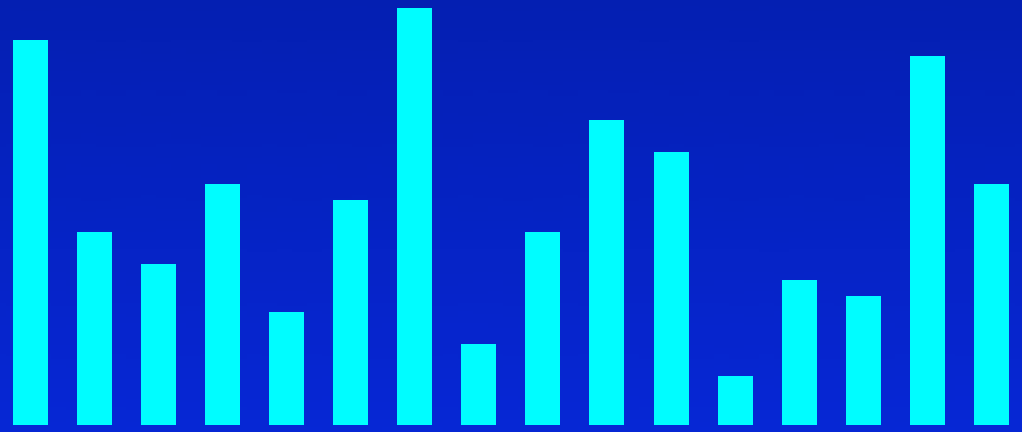
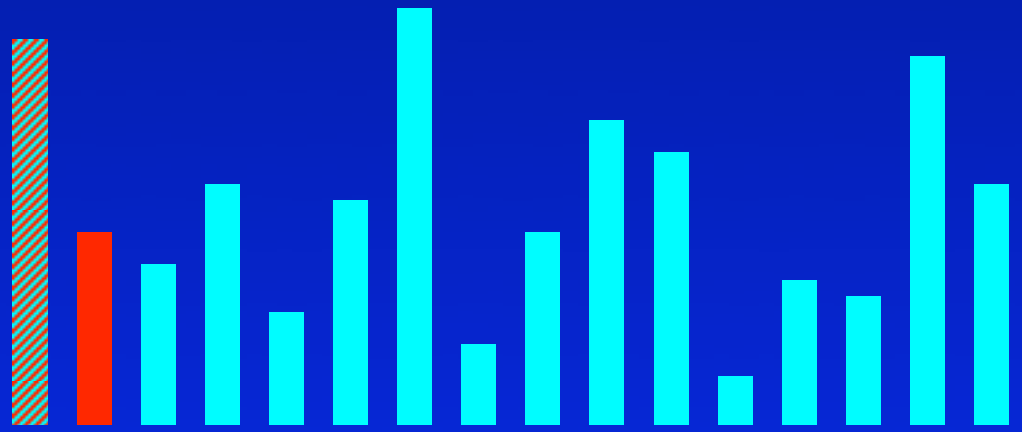
Insertion-Sort



