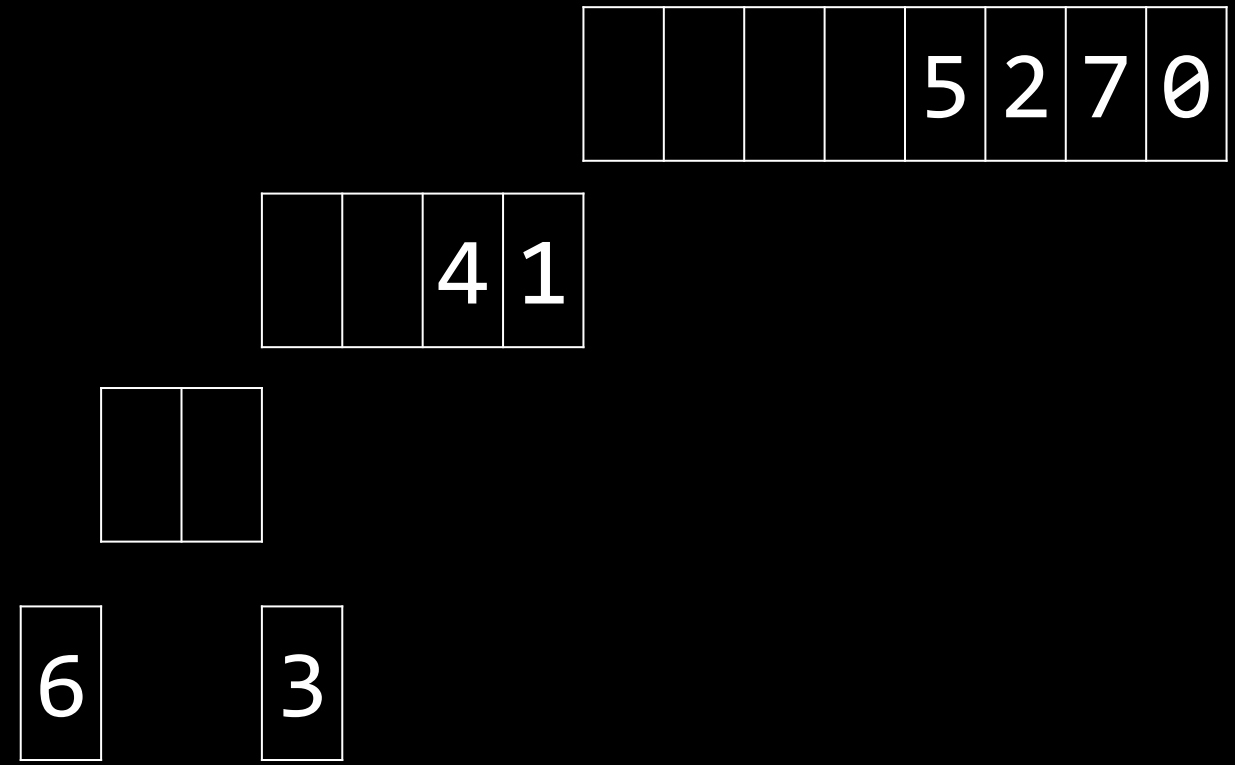

6 3 4 1 5 2 7 0

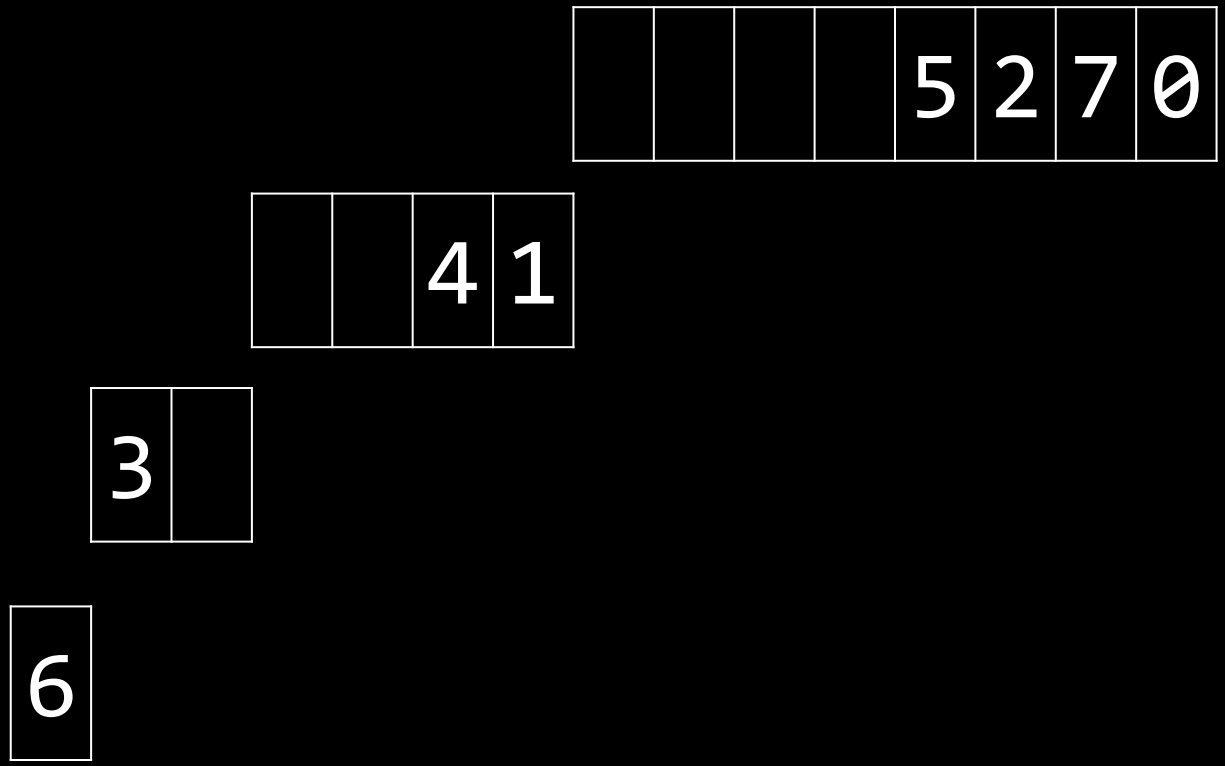

6341

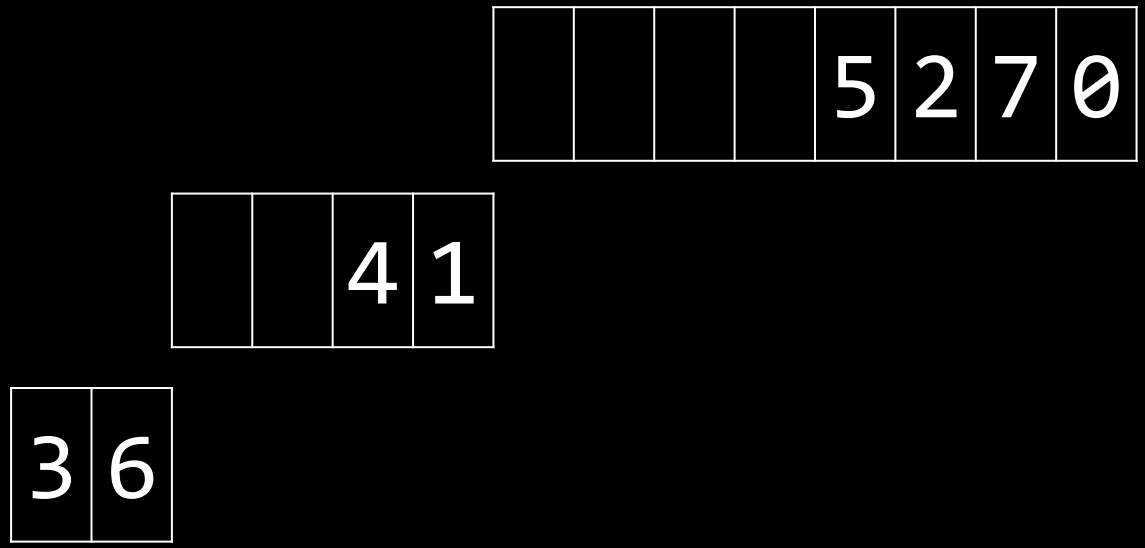

5270

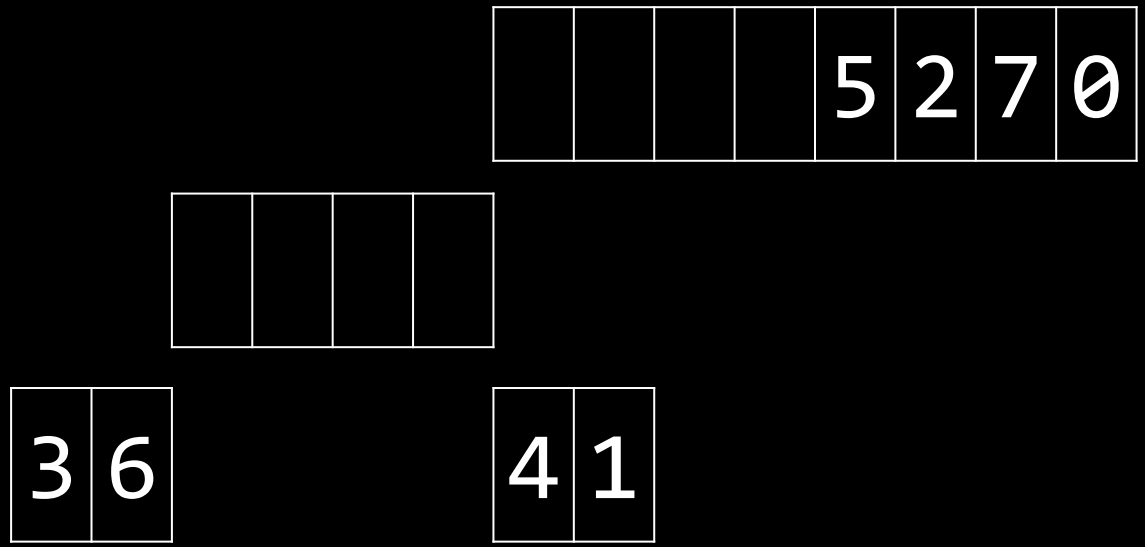

5270
41
63

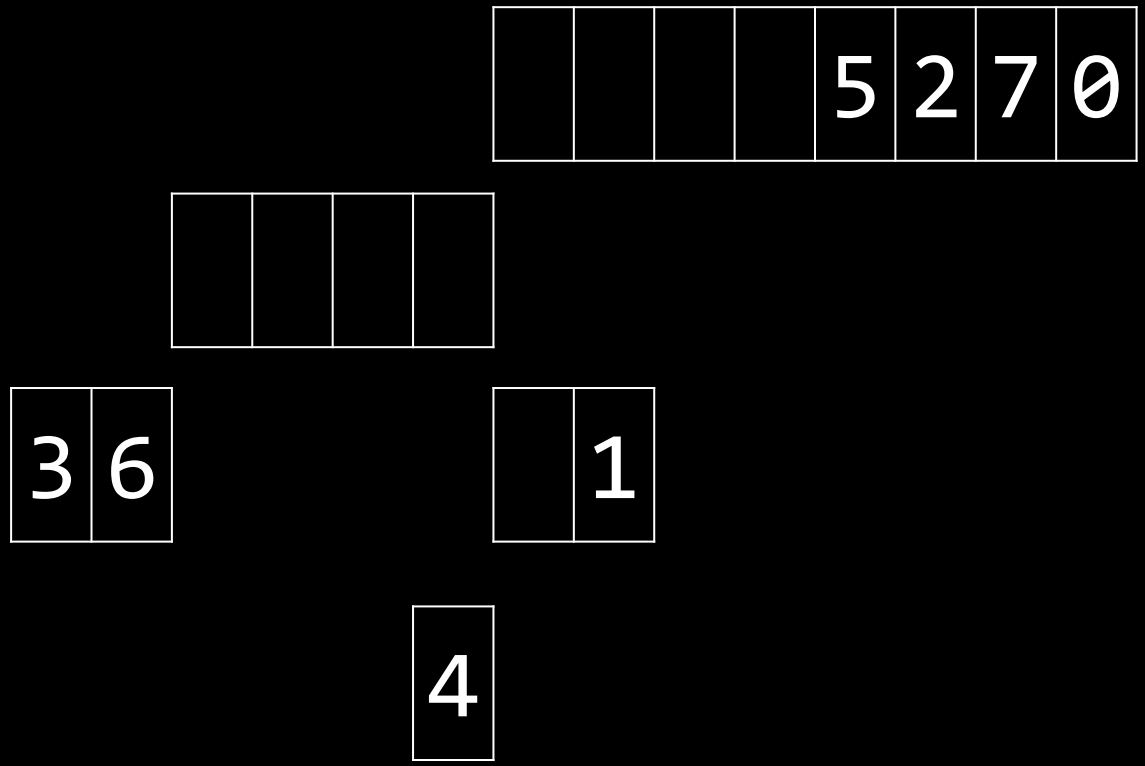

5270
41
3
6

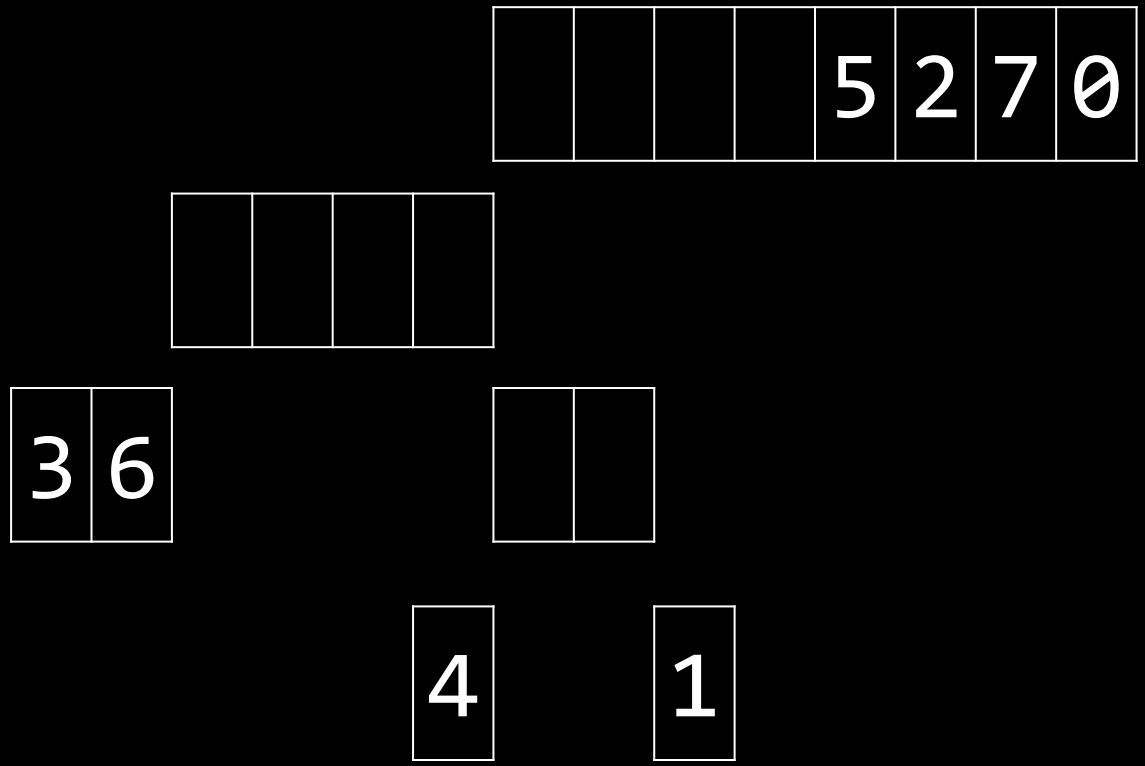

