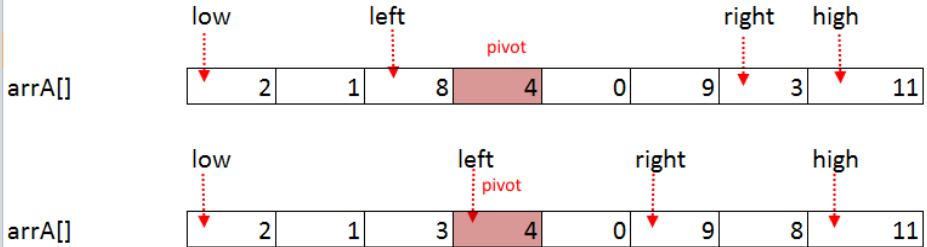


Algorithmen I - Tutorium 5

Sebastian Schmidt – *isibboi@gmail.com*

Arbeitsgruppe Kryptographie und Sicherheit



- Eines der verbreitetsten Probleme der Informatik
- In jeder Sprache in der Standardbibliothek vorhanden
- Wichtig für schnelle Indexdatenstrukturen jeder Art
- Quicksort beliebtes Thema von Bewerbungsgesprächen
- Easy to learn - Hard to master

- Eines der verbreitetsten Probleme der Informatik
- In jeder Sprache in der Standardbibliothek vorhanden
- Wichtig für schnelle Indexdatenstrukturen jeder Art
- Quicksort beliebtes Thema von Bewerbungsgesprächen
- Easy to learn - Hard to master

- Eines der verbreitetsten Probleme der Informatik
- In jeder Sprache in der Standardbibliothek vorhanden
- Wichtig für schnelle Indexdatenstrukturen jeder Art
- Quicksort beliebtes Thema von Bewerbungsgesprächen
- Easy to learn - Hard to master

- Eines der verbreitetsten Probleme der Informatik
- In jeder Sprache in der Standardbibliothek vorhanden
- Wichtig für schnelle Indexdatenstrukturen jeder Art
- Quicksort beliebtes Thema von Bewerbungsgesprächen
- Easy to learn - Hard to master

- Eines der verbreitetsten Probleme der Informatik
- In jeder Sprache in der Standardbibliothek vorhanden
- Wichtig für schnelle Indexdatenstrukturen jeder Art
- Quicksort beliebtes Thema von Bewerbungsgesprächen
- Easy to learn - Hard to master

Gegeben eine Folge von n Elementen $\alpha_1, \dots, \alpha_n$.

Finde eine Permutation σ , sodass gilt:

$$\forall i \in \{1, \dots, n-1\} : \alpha_{\sigma(i)} \leq \alpha_{\sigma(i+1)}$$

- Binary Tree Sort (höhen-balanciert)
- Binary Tree Sort
- Bogosort
- Bubblesort
- Bucketsort
- Combsort
- Countingsort
- Flashsort
- Gnomesort
- Heapsort
- Insertionsort
- Introsort
- Merge Insertion
- Mergesort
- Natural Mergesort
- Quicksort
- Radixsort (LSB)
- Radixsort (MSB)
- Selectionsort
- Shakersort (Cocktailsort)
- Shellsort
- Slowsort
- Smoothsort
- Stoogesort
- Swap-Sort
- Timsort

⇒ Es machen sich sehr viele Leute sehr viele Gedanken darüber, wie man schnell sortiert.

- Binary Tree Sort (höhen-balanciert)
- Binary Tree Sort
- Bogosort
- Bubblesort
- Bucketsort
- Combsort
- Countingsort
- Flashsort
- Gnomesort
- Heapsort
- Insertionsort
- Introsort
- Merge Insertion
- Mergesort
- Natural Mergesort
- Quicksort
- Radixsort (LSB)
- Radixsort (MSB)
- Selectionsort
- Shakersort (Cocktailsort)
- Shellsort
- Slowsort
- Smoothsort
- Stoogesort
- Swap-Sort
- Timsort

⇒ Es machen sich sehr viele Leute sehr viele Gedanken darüber, wie man schnell sortiert.

- Binary Tree Sort (höhen-balanciert)
- Binary Tree Sort
- Bogosort
- Bubblesort
- Bucketsort
- Combsort
- Countingsort
- Flashsort
- Gnomesort
- Heapsort
- Insertionsort
- Introsort
- Merge Insertion
- Mergesort
- Natural Mergesort
- Quicksort
- Radixsort (LSB)
- Radixsort (MSB)
- Selectionsort
- Shakersort (Cocktailsort)
- Shellsort
- Slowsort
- Smoothsort
- Stoogesort
- Swap-Sort
- Timsort

⇒ Es machen sich sehr viele Leute sehr viele Gedanken darüber, wie man schnell sortiert.