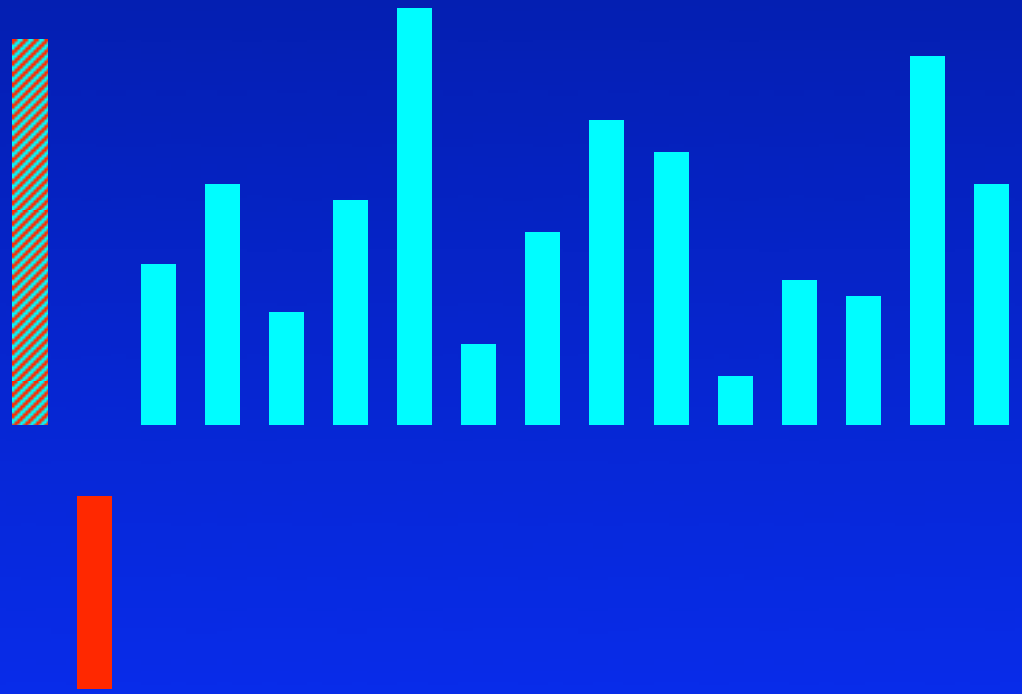
M. Esponda

Insertion-Sort



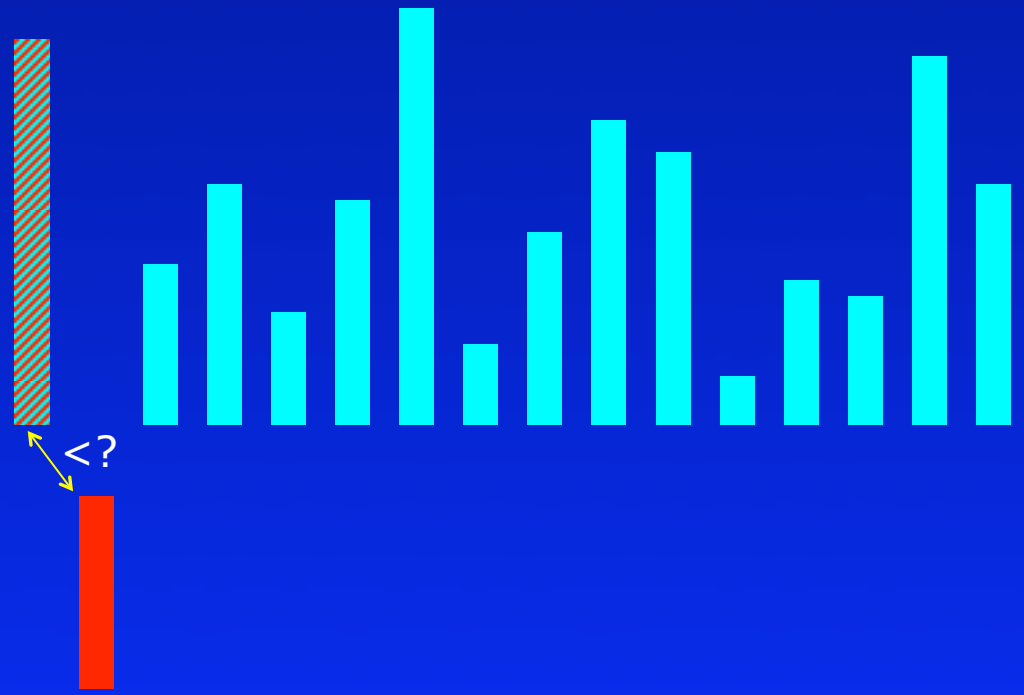
M. Esponda

Insertion-Sort



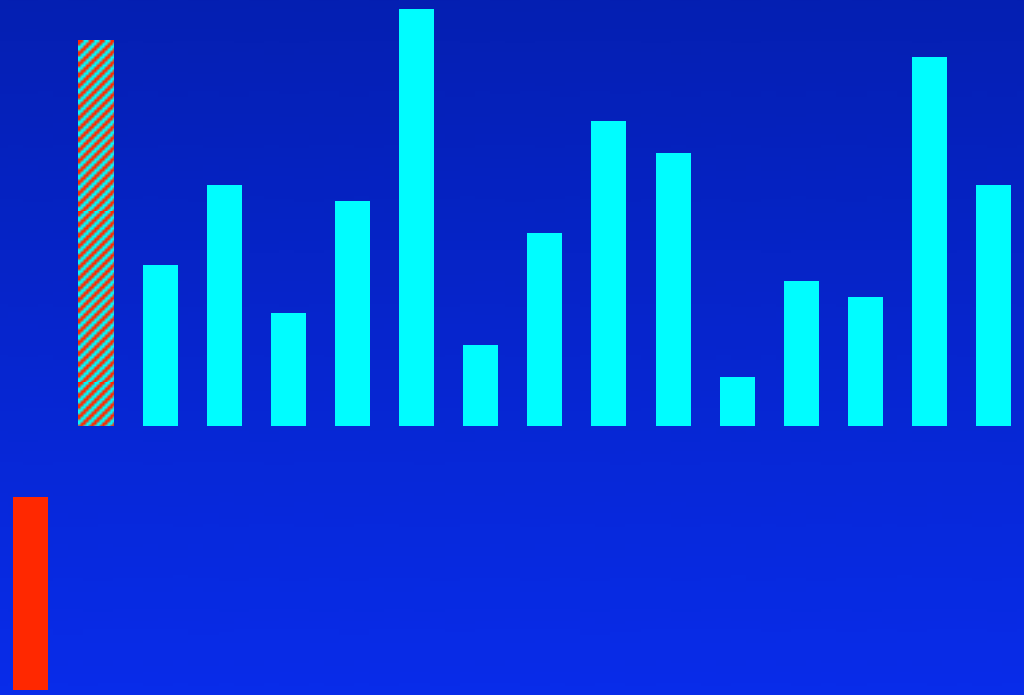
M. Esponda

Insertion-Sort



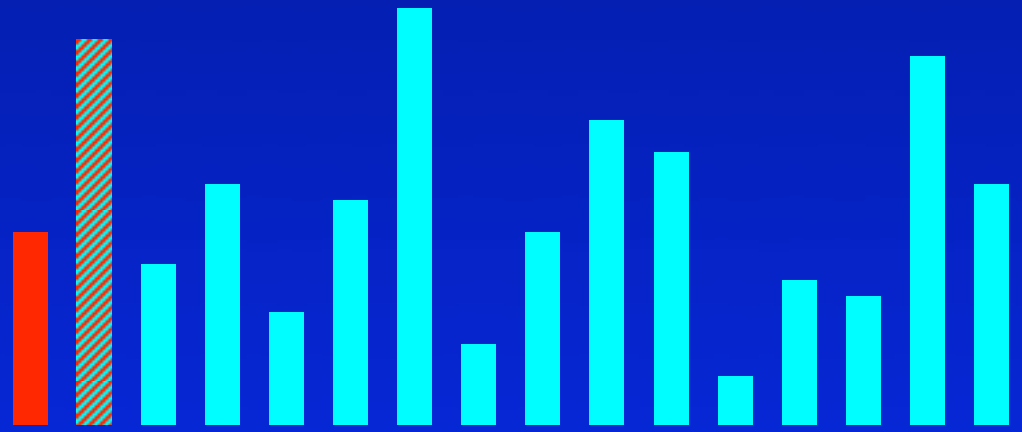
M. Esponda

Insertion-Sort



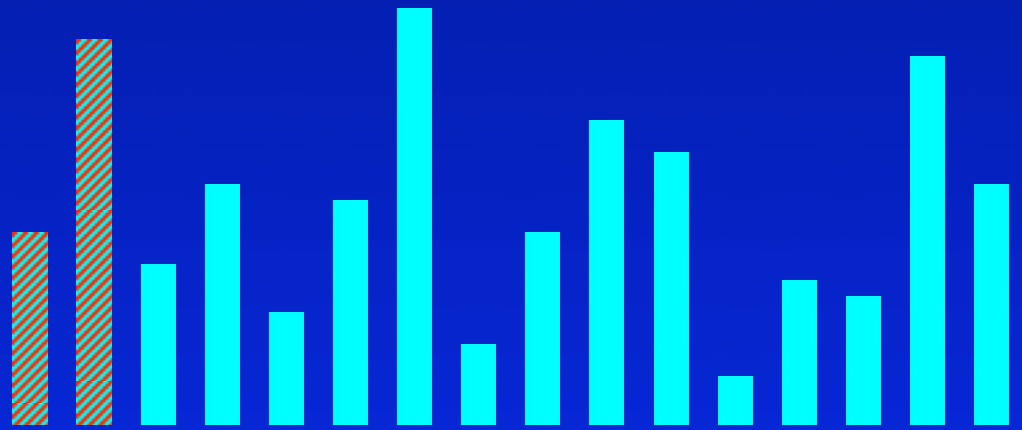