5270
41
6
3

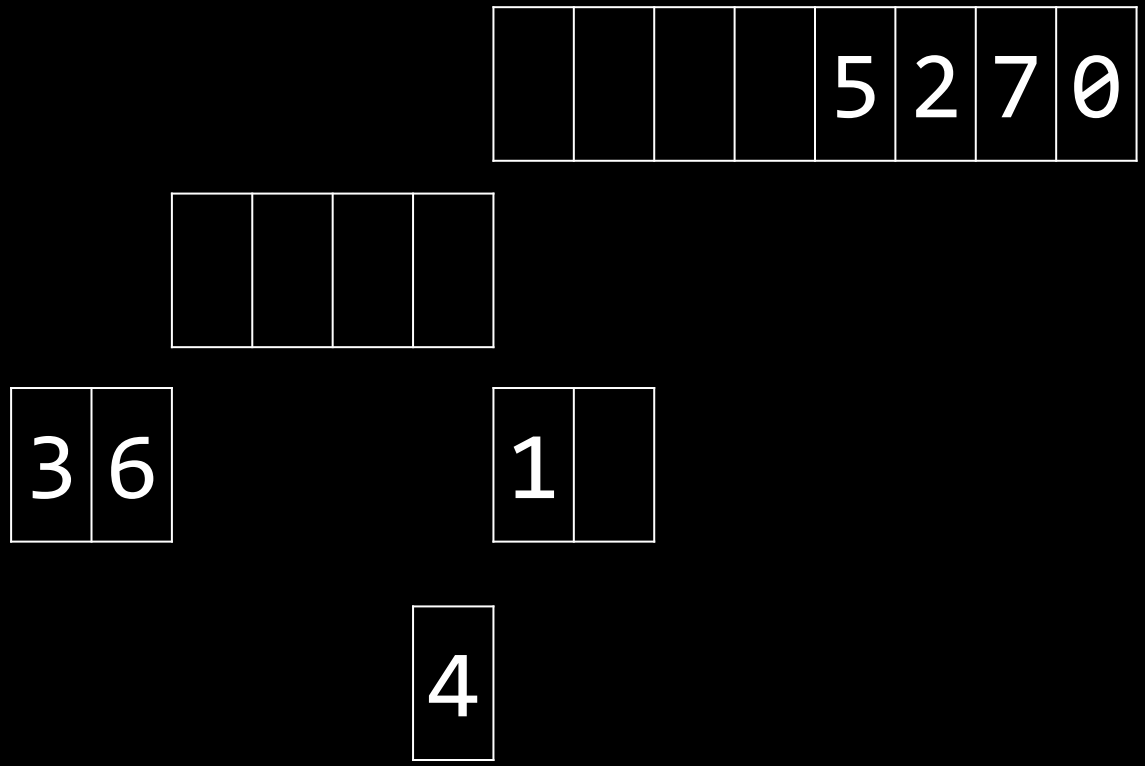

5270
41
3
6

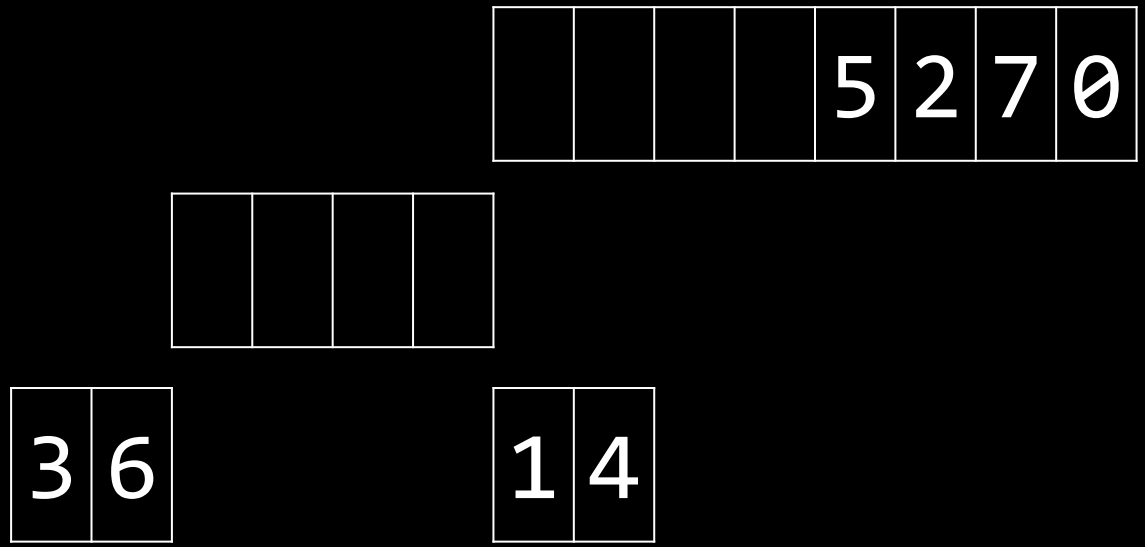

5270
41
36

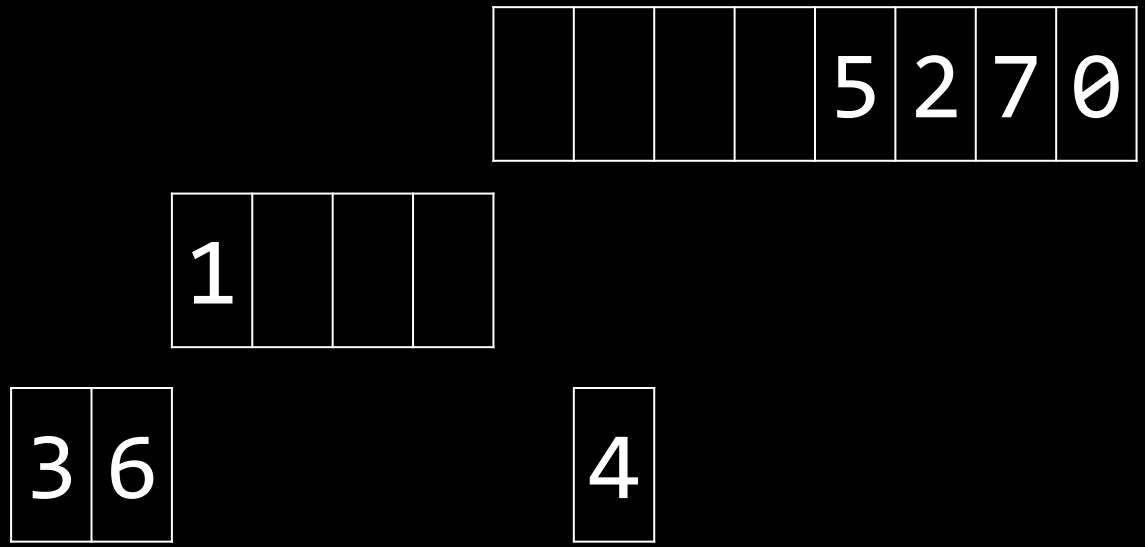

5270
36
41

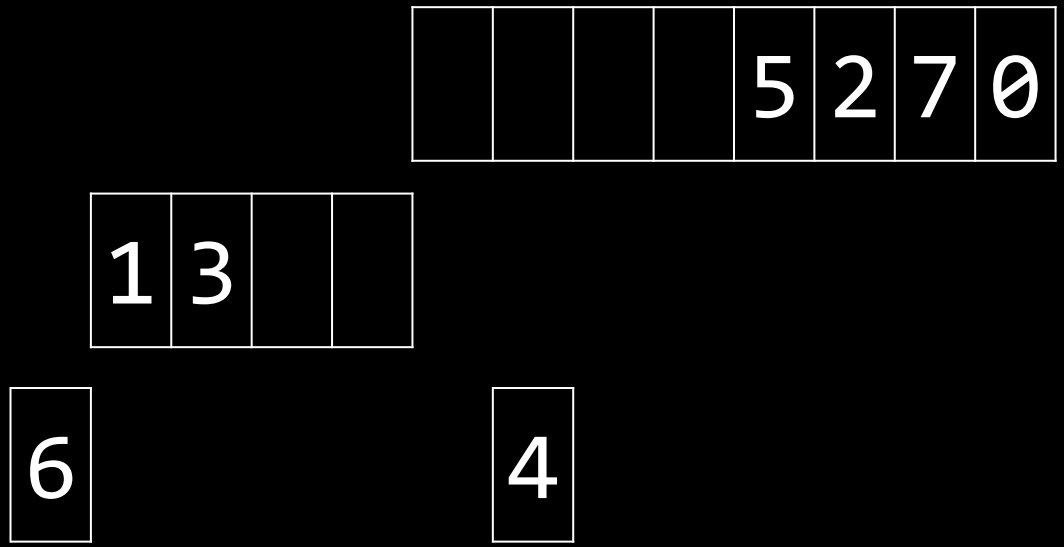

5270
36
1
4

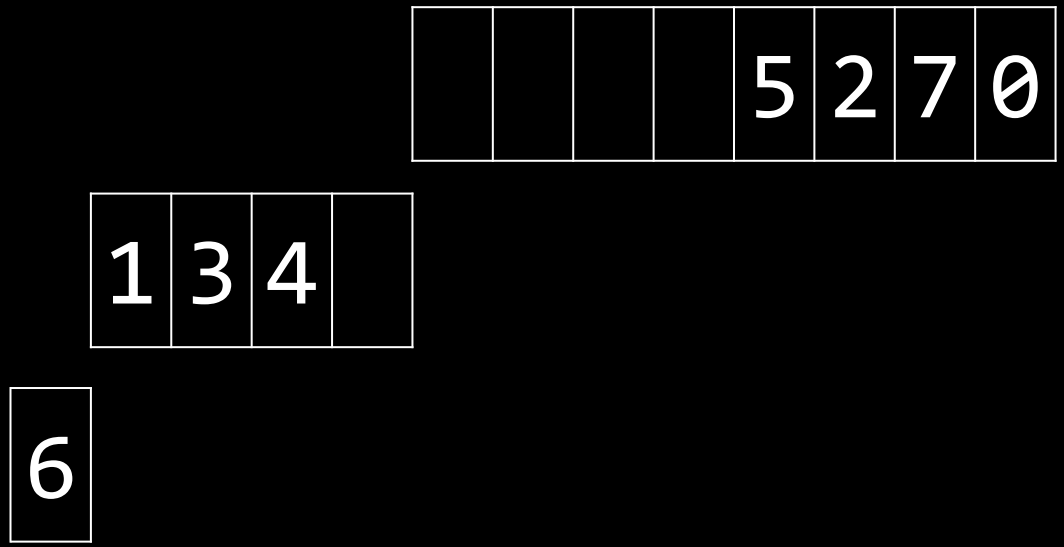

5270
36
4
1


5270
36
1
4