Wie heißt der Algorithmus?

```
function SORT( $A : \text{Number}[1, \dots, n]$ )  
  for  $i$  from 1 to  $n - 1$  do  
     $k \leftarrow i$   
    for  $j$  from  $i + 1$  to  $n$  do  
      if  $A[k] > A[j]$  then  
         $k \leftarrow j$   
      end if  
    end for  
    SWAP( $A, i, k$ )  
  end for  
end function
```

Wie heißt der Algorithmus?

```
function SELECTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
  for  $sorted$  from 1 to  $n - 1$  do  
     $minIndex \leftarrow sorted$   
    for  $index$  from  $sorted + 1$  to  $n$  do  
      if  $A[minIndex] > A[index]$  then  
         $minIndex \leftarrow index$   
      end if  
    end for  
    SWAP( $A, sorted, minIndex$ )  
  end for  
end function
```

Welche Laufzeit hat der Algorithmus im Worst-Case im O -Kalkül? Was ist eine Worst-Case-Eingabe?

```
function SELECTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
  for  $sorted$  from 1 to  $n - 1$  do  
     $minIndex \leftarrow sorted$   
    for  $index$  from  $sorted + 1$  to  $n$  do  
      if  $A[minIndex] > A[index]$  then  
         $minIndex \leftarrow index$   
      end if  
    end for  
    SWAP( $A, sorted, minIndex$ )  
  end for  
end function
```

Welche Laufzeit hat der Algorithmus im Best-Case im O -Kalkül? Was ist eine Best-Case-Eingabe?

```
function SELECTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
  for  $sorted$  from 1 to  $n - 1$  do  
     $minIndex \leftarrow sorted$   
    for  $index$  from  $sorted + 1$  to  $n$  do  
      if  $A[minIndex] > A[index]$  then  
         $minIndex \leftarrow index$   
      end if  
    end for  
    SWAP( $A, sorted, minIndex$ )  
  end for  
end function
```

Welche Laufzeit hat der Algorithmus im Average-Case im O -Kalkül?

```
function SELECTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
  for  $sorted$  from 1 to  $n - 1$  do  
     $minIndex \leftarrow sorted$   
    for  $index$  from  $sorted + 1$  to  $n$  do  
      if  $A[minIndex] > A[index]$  then  
         $minIndex \leftarrow index$   
      end if  
    end for  
    SWAP( $A, sorted, minIndex$ )  
  end for  
end function
```

Wie ist der zusätzliche Speicherbedarf im O -Kalkül?

```
function SELECTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
  for  $sorted$  from 1 to  $n - 1$  do  
     $minIndex \leftarrow sorted$   
    for  $index$  from  $sorted + 1$  to  $n$  do  
      if  $A[minIndex] > A[index]$  then  
         $minIndex \leftarrow index$   
      end if  
    end for  
    SWAP( $A, sorted, minIndex$ )  
  end for  
end function
```

Ist der Sortieralgorithmus stabil?

```
function SELECTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
  for  $sorted$  from 1 to  $n - 1$  do  
     $minIndex \leftarrow sorted$   
    for  $index$  from  $sorted + 1$  to  $n$  do  
      if  $A[minIndex] > A[index]$  then  
         $minIndex \leftarrow index$   
      end if  
    end for  
    SWAP( $A, sorted, minIndex$ )  
  end for  
end function
```

Wie heißt der Algorithmus?

```
function SORT( $A : \text{Number}[1, \dots, n]$ )  
   $A[0] \leftarrow -\infty$   
  for  $i$  from 2 to  $n$  do  
     $j \leftarrow i - 1$   
    while  $A[j] > A[j + 1]$  do  
      SWAP( $A, j, j + 1$ )  
       $j \leftarrow j - 1$   
    end while  
  end for  
end function
```

Wie heißt der Algorithmus?

```
function INSERTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
   $A[0] \leftarrow -\infty$   
  for sorted from 2 to  $n$  do  
     $\text{swapIndex} \leftarrow \text{sorted} - 1$   
    while  $A[\text{swapIndex}] > A[\text{swapIndex} + 1]$  do  
      SWAP( $A, \text{swapIndex}, \text{swapIndex} + 1$ )  
       $\text{swapIndex} \leftarrow \text{swapIndex} - 1$   
    end while  
  end for  
end function
```

Welche Laufzeit hat der Algorithmus im Worst-Case im O -Kalkül? Was ist eine Worst-Case-Eingabe?

```
function INSERTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
   $A[0] \leftarrow -\infty$   
  for sorted from 2 to  $n$  do  
     $\text{swapIndex} \leftarrow \text{sorted} - 1$   
    while  $A[\text{swapIndex}] > A[\text{swapIndex} + 1]$  do  
      SWAP( $A, \text{swapIndex}, \text{swapIndex} + 1$ )  
       $\text{swapIndex} \leftarrow \text{swapIndex} - 1$   
    end while  
  end for  
end function
```