M. Esponda

Insertion-Sort



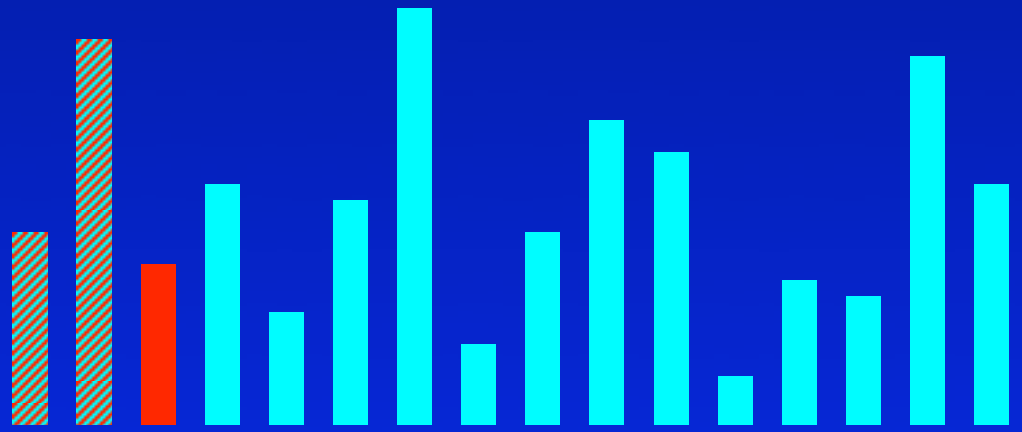
M. Esponda

Insertion-Sort



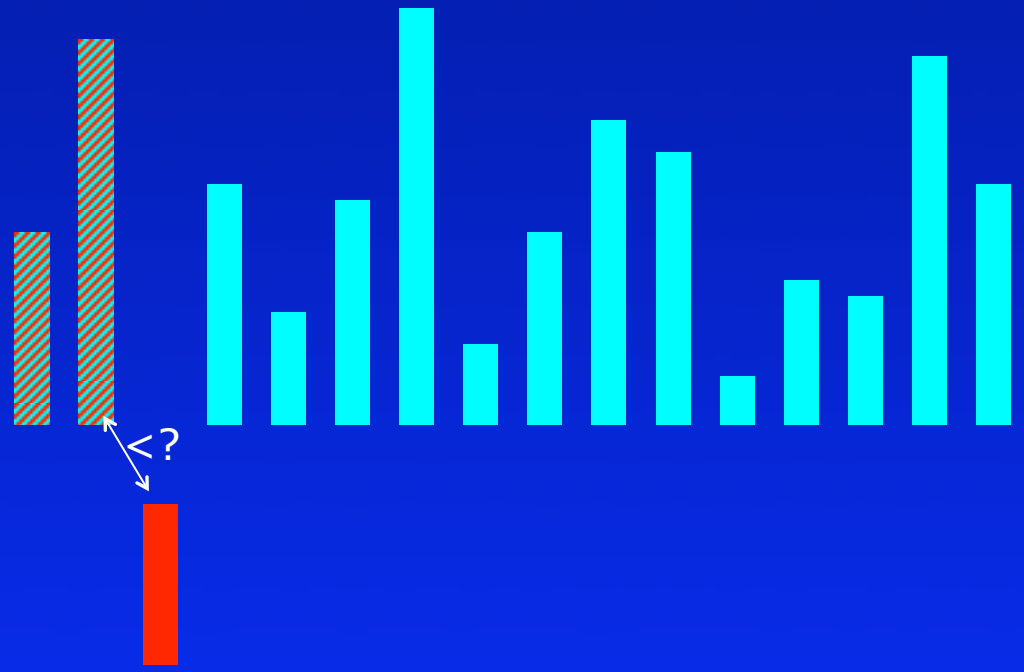
M. Esponda

Insertion-Sort



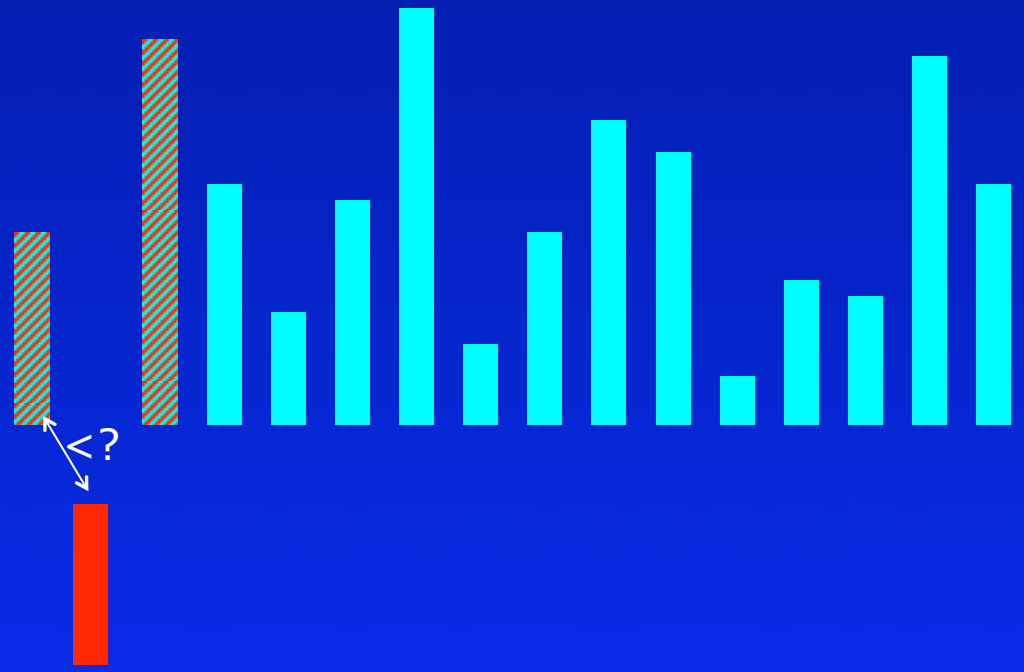
M. Esponda

Insertion-Sort



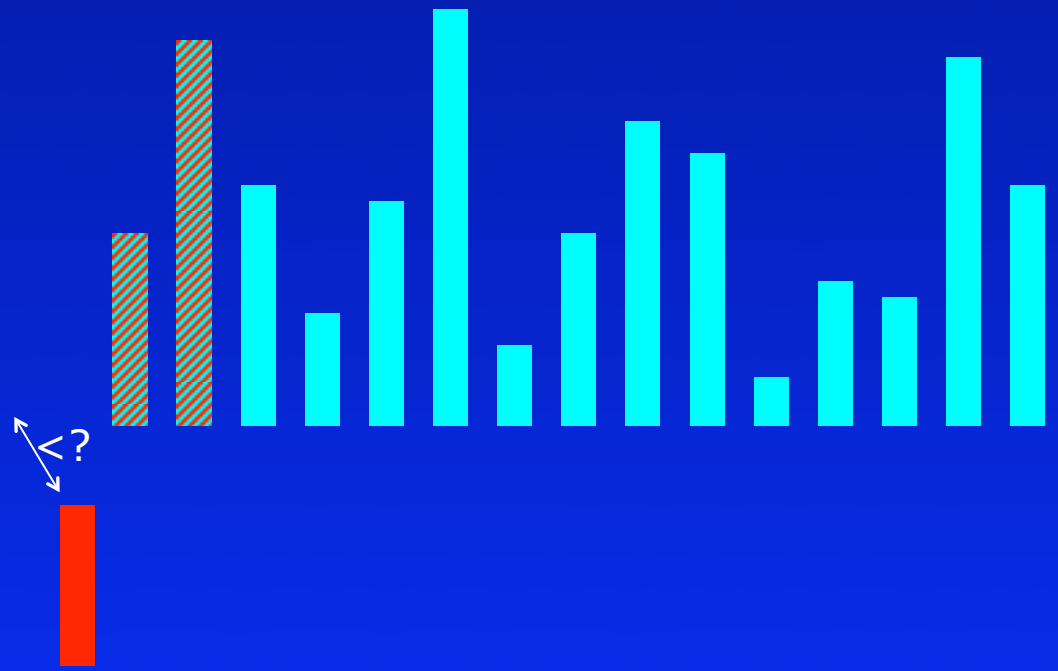