5270
36
14

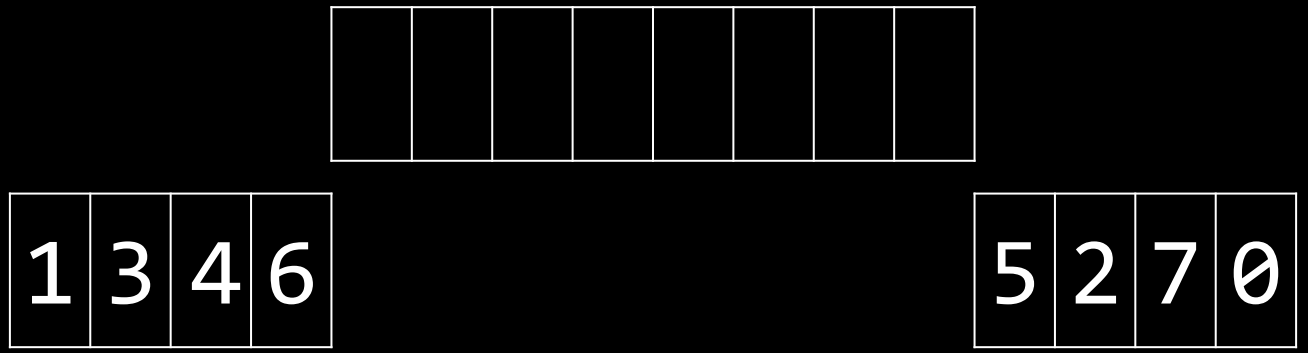

5270
1
36
4

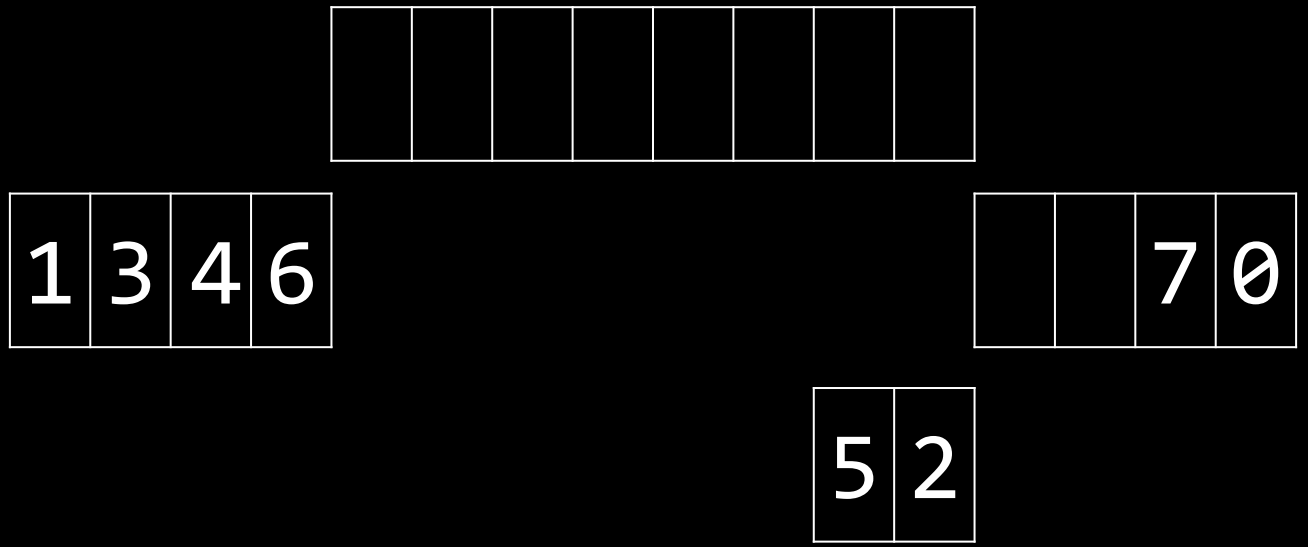

5270
13
6
4

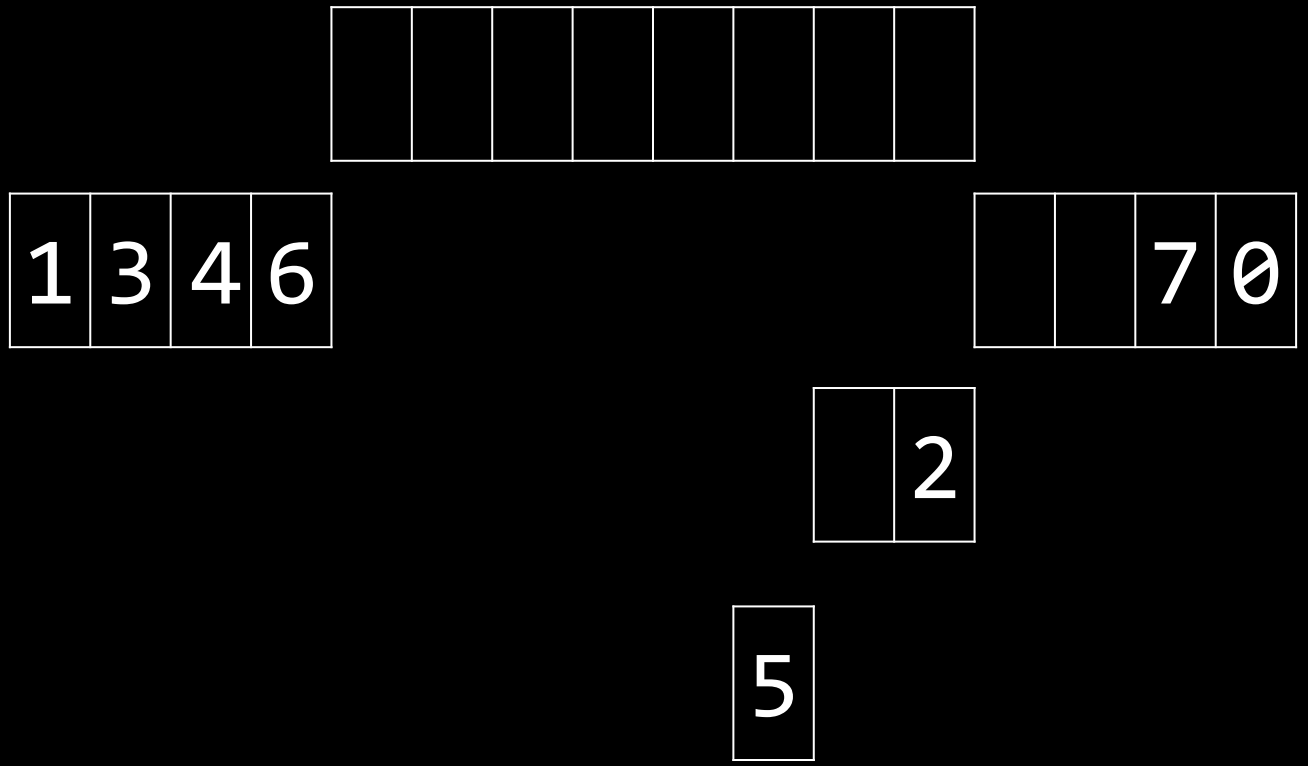

5270
134
6

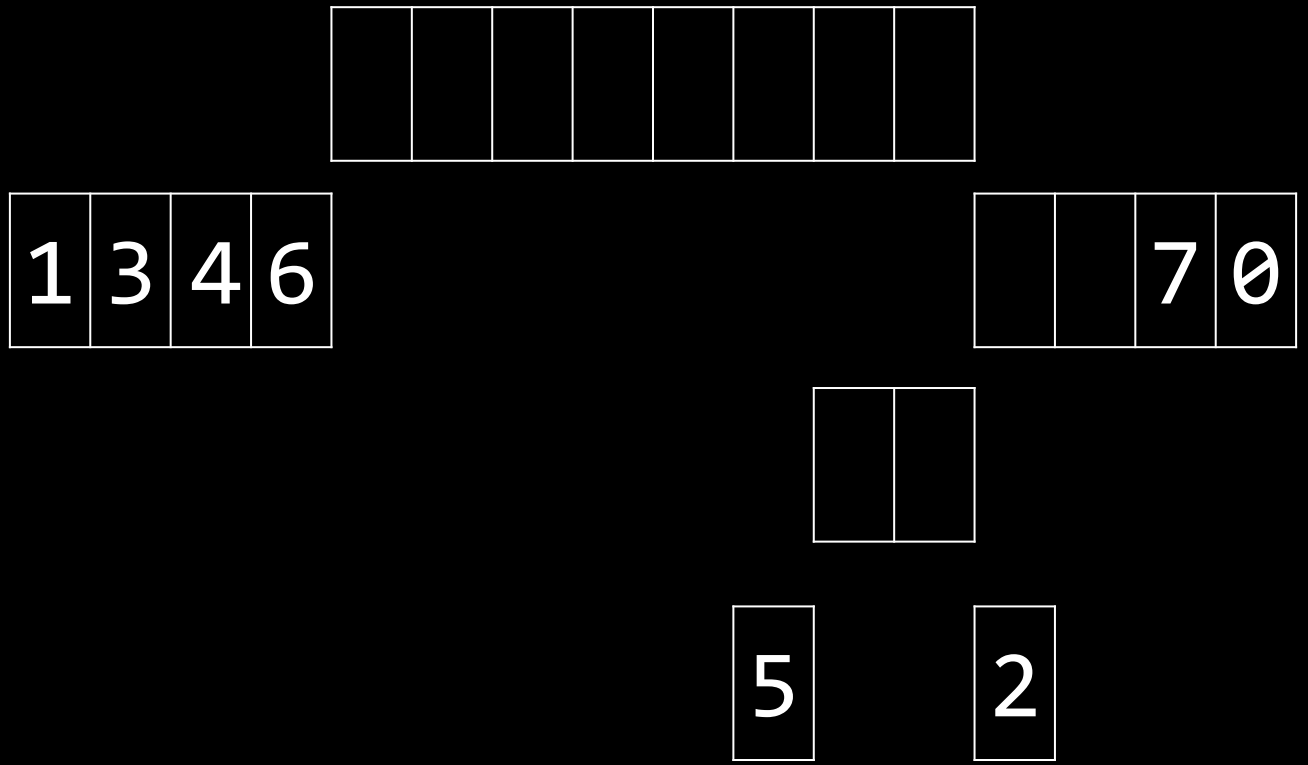

1346

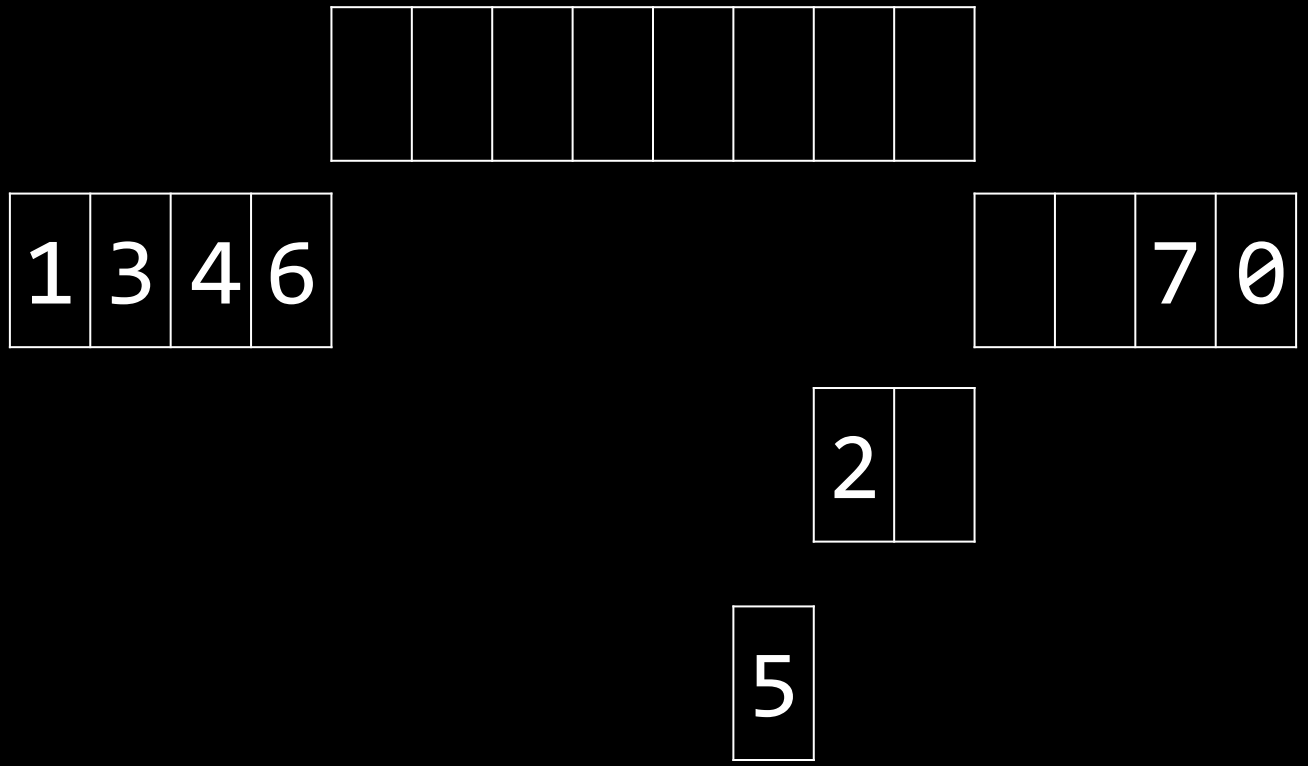

5270

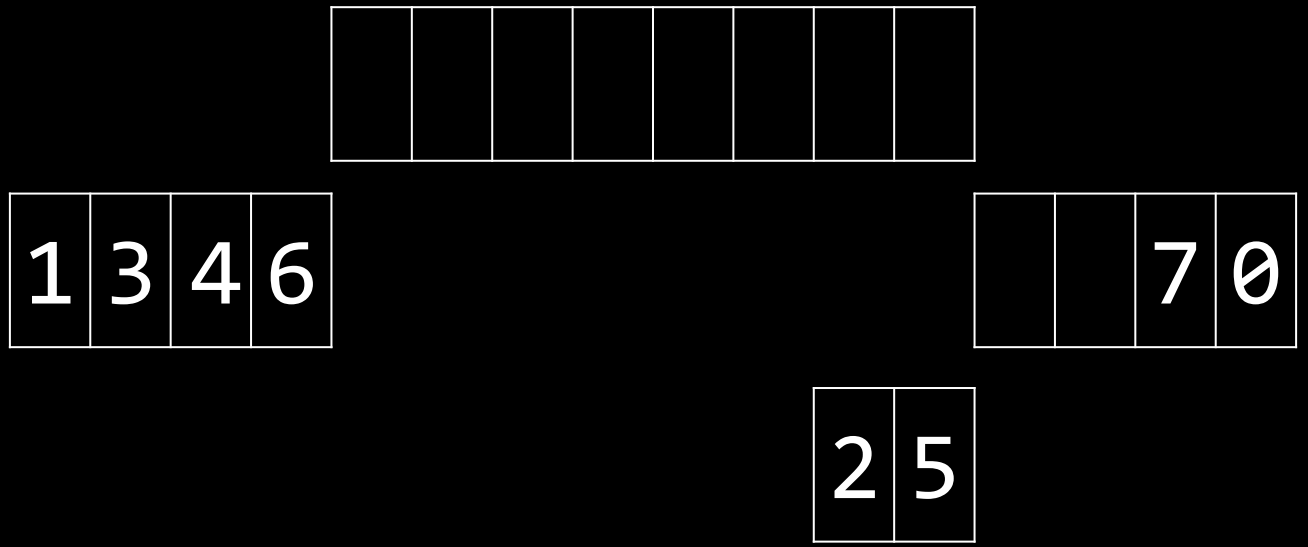