Welche Laufzeit hat der Algorithmus im Best-Case im O -Kalkül? Was ist eine Best-Case-Eingabe?

```
function INSERTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
   $A[0] \leftarrow -\infty$   
  for sorted from 2 to  $n$  do  
     $\text{swapIndex} \leftarrow \text{sorted} - 1$   
    while  $A[\text{swapIndex}] > A[\text{swapIndex} + 1]$  do  
      SWAP( $A, \text{swapIndex}, \text{swapIndex} + 1$ )  
       $\text{swapIndex} \leftarrow \text{swapIndex} - 1$   
    end while  
  end for  
end function
```

Welche Laufzeit hat der Algorithmus im Average-Case im O -Kalkül?

```
function INSERTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
     $A[0] \leftarrow -\infty$   
    for sorted from 2 to  $n$  do  
         $\text{swapIndex} \leftarrow \text{sorted} - 1$   
        while  $A[\text{swapIndex}] > A[\text{swapIndex} + 1]$  do  
            SWAP( $A, \text{swapIndex}, \text{swapIndex} + 1$ )  
             $\text{swapIndex} \leftarrow \text{swapIndex} - 1$   
        end while  
    end for  
end function
```

Wie ist der zusätzliche Speicherbedarf im O -Kalkül?

```
function INSERTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
   $A[0] \leftarrow -\infty$   
  for sorted from 2 to  $n$  do  
     $\text{swapIndex} \leftarrow \text{sorted} - 1$   
    while  $A[\text{swapIndex}] > A[\text{swapIndex} + 1]$  do  
      SWAP( $A, \text{swapIndex}, \text{swapIndex} + 1$ )  
       $\text{swapIndex} \leftarrow \text{swapIndex} - 1$   
    end while  
  end for  
end function
```

Ist der Sortieralgorithmus stabil?

```
function INSERTIONSORT( $A : \text{Number}[1, \dots, n]$ )  
   $A[0] \leftarrow -\infty$   
  for sorted from 2 to  $n$  do  
     $\text{swapIndex} \leftarrow \text{sorted} - 1$   
    while  $A[\text{swapIndex}] > A[\text{swapIndex} + 1]$  do  
      SWAP( $A, \text{swapIndex}, \text{swapIndex} + 1$ )  
       $\text{swapIndex} \leftarrow \text{swapIndex} - 1$   
    end while  
  end for  
end function
```

Wie heißt der Algorithmus?

```
function SORT( $A : \text{Number}[1, \dots, n]$ )  
  if  $n = 1$  then return  $A$   
  else  
     $(A_1, A_2) \leftarrow \text{SPLITTOEQUALSIZE}(A)$   
     $(B_1, B_2) \leftarrow (\text{SORT}(A_1), \text{SORT}(A_2))$   
     $B \leftarrow \text{new Number}[1, \dots, n]$   
     $i_1 \leftarrow i_2 \leftarrow 1$   
    for  $j$  from 1 to  $n$  do  
       $B[j] \leftarrow \text{MIN}(B_1[i_1], B_2[i_2])$   
      Increment  $i_{1/2}$  if it was chosen  
    end for  
  end if  
  return  $B \text{ concat } \infty$   
end function
```

Wie heißt der Algorithmus?

```
function MERGESORT( $A : \text{Number}[1, \dots, n]$ )  
  if  $n = 1$  then return  $A$   
  else  
     $(A_1, A_2) \leftarrow \text{SPLITTOEQUALSIZE}(A)$   
     $(B_1, B_2) \leftarrow (\text{MERGESORT}(A_1), \text{MERGESORT}(A_2))$   
     $B \leftarrow \text{new Number}[1, \dots, n]$   
     $\text{lowest}_1 \leftarrow \text{lowest}_2 \leftarrow 1$   
    for  $\text{index}$  from 1 to  $n$  do  
       $B[\text{index}] \leftarrow \text{MIN}(B_1[\text{lowest}_1], B_2[\text{lowest}_2])$   
      Increment  $\text{lowest}_{1/2}$  if it was chosen  
    end for  
  end if  
  return  $B \text{ concat } \infty$   
end function
```