M. Esponda

Insertion-Sort



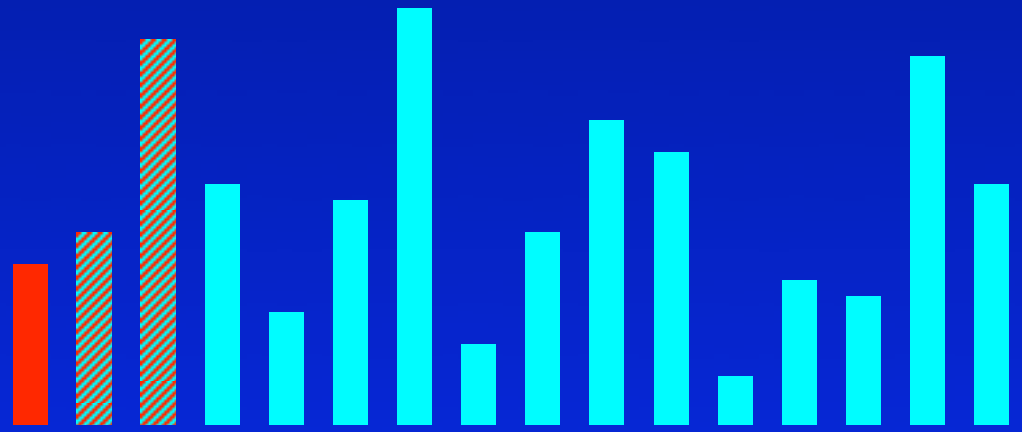
M. Esponda

Insertion-Sort



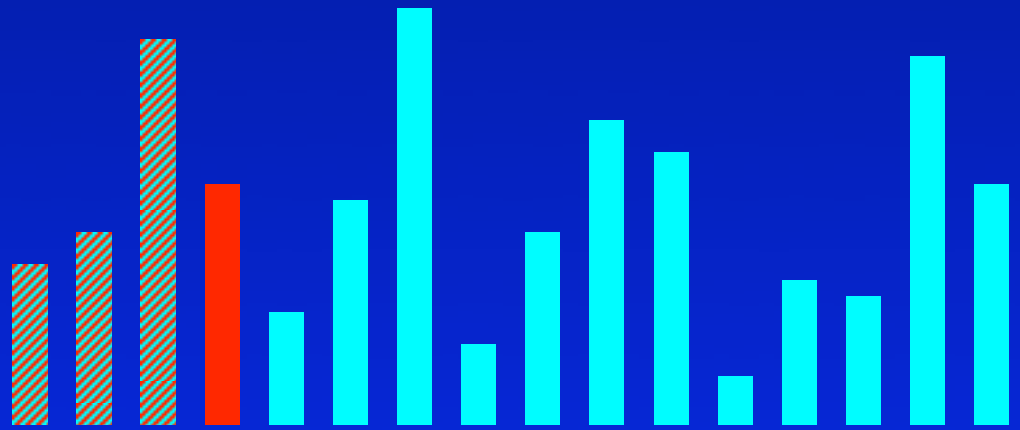
M. Esponda

Insertion-Sort



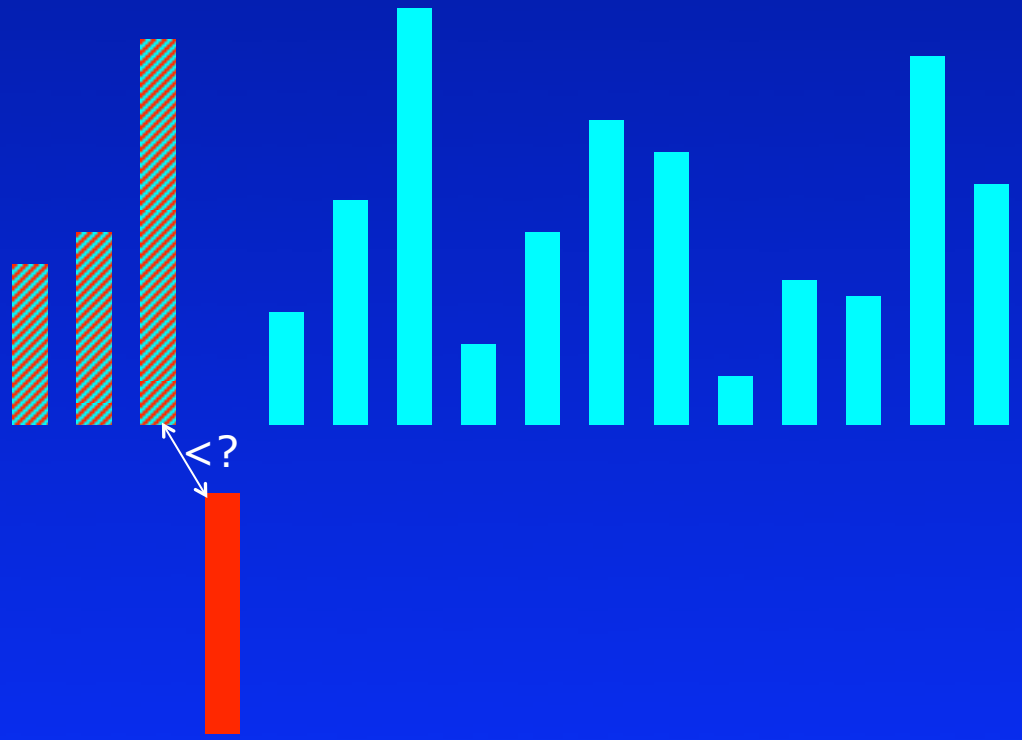
M. Esponda

Insertion-Sort



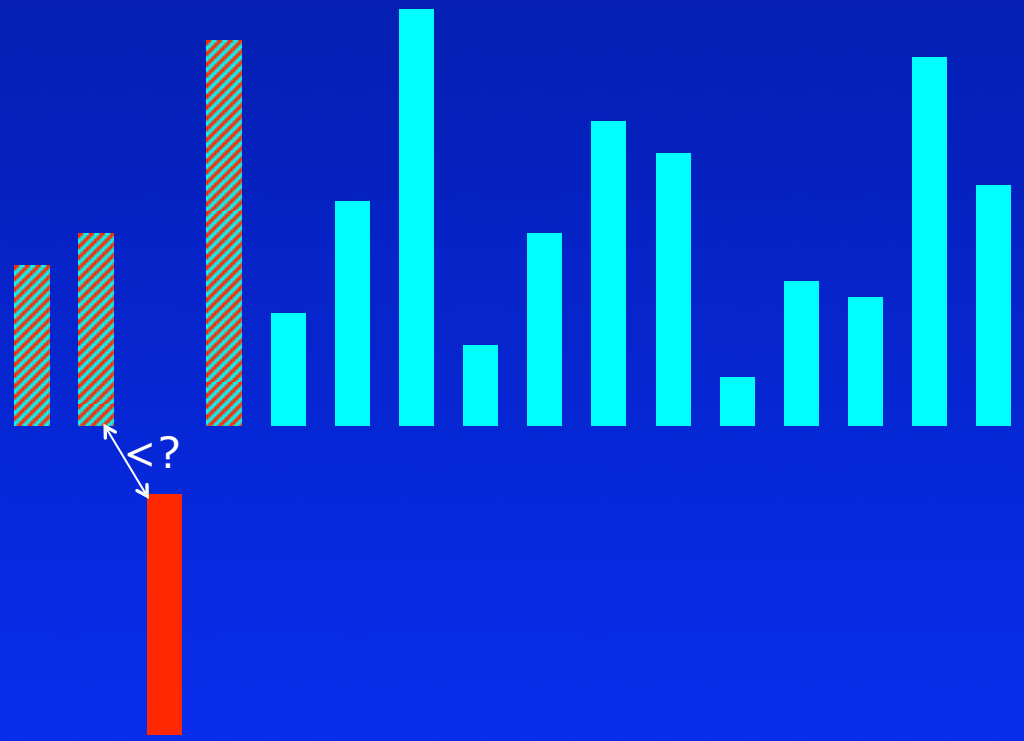
M. Esponda

Insertion-Sort