1 3 4 6
5 2 7 0

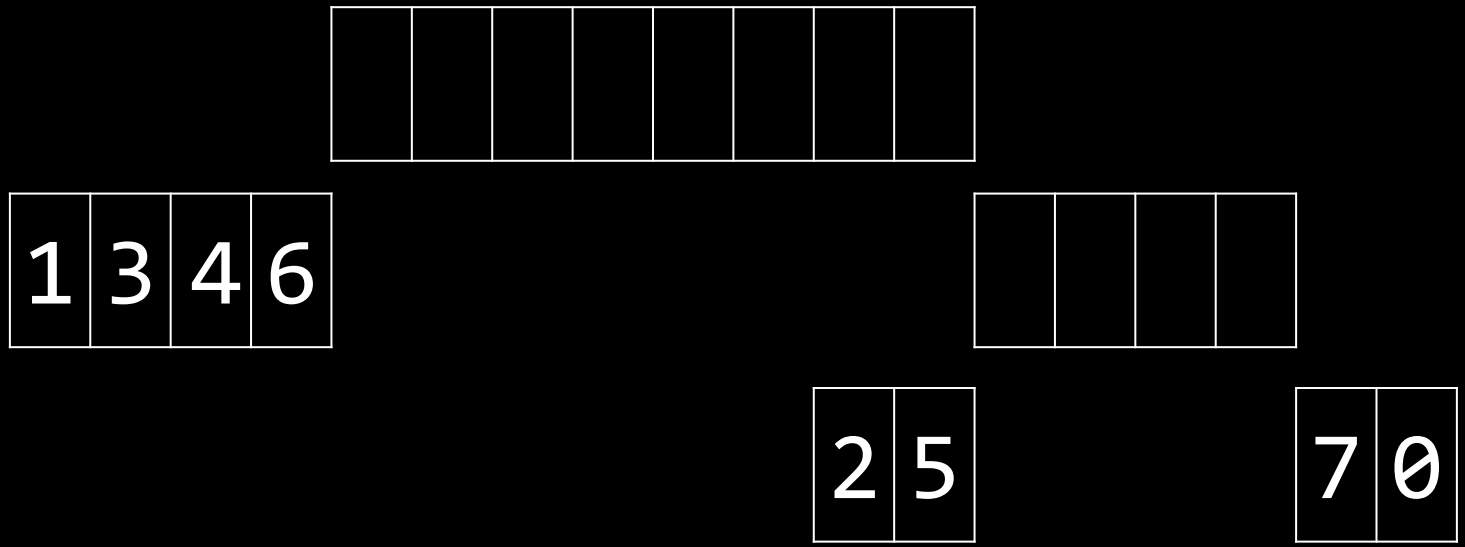

1 3 4 6
7 0
5 2

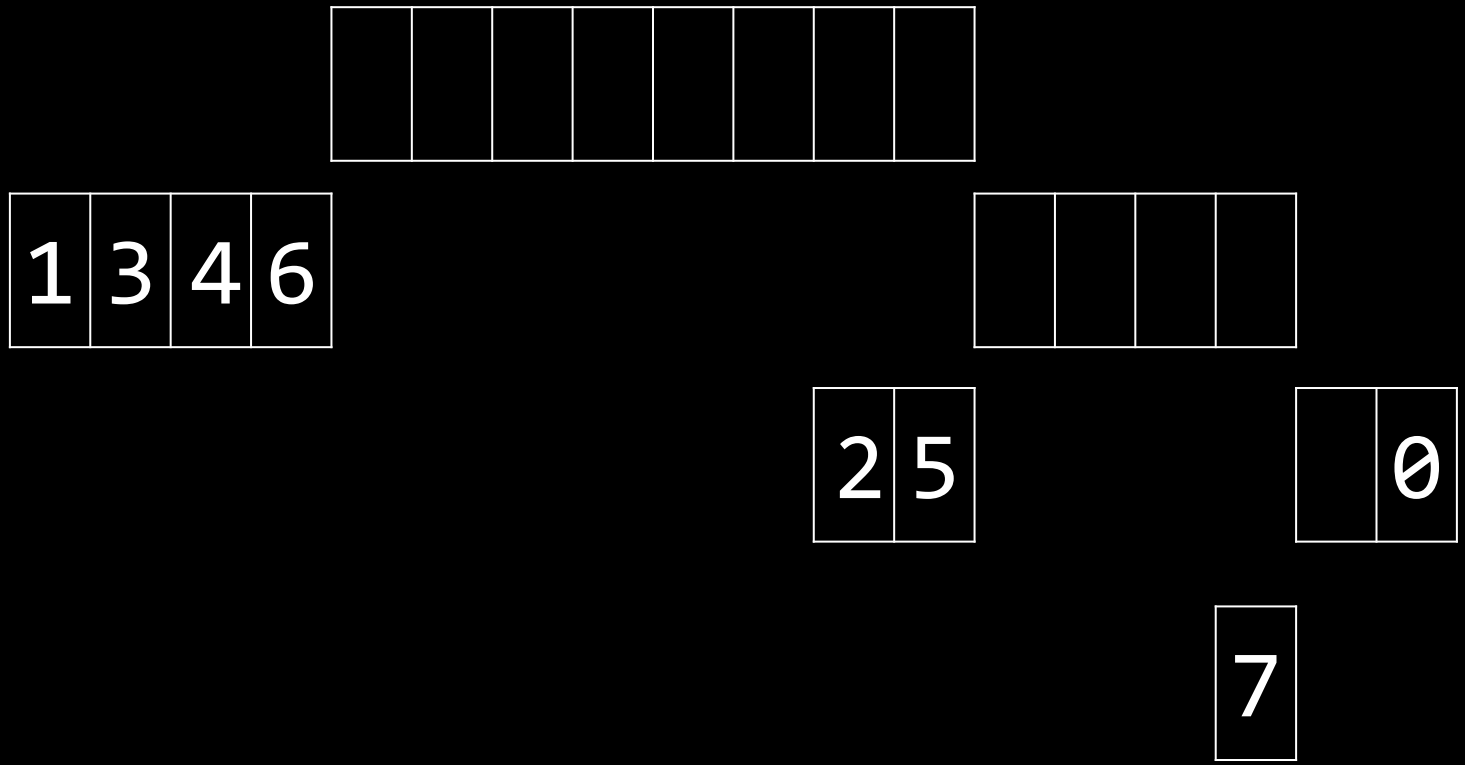

1 3 4 6
7 0
2
5

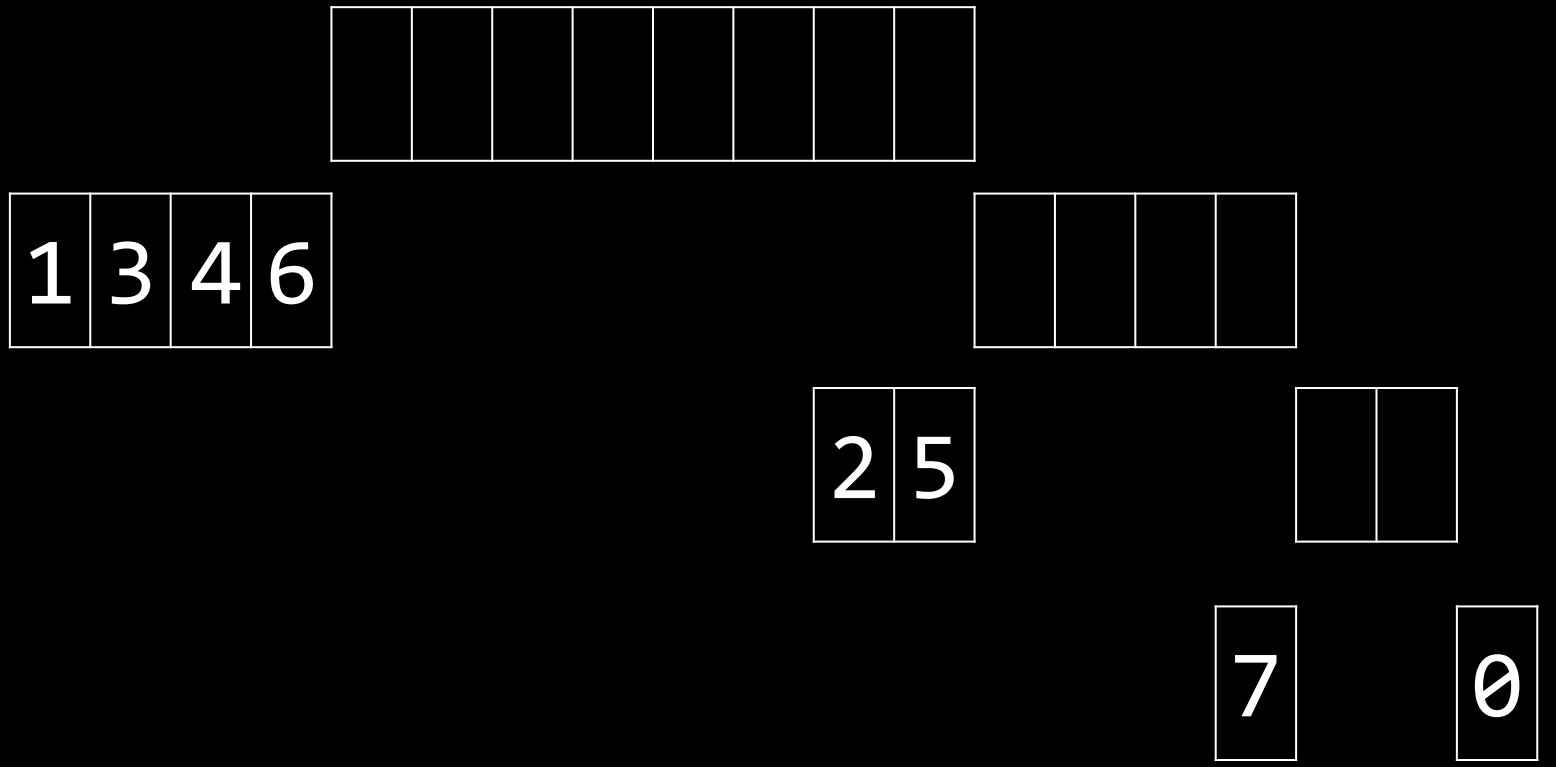

1 3 4 6
7 0
5
2

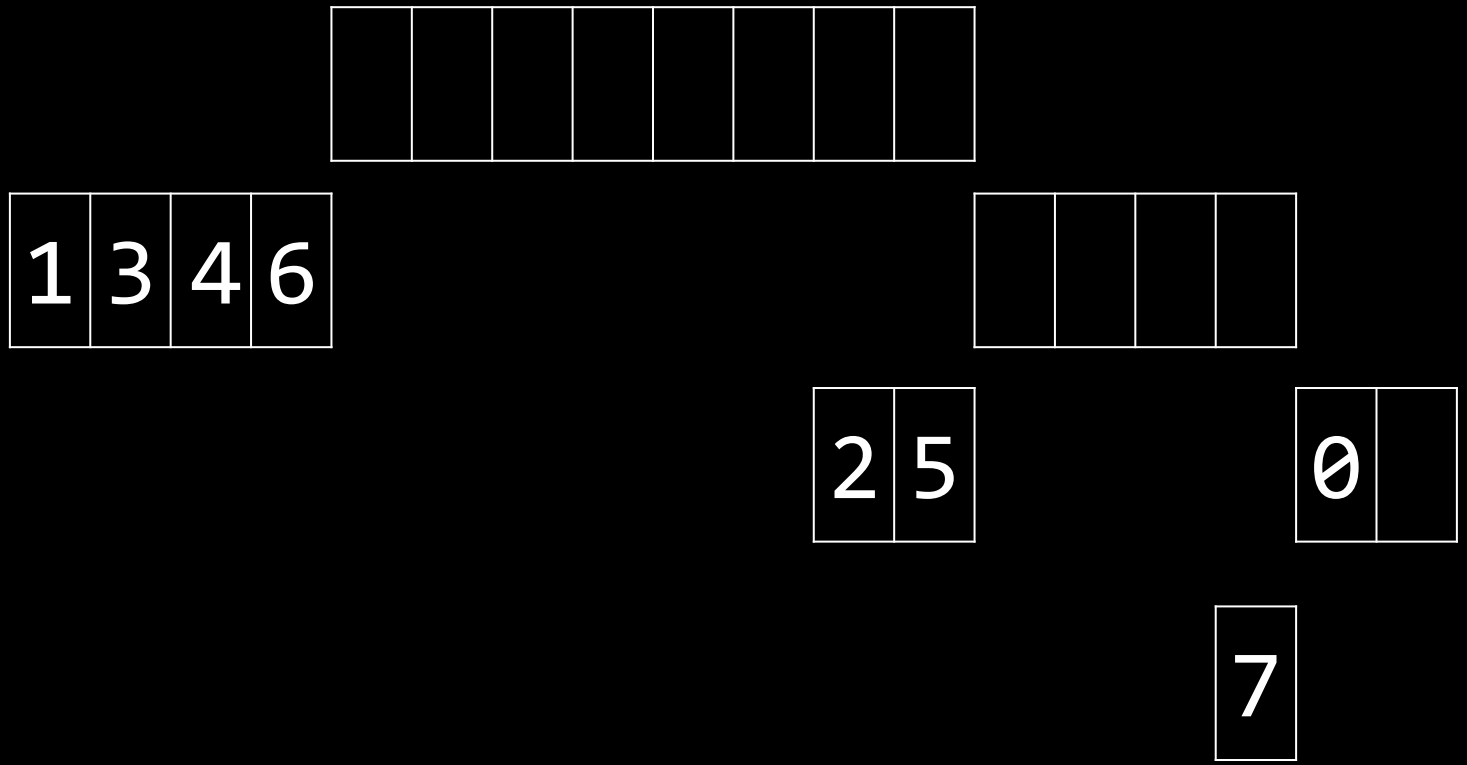

1 3 4 6
7 0
2
5

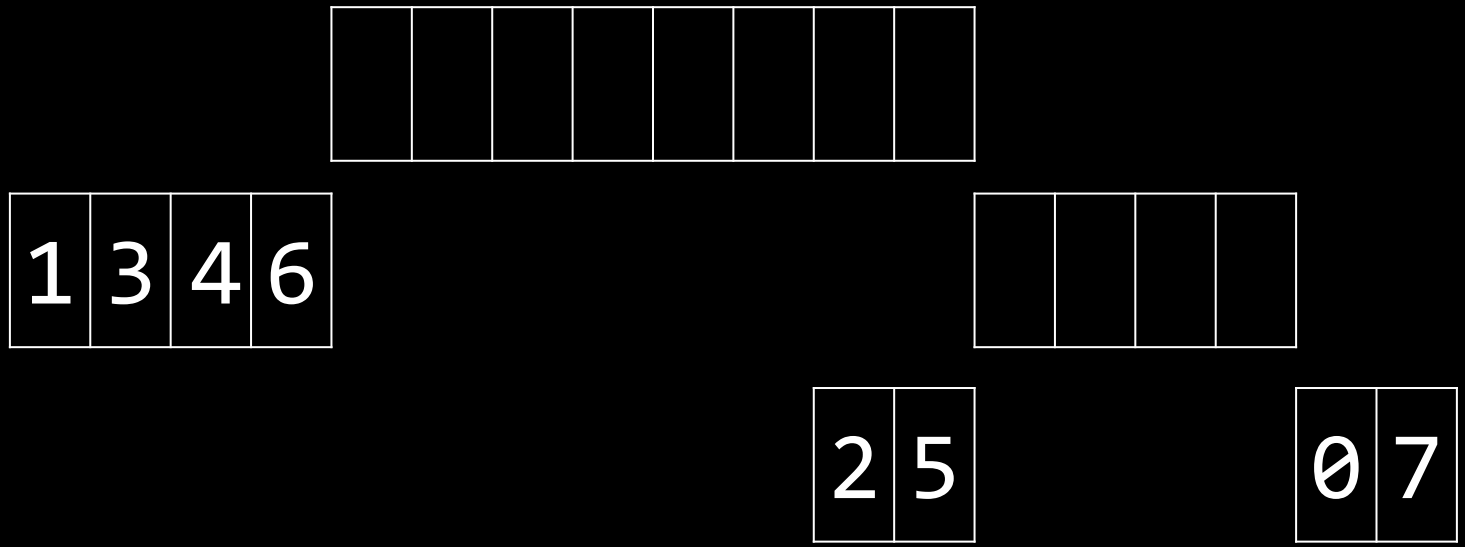