Schon wieder ein Sortieralgorithmus

Gibt es einen Unterschied zwischen Worst-Case- und Best-Case-Laufzeit im O -Kalkül?

```
function MERGESORT( $A : \text{Number}[1, \dots, n]$ )  
  if  $n = 1$  then return  $A$   
  else  
     $(A_1, A_2) \leftarrow \text{SPLITTOEQUALSIZE}(A)$   
     $(B_1, B_2) \leftarrow (\text{MERGESORT}(A_1), \text{MERGESORT}(A_2))$   
     $B \leftarrow \text{new Number}[1, \dots, n]$   
     $\text{lowest}_1 \leftarrow \text{lowest}_2 \leftarrow 1$   
    for  $\text{index}$  from 1 to  $n$  do  
       $B[\text{index}] \leftarrow \text{MIN}(B_1[\text{lowest}_1], B_2[\text{lowest}_2])$   
      Increment  $\text{lowest}_{1/2}$  if it was chosen  
    end for  
  end if  
  return  $B$  concat  $\infty$   
end function
```

Welche Laufzeit hat der Algorithmus im Worst-Case im O -Kalkül?

```
function MERGESORT( $A : \text{Number}[1, \dots, n]$ )  
  if  $n = 1$  then return  $A$   
  else  
     $(A_1, A_2) \leftarrow \text{SPLITTOEQUALSIZE}(A)$   
     $(B_1, B_2) \leftarrow (\text{MERGESORT}(A_1), \text{MERGESORT}(A_2))$   
     $B \leftarrow \text{new Number}[1, \dots, n]$   
     $\text{lowest}_1 \leftarrow \text{lowest}_2 \leftarrow 1$   
    for  $\text{index}$  from 1 to  $n$  do  
       $B[\text{index}] \leftarrow \text{MIN}(B_1[\text{lowest}_1], B_2[\text{lowest}_2])$   
      Increment  $\text{lowest}_{1/2}$  if it was chosen  
    end for  
  end if  
  return  $B \text{ concat } \infty$   
end function
```

Wie ist der zusätzliche Speicherbedarf im O -Kalkül?

```
function MERGESORT( $A : \text{Number}[1, \dots, n]$ )  
  if  $n = 1$  then return  $A$   
  else  
     $(A_1, A_2) \leftarrow \text{SPLITTOEQUALSIZE}(A)$   
     $(B_1, B_2) \leftarrow (\text{MERGESORT}(A_1), \text{MERGESORT}(A_2))$   
     $B \leftarrow \text{new Number}[1, \dots, n]$   
     $\text{lowest}_1 \leftarrow \text{lowest}_2 \leftarrow 1$   
    for  $\text{index}$  from 1 to  $n$  do  
       $B[\text{index}] \leftarrow \text{MIN}(B_1[\text{lowest}_1], B_2[\text{lowest}_2])$   
      Increment  $\text{lowest}_{1/2}$  if it was chosen  
    end for  
  end if  
  return  $B \text{ concat } \infty$   
end function
```

Schon wieder ein Sortieralgorithmus

Bekommt man auch $O(n)$ zusätzlichen Speicher hin?

```
function MERGESORT( $A : \text{Number}[1, \dots, n]$ )  
  if  $n = 1$  then return  $A$   
  else  
     $(A_1, A_2) \leftarrow \text{SPLITTOEQUALSIZE}(A)$   
     $(B_1, B_2) \leftarrow (\text{MERGESORT}(A_1), \text{MERGESORT}(A_2))$   
     $B \leftarrow \text{new Number}[1, \dots, n]$   
     $\text{lowest}_1 \leftarrow \text{lowest}_2 \leftarrow 1$   
    for  $\text{index}$  from 1 to  $n$  do  
       $B[\text{index}] \leftarrow \text{MIN}(B_1[\text{lowest}_1], B_2[\text{lowest}_2])$   
      Increment  $\text{lowest}_{1/2}$  if it was chosen  
    end for  
  end if  
  return  $B \text{ concat } \infty$   
end function
```

Ist der Sortialgorithmus stabil?

```
function MERGESORT( $A : \text{Number}[1, \dots, n]$ )  
  if  $n = 1$  then return  $A$   
  else  
     $(A_1, A_2) \leftarrow \text{SPLITTOEQUALSIZE}(A)$   
     $(B_1, B_2) \leftarrow (\text{MERGESORT}(A_1), \text{MERGESORT}(A_2))$   
     $B \leftarrow \text{new Number}[1, \dots, n]$   
     $\text{lowest}_1 \leftarrow \text{lowest}_2 \leftarrow 1$   
    for  $\text{index}$  from 1 to  $n$  do  
       $B[\text{index}] \leftarrow \text{MIN}(B_1[\text{lowest}_1], B_2[\text{lowest}_2])$   
      Increment  $\text{lowest}_{1/2}$  if it was chosen  
    end for  
  end if  
  return  $B \text{ concat } \infty$   
end function
```