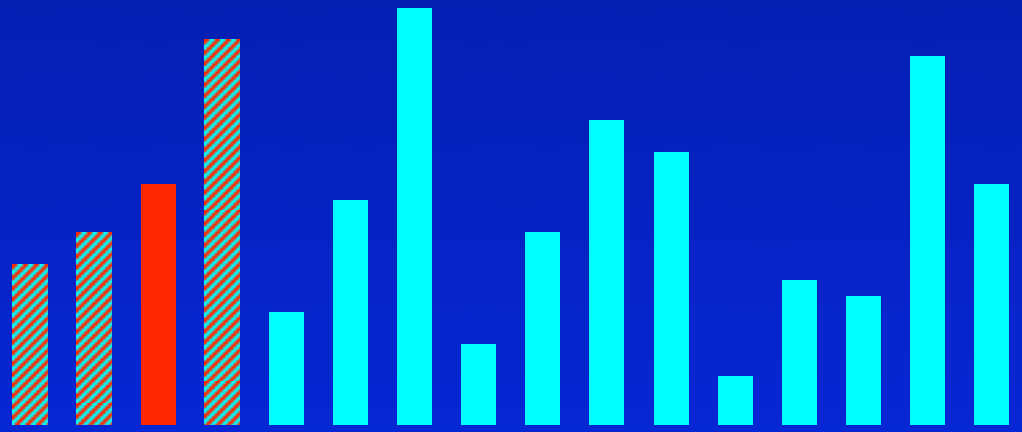
M. Esponda

Insertion-Sort



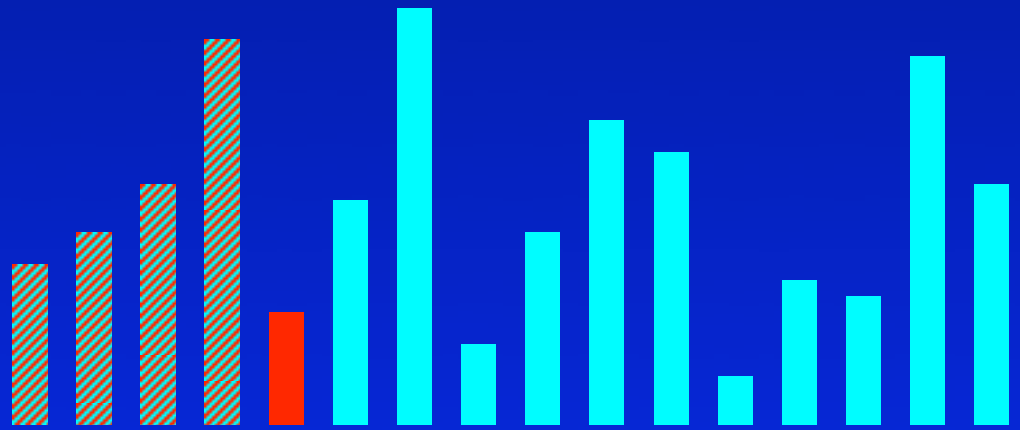
M. Esponda

Insertion-Sort



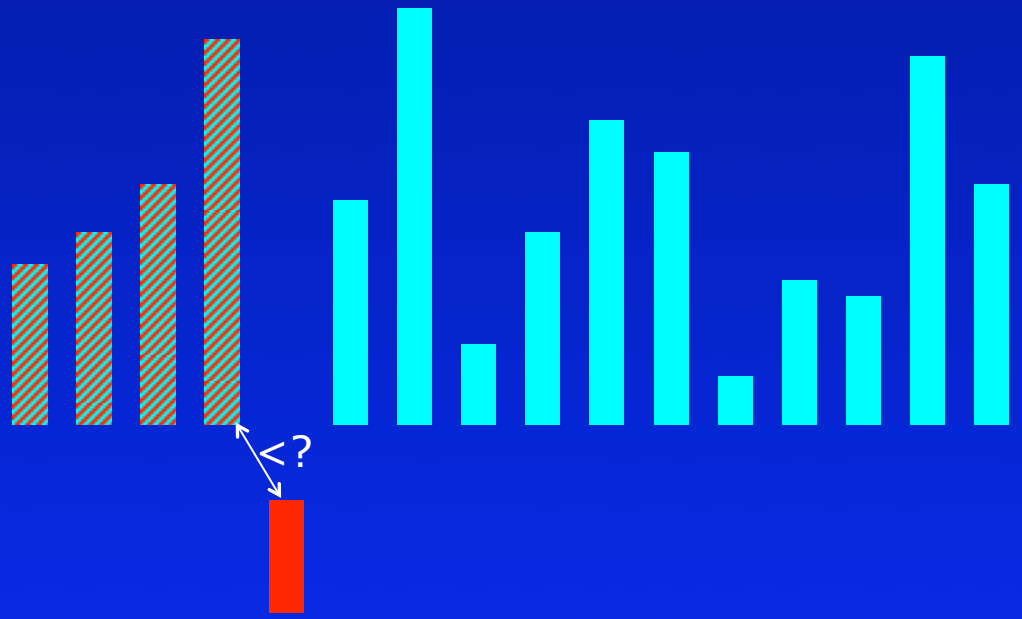
M. Esponda

Insertion-Sort



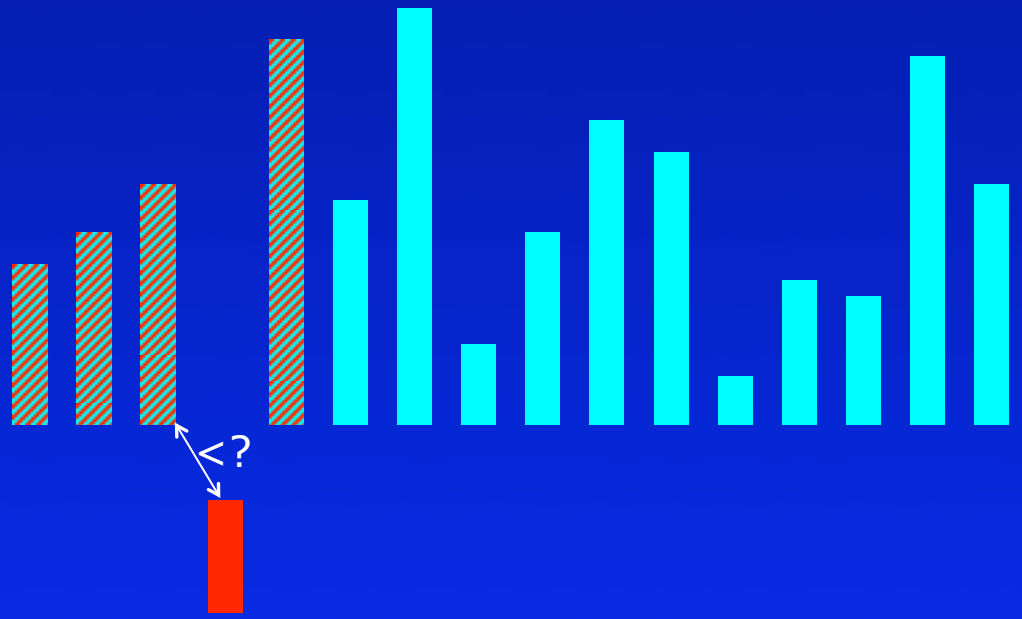
M. Esponda

Insertion-Sort