1 3 4 6
7 0
2 5

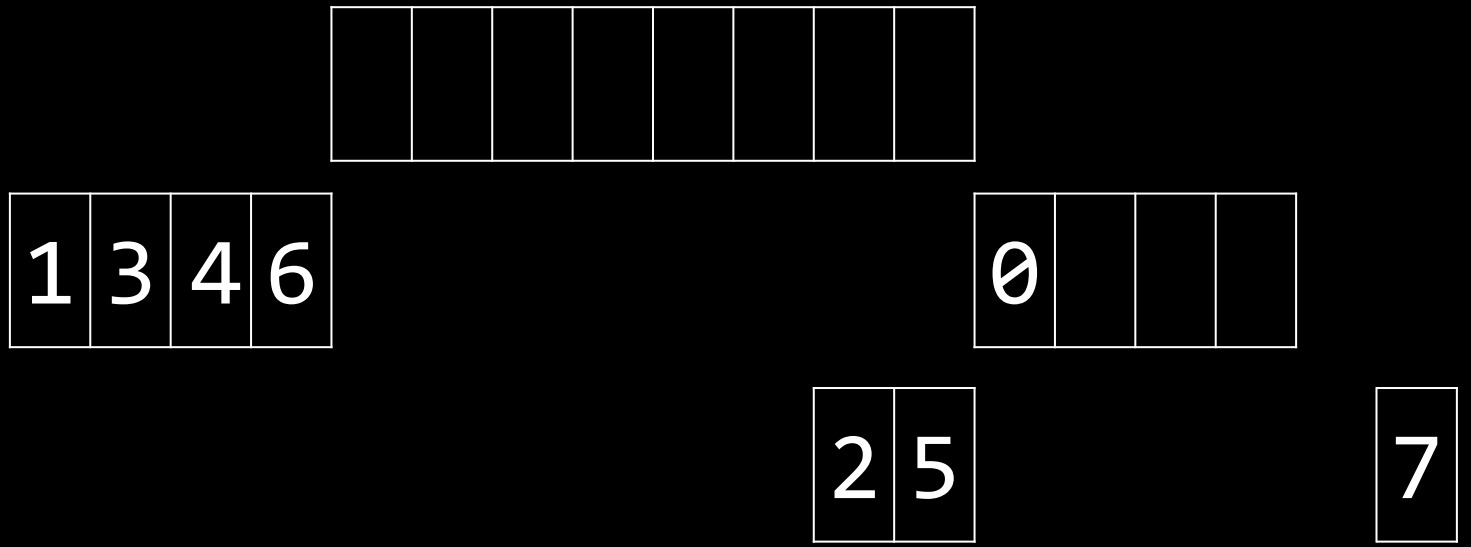

1 3 4 6
2 5
7 0

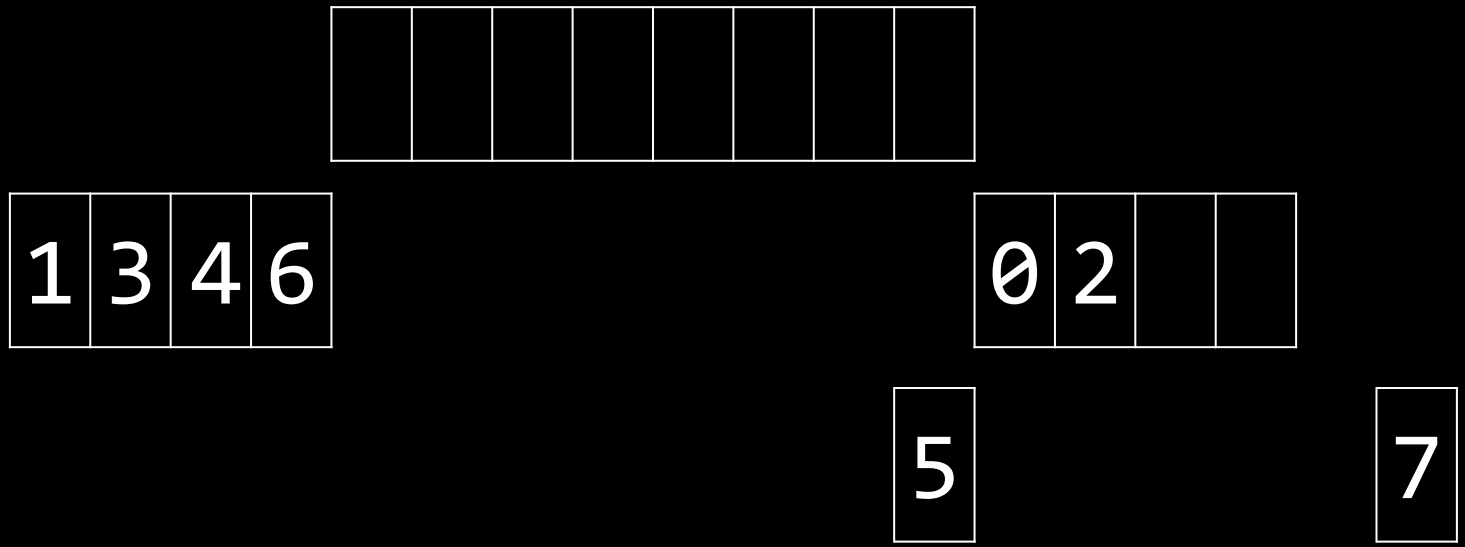

1 3 4 6
2 5
0
7

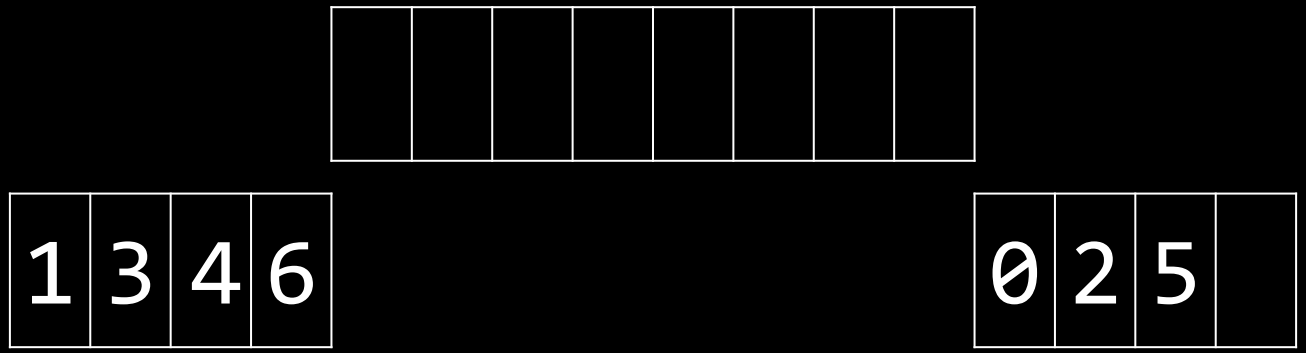

1 3 4 6
2 5
7
0

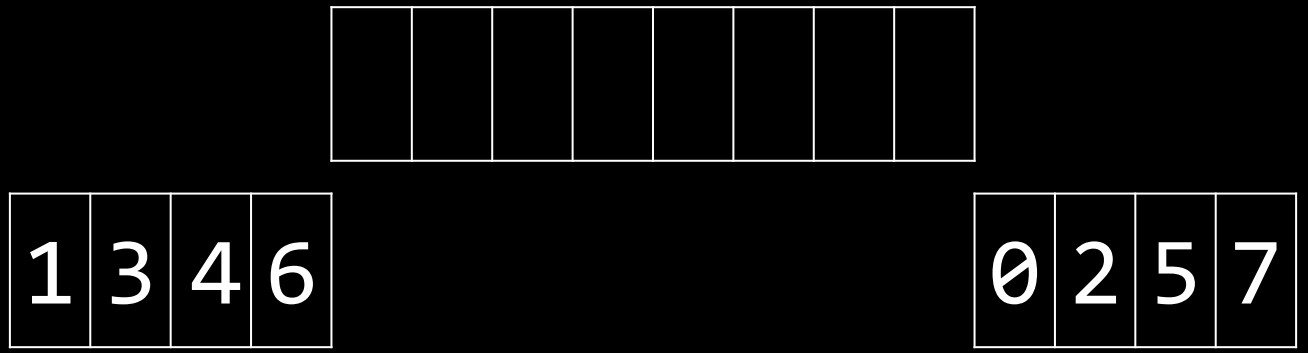

1 3 4 6
2 5
0
7

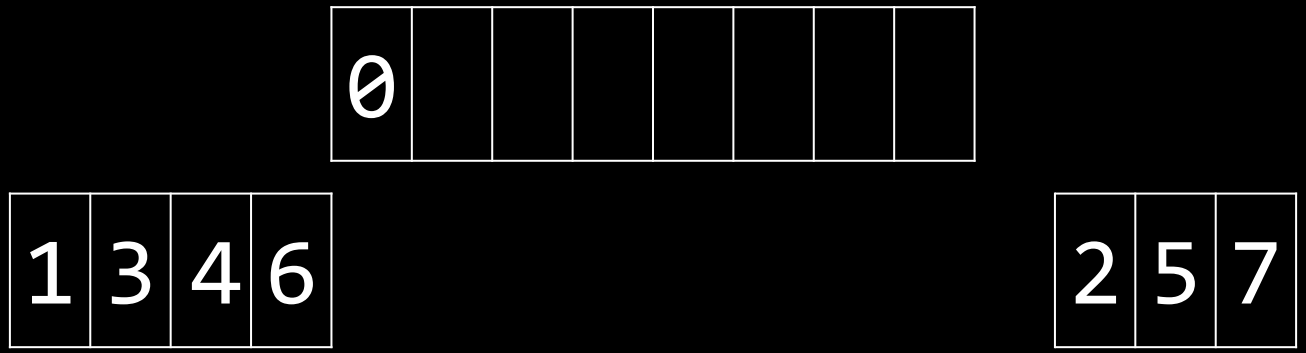

1 3 4 6