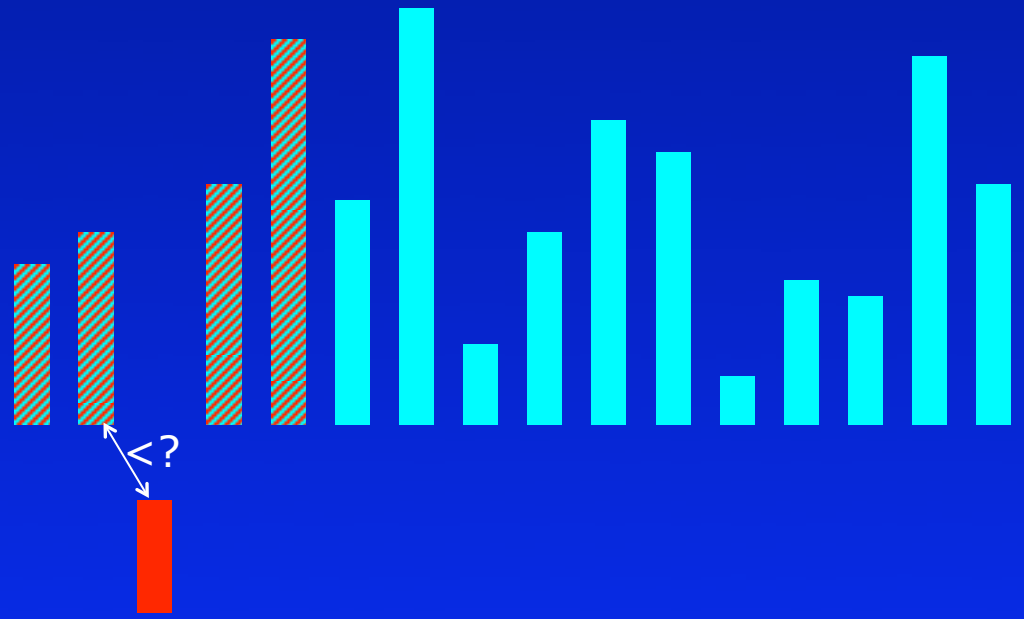
M. Esponda

Insertion-Sort



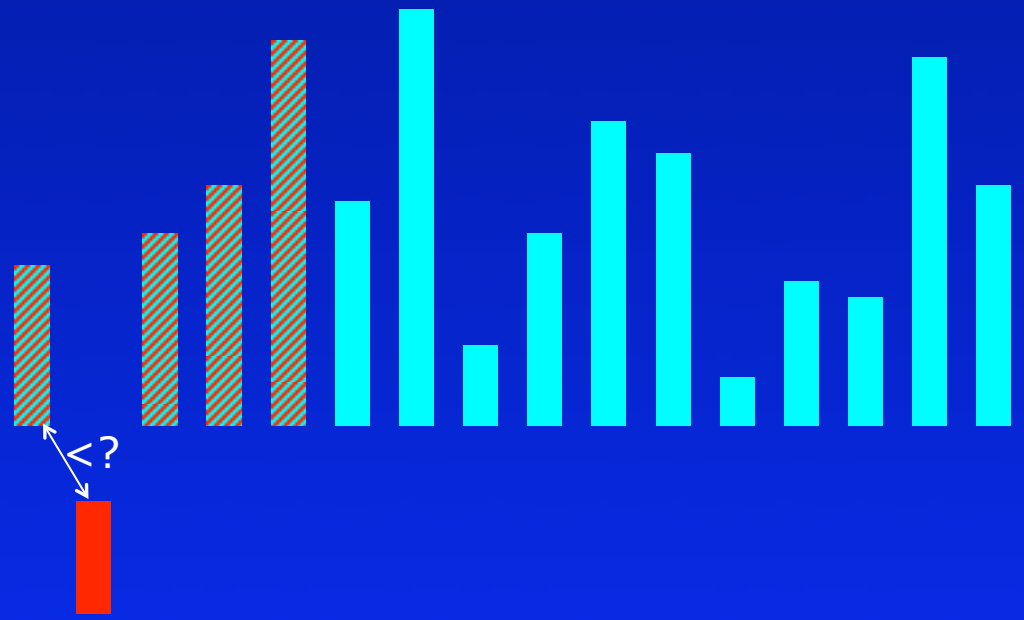
M. Esponda

Insertion-Sort



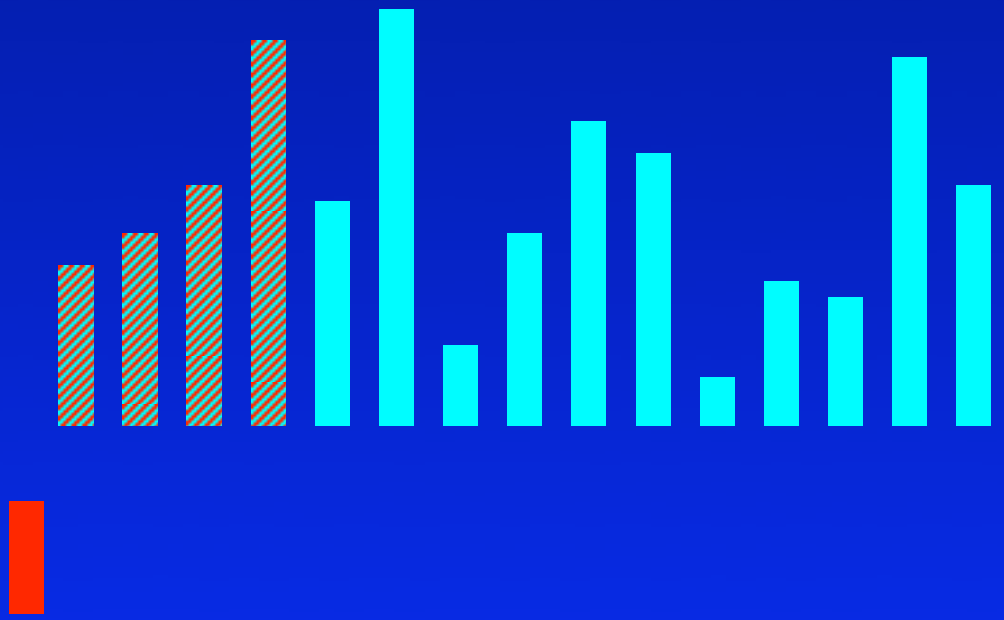
M. Esponda

Insertion-Sort



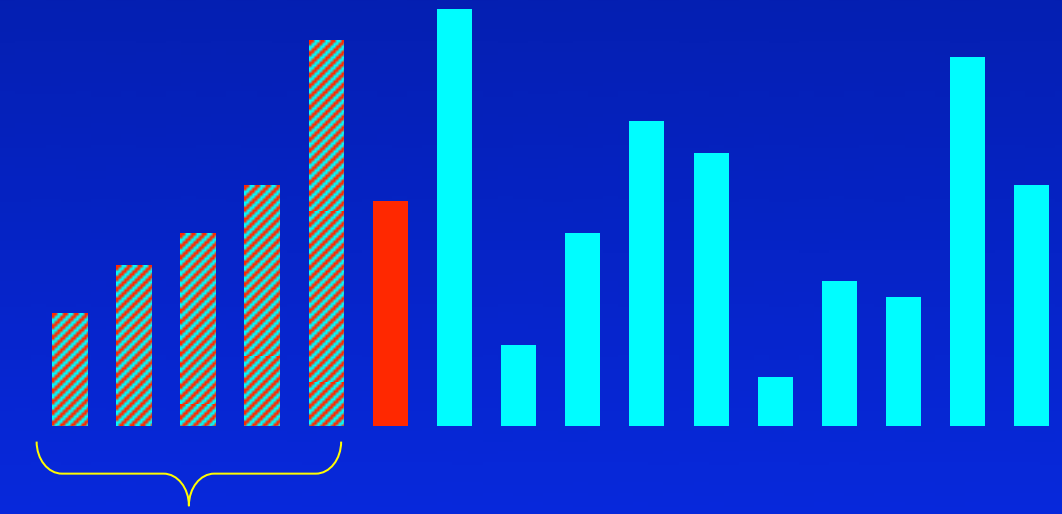
M. Esponda

Insertion-Sort



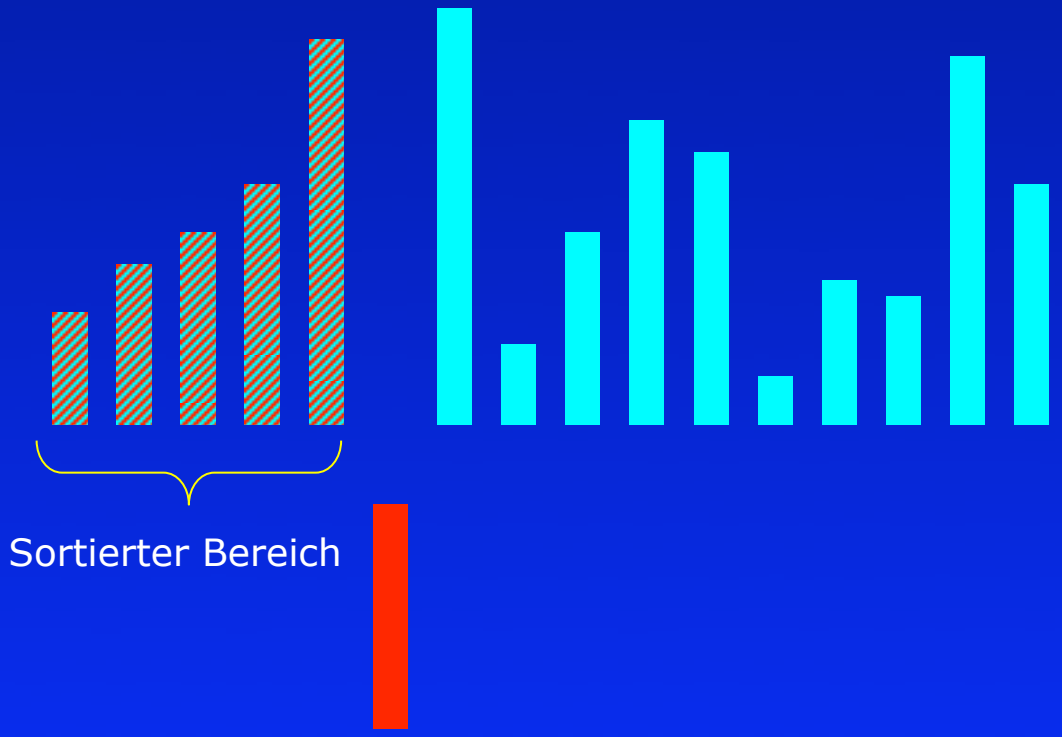
M. Esponda

Insertion-Sort



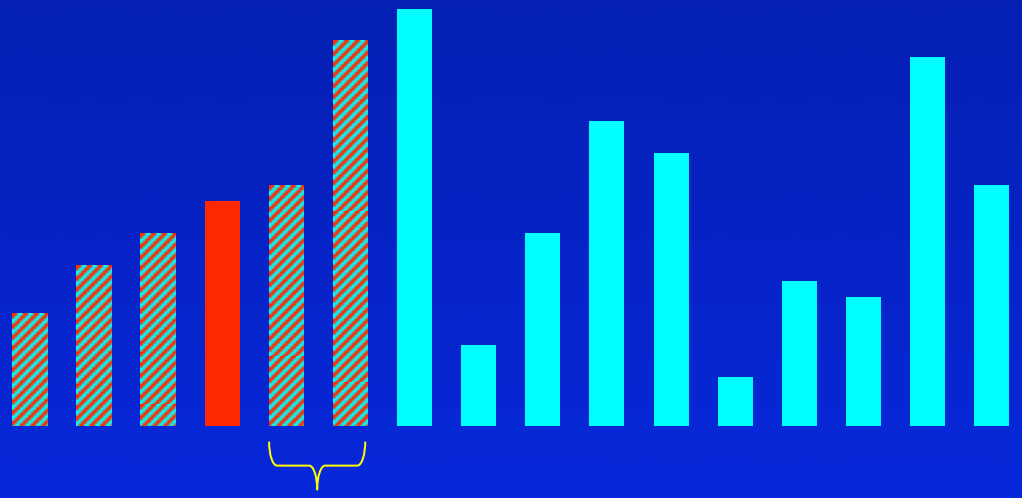
M. Esponda

Insertion-Sort



M. Esponda

Insertion-Sort

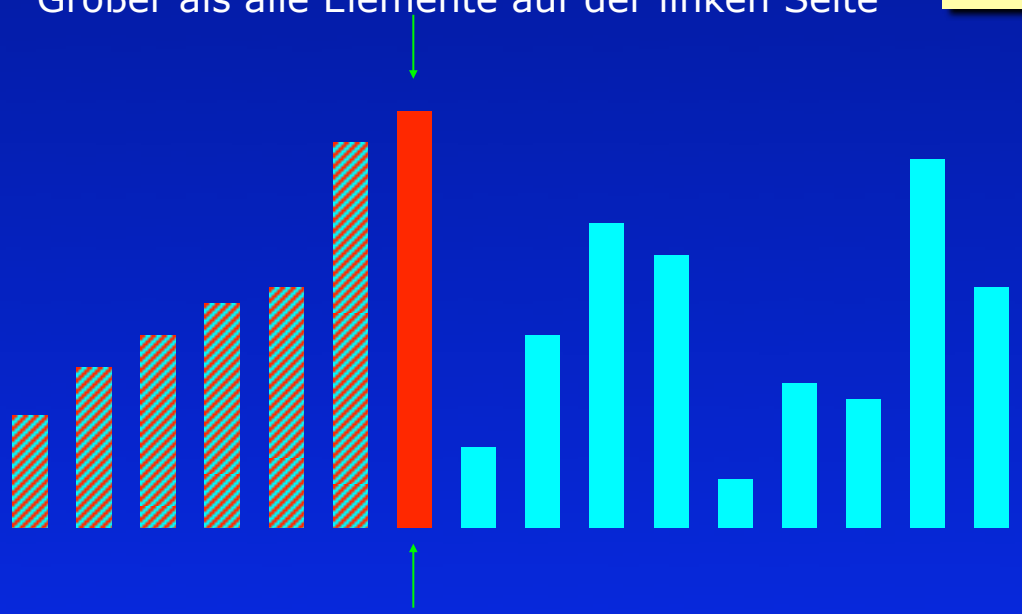


M. Esponda

Insertion-Sort

Bester Fall

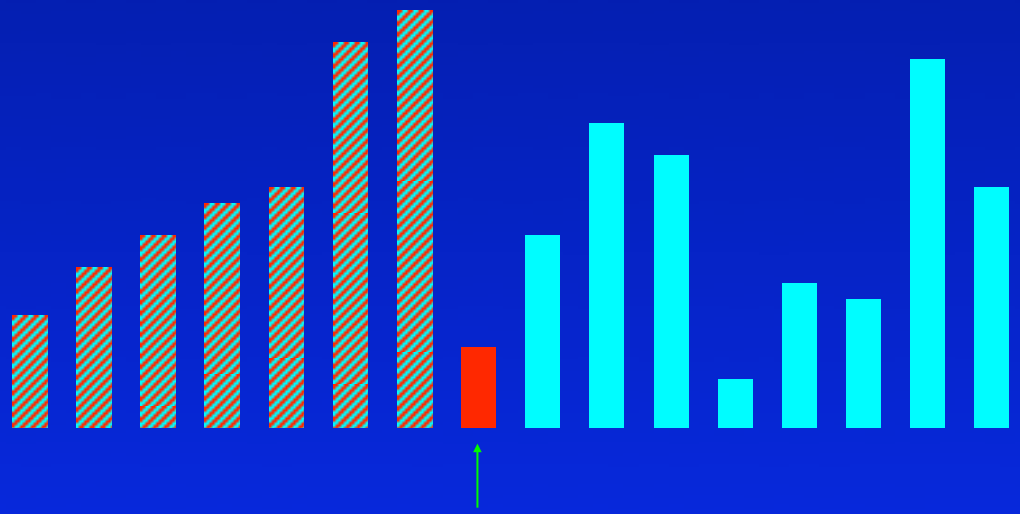
Größer als alle Elemente auf der linken Seite