2 5
0 7

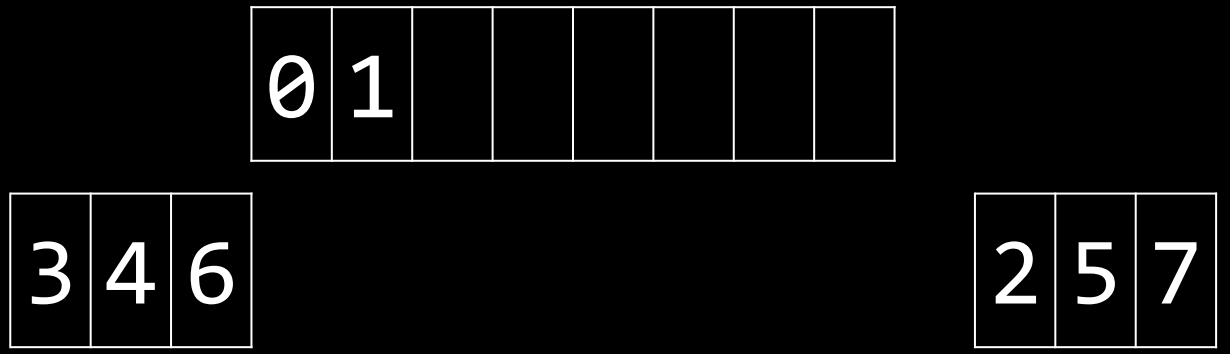

1 3 4 6
0
2 5
7

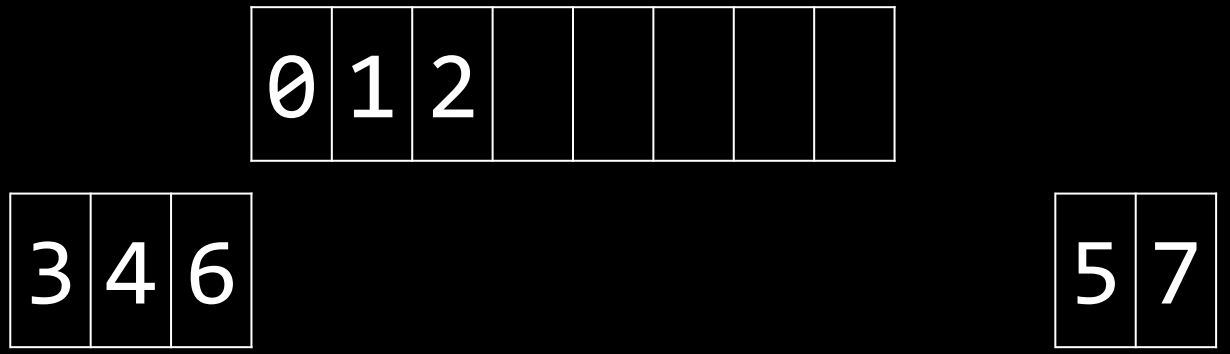

1 3 4 6
0 2
5
7

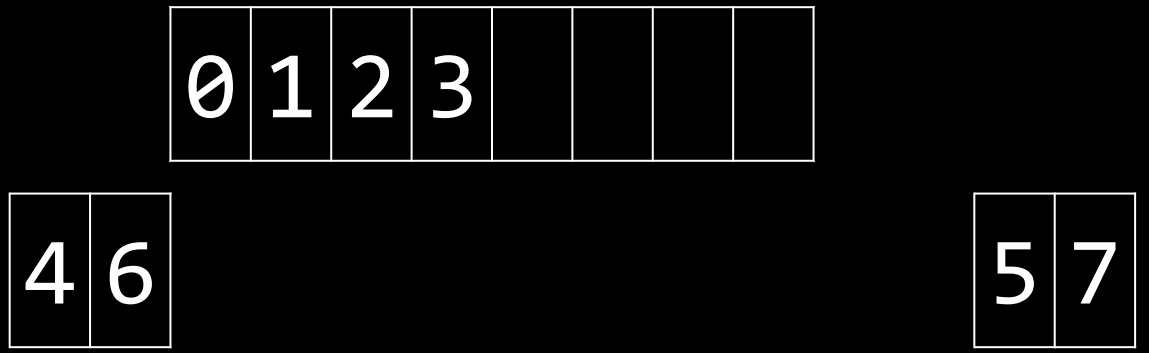

1 3 4 6
0 2 5
7

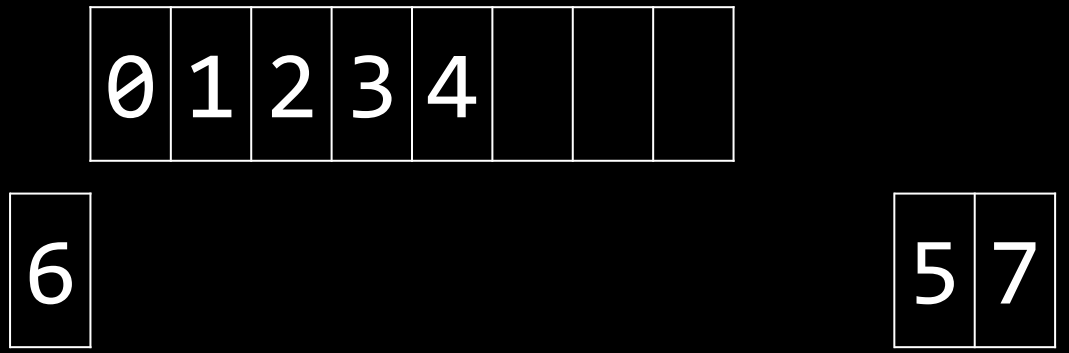

1 3 4 6
0 2 5 7

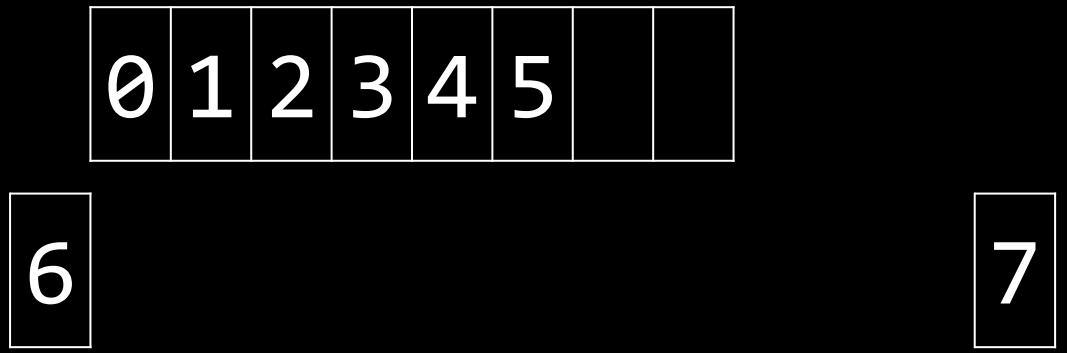

0
1 3 4 6
2 5 7


0 1
3 4 6
2 5 7


0 1 2
3 4 6
5 7


0 1 2 3
4 6
5 7


0 1 2 3 4
6
5 7


0 1 2 3 4 5
6
7

7


0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

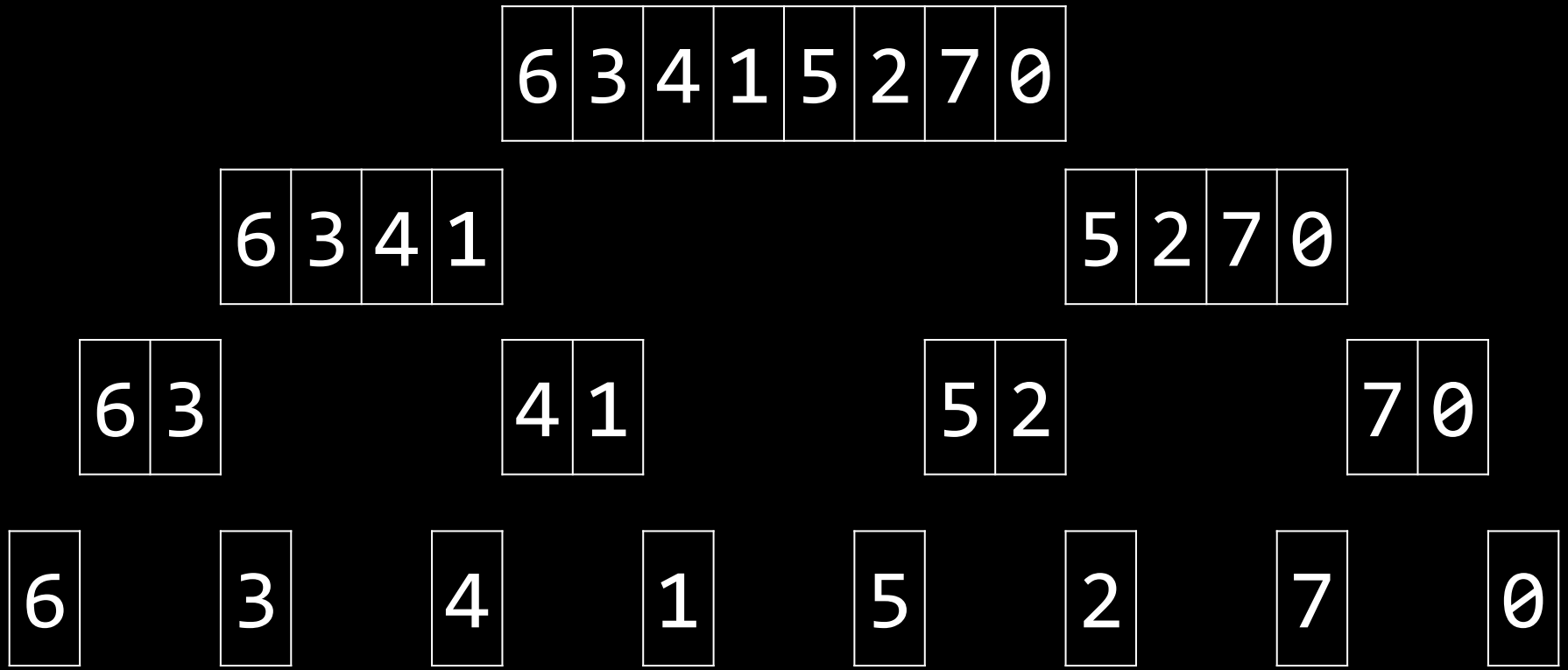
$O(n^2)$

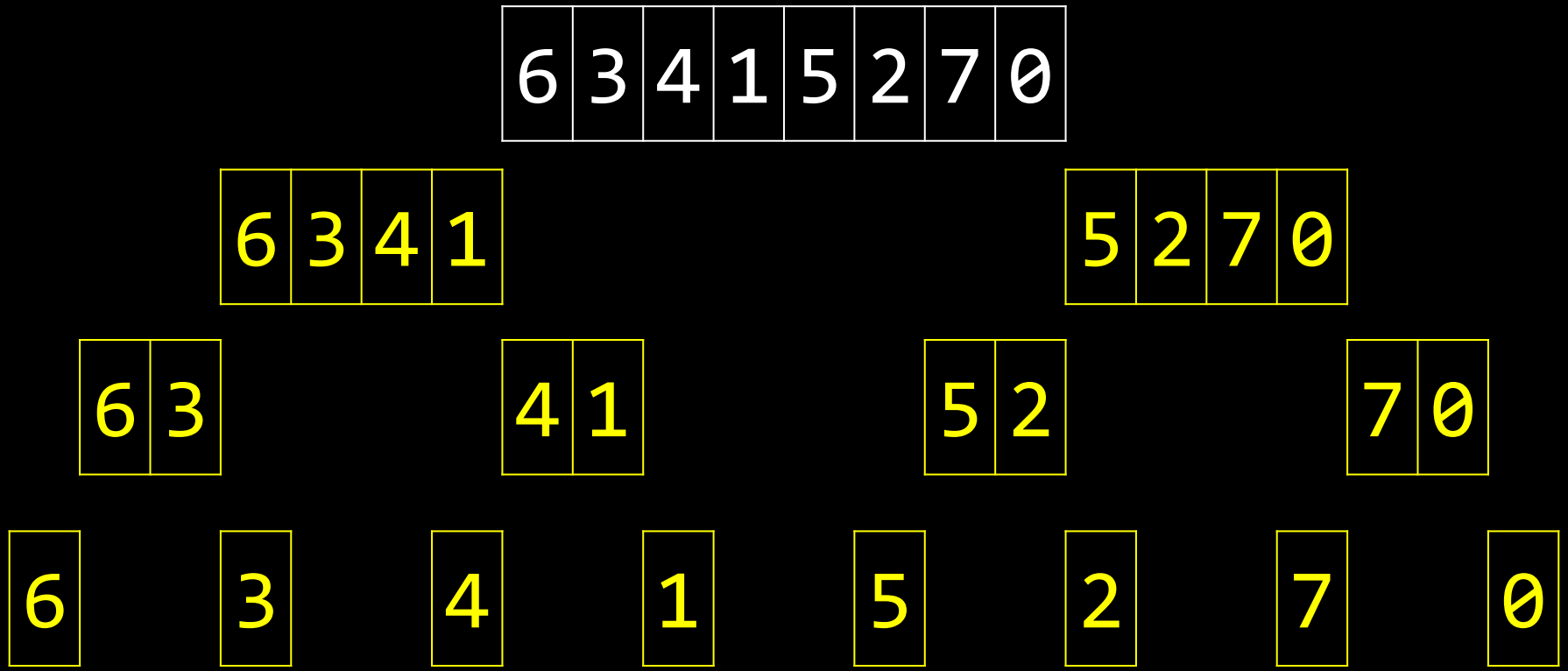
$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$