Es ist kein weiterer
Vergleich notwendig

M. Esponda

Insertion-Sort

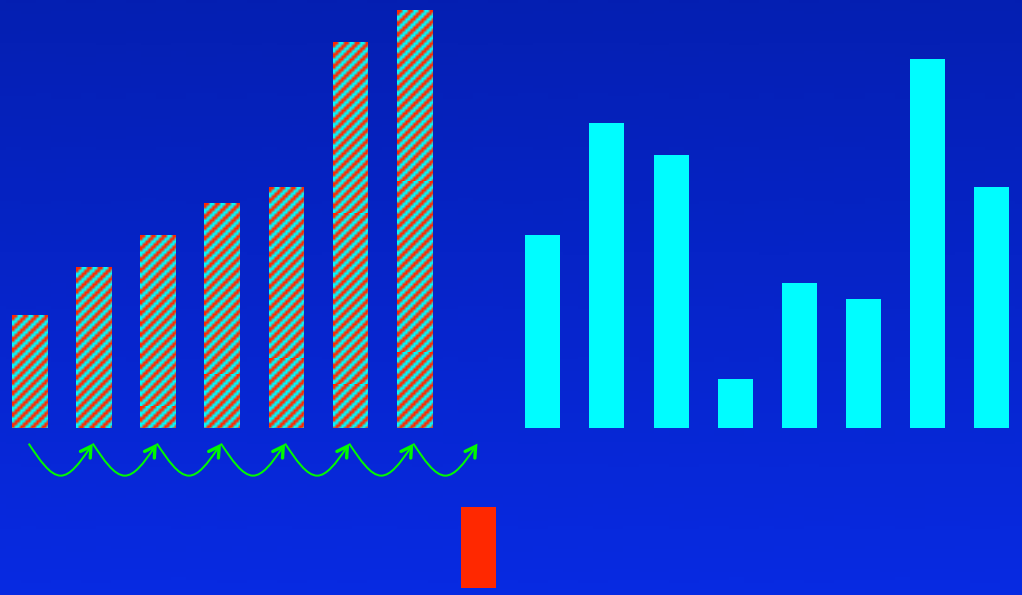


Kleiner als alle Elemente
der linken Seite

Schlimmster Fall

M. Esponda

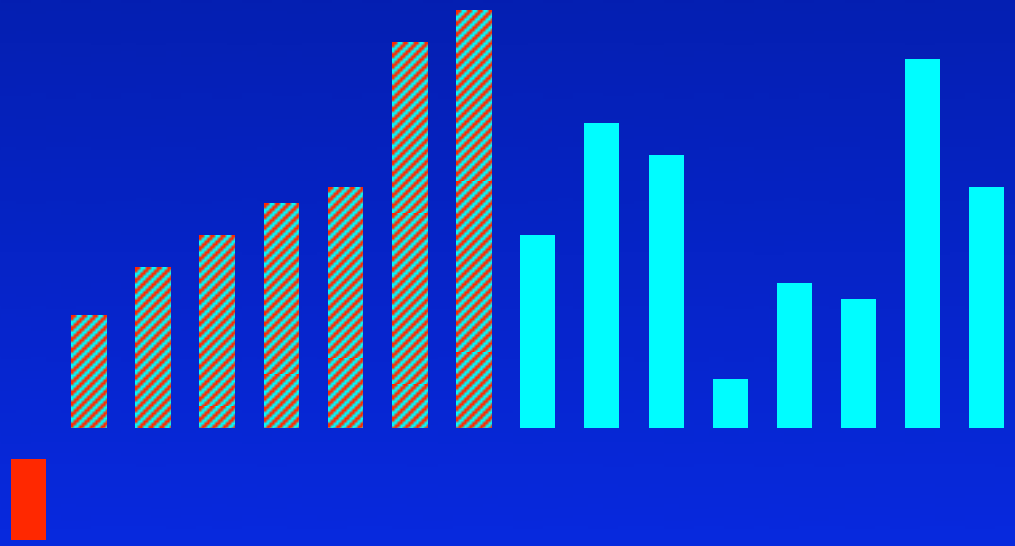
Insertion-Sort



Alle Elemente müssen verschoben werden

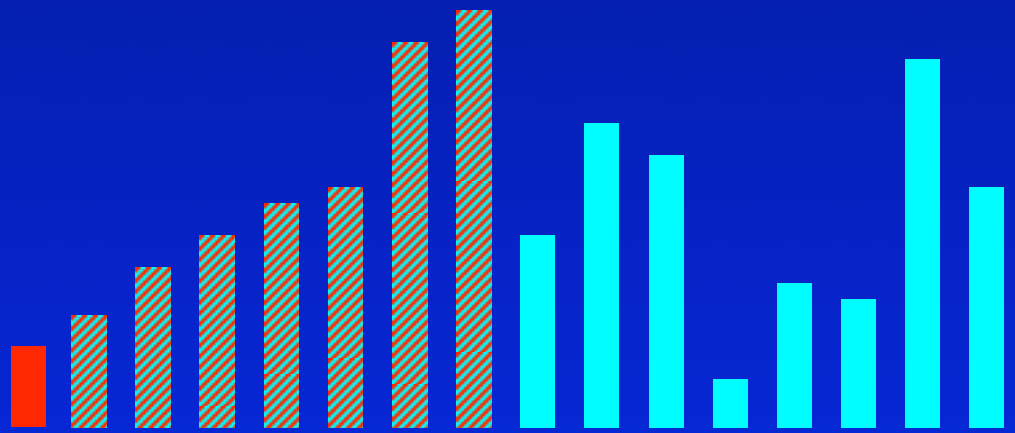
M. Esponda

Insertion-Sort



M. Esponda

Insertion-Sort



M. Esponda

Insertion-Sort

```
isort :: [ Integer ] -> [ Integer ]

isort [] = []

isort (a:x) = ins a (isort x)


ins :: Integer -> [Integer] -> [Integer]

ins a [] = [a]

ins a (b:y)

        |  a<= b  =  a:(b:y)

        | otherwise = b: (ins a y)
```

Das Problem in Haskell ist vor allem der Speicherverbrauch

Insertion-Sort (imperativ)

Einfacher Sortieralgorithmus

- **In-Place** und kein zusätzlicher Speicherbedarf O(1)
- **Stabil**
- **gut für kleine Mengen** oder **leicht unsortierte Informationen**

```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Eine geeignete Position wird gesucht und die Elemente des sortierten Bereichs verschoben

Die einzusortierende Zahl wird in den gefundenen Platz kopiert

Insertion-Sort (imperativ)


```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)


```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)


```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)

key

1

```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)


```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)


```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)


```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)


```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)

key

1

```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)


```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)

key

7

```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)


```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)


```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```

Insertion-Sort (imperativ)

key

7

```
def insertsort(seq):
    for j in range(1,len(seq)):
        key = seq[j]
        k = j-1;
        while k>=0 and seq[k]>key:
            seq[k+1] = seq[k]
            k = k-1
        seq[k+1] = key
```