6 3 4 1 5 2 7 0
6 3 4 1
5 2 7 0
6 3
4 1
5 2
7 0
6
3
4
1
5
2
7
0

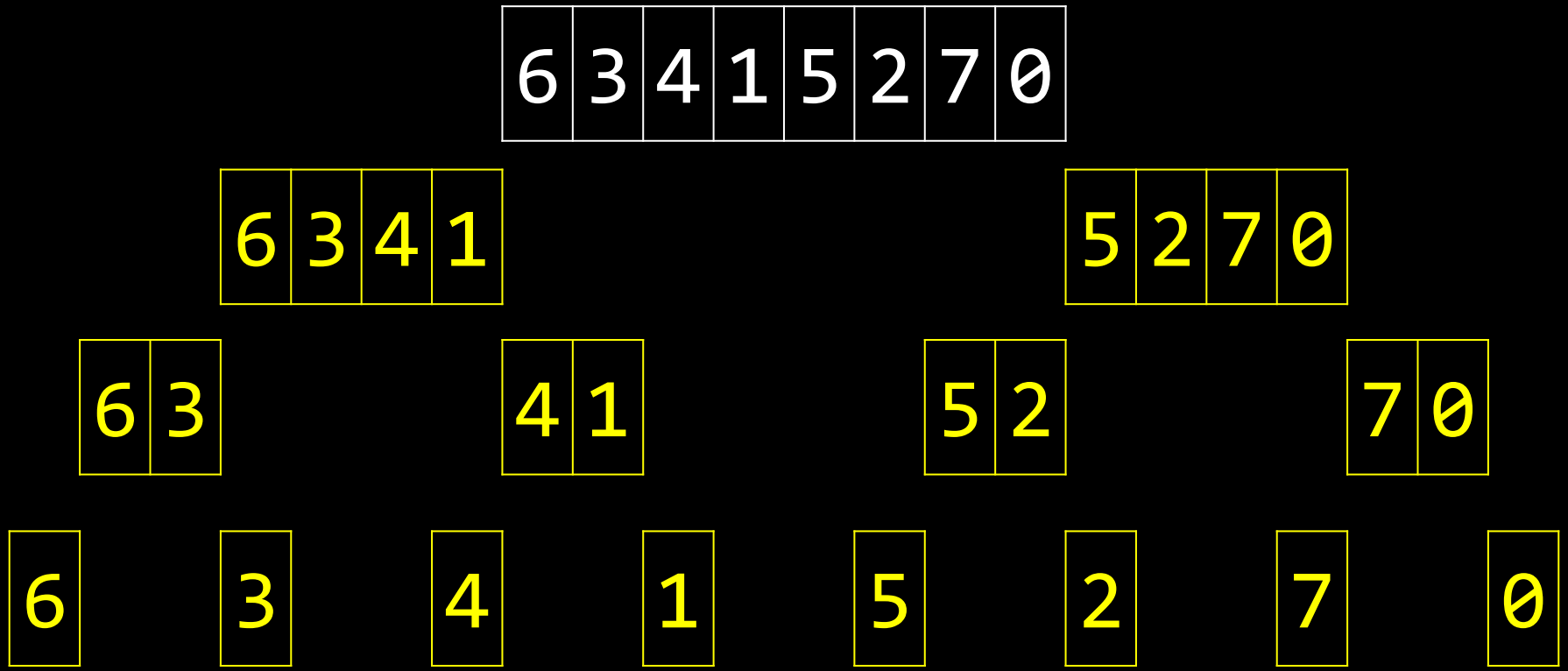

6 3 4 1 5 2 7 0
6 3 4 1
5 2 7 0
6 3
4 1
5 2
7 0
6
3
4
1
5
2
7
0

$\log_2 n$

$\log_2 8$

$\log_2 2^3$

3


6 3 4 1 5 2 7 0
6 3 4 1
5 2 7 0
6 3
4 1
5 2
7 0
6
3
4
1
5
2
7
0

$n \log_2 n$

$n \log n$

$O(n^2)$

$O(n \log n)$ Merge Sort

$O(n)$

$O(\log n)$

$O(1)$

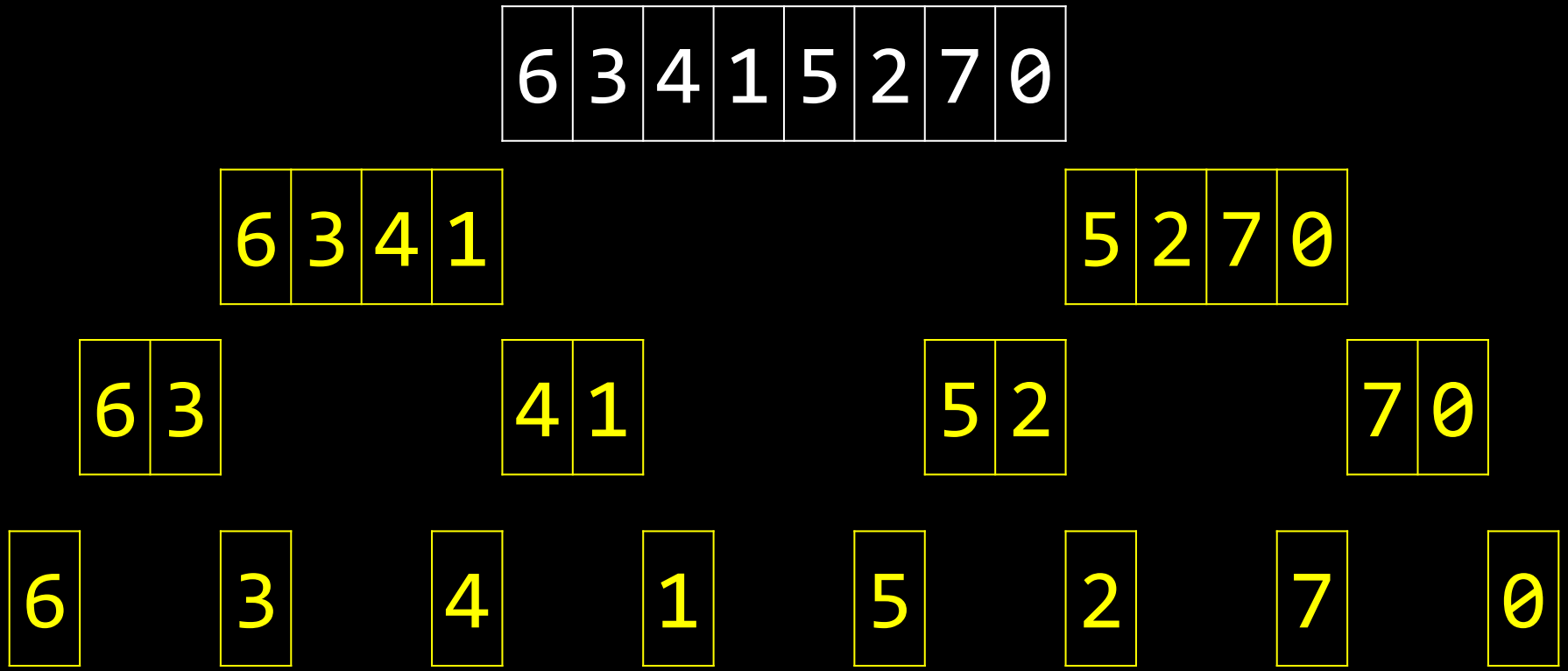
$\Omega(n^2)$

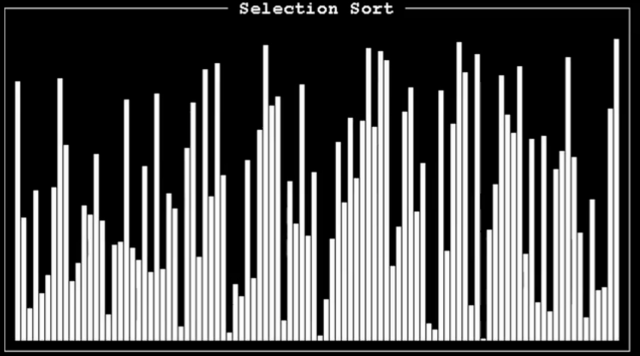
$\Omega(n \log n)$ Merge Sort

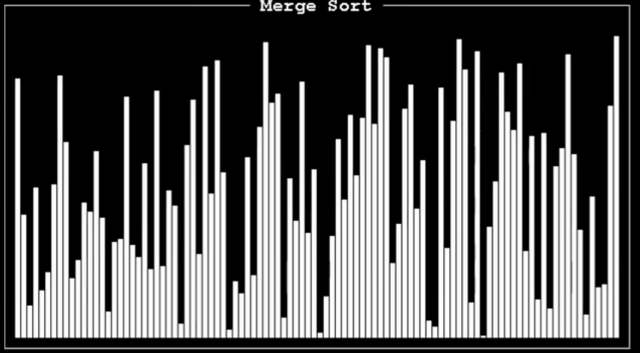
$\Omega(n)$

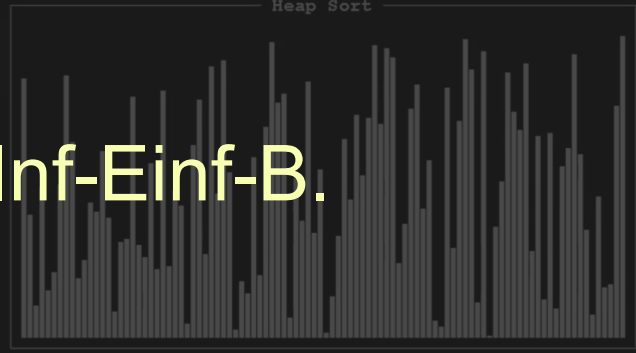
$\Omega(\log n)$

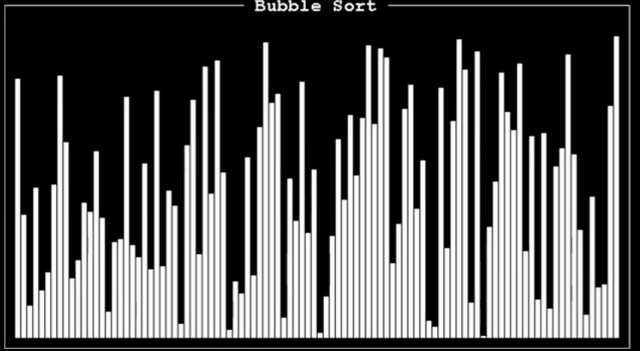
$\Omega(1)$

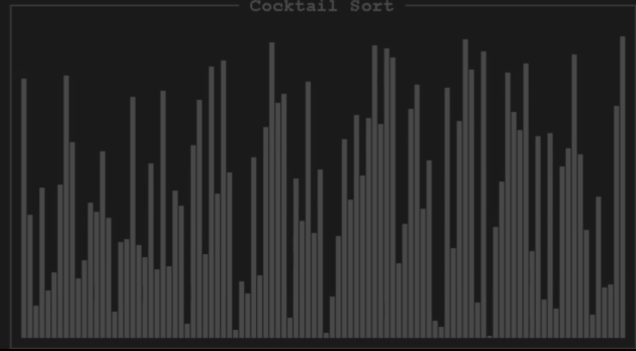

6 3 4 1 5 2 7 0
6 3 4 1
5 2 7 0
6 3
4 1
5 2
7 0
6
3
4
1
5
2
7
0



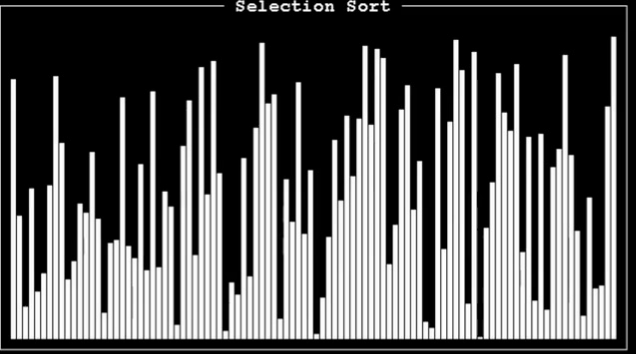


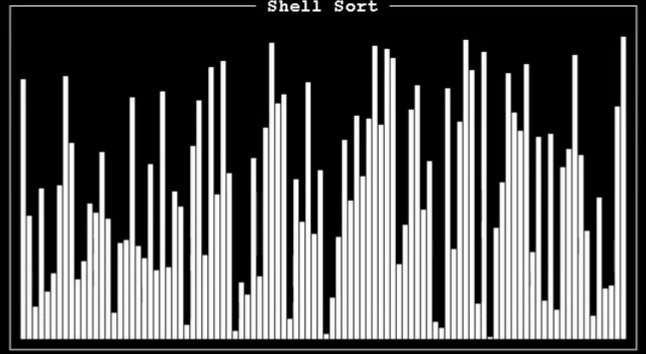




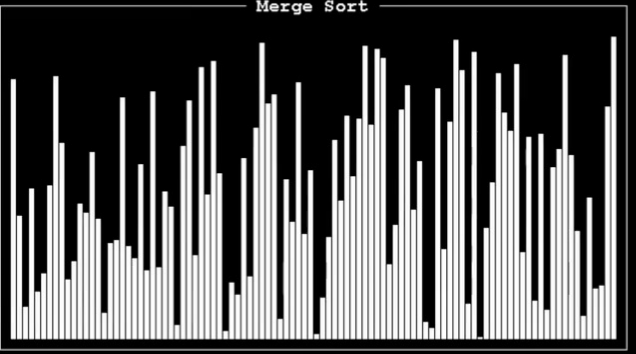


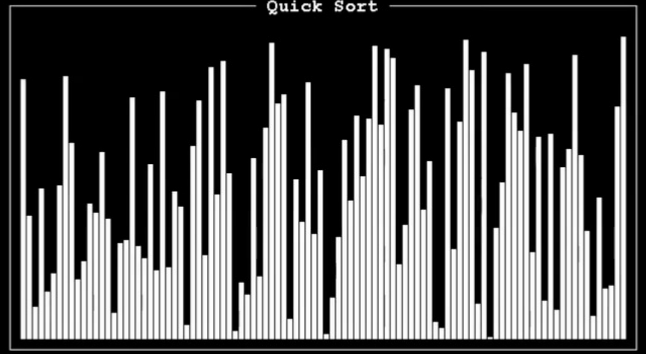
Dies war Inf-Einf-B.

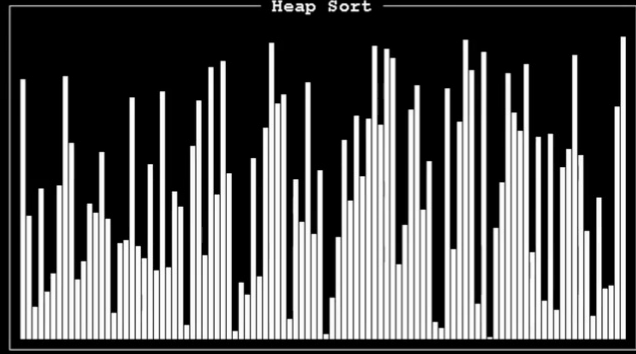

Selection Sort

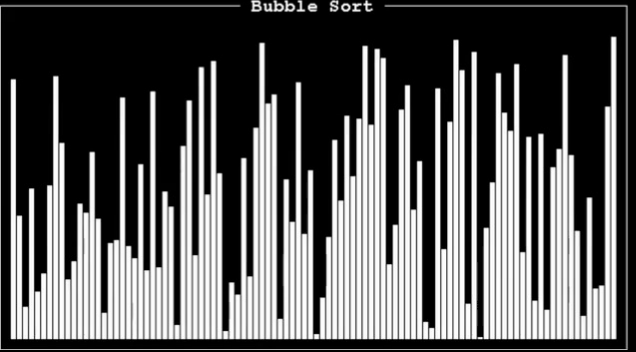

Shell Sort

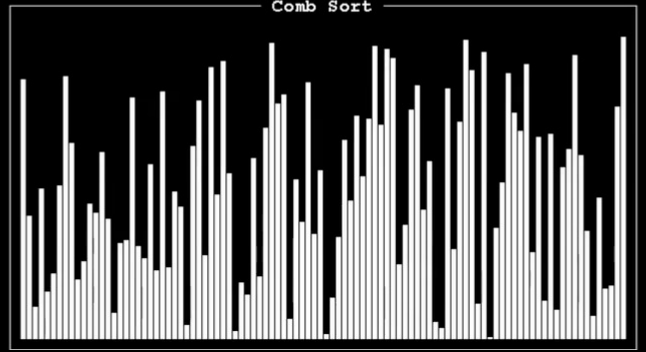

Insertion Sort

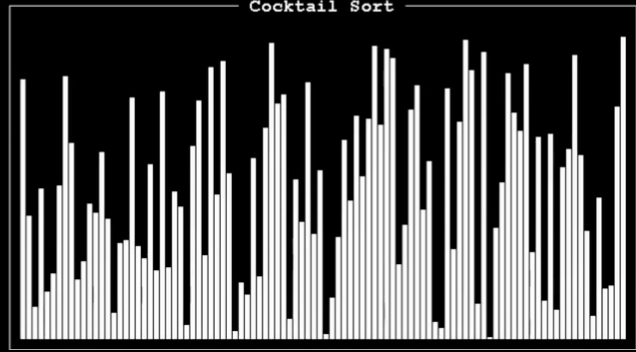

Merge Sort


Quick Sort


Heap Sort


Bubble Sort


Comb Sort


Cocktail Sort

Dies war Inf-Einf-B.