Asymptotische Laufzeitanalyse: Beispiel

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Insertionsort:

```
IS( A[] )
   n = length( A )
   for i = 2 to n do
      x = A[i]
      j = i
      while j>1 and A[j-1] > x do
            A[j] = A[j-1]
            j = j-1
      A[j] = x
```


Asymptotische Laufzeitanalyse: Beispiel

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Insertionsort:

```
IS( A[] )
   n = length( A )          } O(1)
   for i = 2 to n do
      x = A[i]              }
      j = i                 } O(1)
      while j>1 and A[j-1] > x do
            A[j] = A[j-1]   }
            j = j-1         } O(1)
      A[j] = x              } O(1)
```

Asymptotische Laufzeitanalyse: Beispiel

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Insertionsort:

```
IS( A[] )
  n = length( A )                    } O(1)
  for i = 2 to n do
    x = A[i]                         }
    j = i                            } O(1)
    while j>1 and A[j-1] > x do      }
          A[j] = A[j-1]              } O(1)   } i * O(1) = O(i)
          j = j-1                    }
    A[j] = x                         } O(1)
```


Asymptotische Laufzeitanalyse: Beispiel

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Insertionsort:

```
IS( A[] )
  n = length( A )                    } O(1)
  for i = 2 to n do                  }
    x = A[i]                         }
    j = i                            } O(1)
    while j>1 and A[j-1] > x do      }
          A[j] = A[j-1]              } O(1)   } i * O(1) = O(i)   } O(i)
          j = j-1                    }
    A[j] = x                         } O(1)
```

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Asymptotische Laufzeitanalyse: Beispiel

Insertionsort:

```
IS( A[] )
   n = length( A )          } O(1)
   for i = 2 to n do
      x = A[i]
      j = i                 } O(1)
      while j>1 and A[j-1] > x do
            A[j] = A[j-1]
            j = j-1         } O(1)   } i * O(1) = O(i)
      A[j] = x              } O(1)
                                                      } O(i)   } Σ_{2≤i≤n} O(i) = O(n²)
```

$\sum_{2 \leq i \leq n} O(i) = O(n^2)$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Asymptotische Laufzeitanalyse: Beispiel

Insertionsort:

```
IS( A[] )
   n = length( A )          } O(1)
   for i = 2 to n do
      x = A[i]
      j = i                 } O(1)
      while j>1 and A[j-1] > x do
            A[j] = A[j-1]
            j = j-1         } O(1)   } i * O(1) = O(i)
      A[j] = x              } O(1)
                                                      } O(i)   } Σ_{2≤i≤n} O(i) = O(n²)
```

$\sum_{2 \leq i \leq n} O(i) = O(n^2)$

Gesamt: $O(1)+O(n^2) = O(n^2)$

Asymptotische Laufzeitanalyse: Beispiel

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Insertionsort:

```
IS( A[] )
   n = length( A )              } O(1)
   for i = 2 to n do
      x = A[i]
      j = i                     } O(1)
      while j>1 and A[j-1] > x do
            A[j] = A[j-1]
            j = j-1             } O(1)   } i * O(1) = O(i)
      A[j] = x                  } O(1)
                                                           } O(i)
                                                                   } Σ_{2≤i≤n} O(i) = O(n²)
```

$\Sigma_{2 \leq i \leq n} O(i) = O(n^2)$

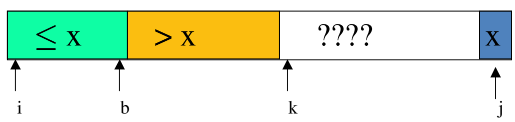
Gesamt: $O(1)+O(n^2) = O(n^2)$

"quadratisch in der Feldgröße n"

Partitionieren

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Invariante:

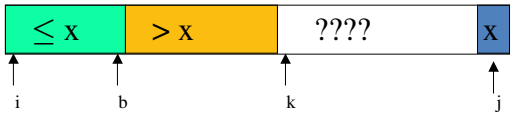



```
Partition( A , i , j , x )
      b = i-1
      for k = i to j do
            swap( A[k] , A[b+1] )
            if  A[b+1] ≤ x then b++
      return b
```

Partitionieren

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Invariante:

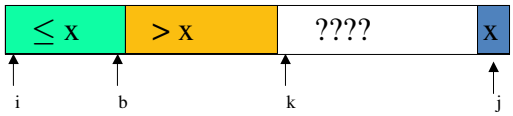



```
Partition( A , i , j , x )
    b = i-1                          } O(1)
    for k = i to j do
        swap( A[k] , A[b+1] )        } O(1)
        if  A[b+1] ≤ x then b++      }
    return b                         } O(1)
```


Partitionieren

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Invariante:



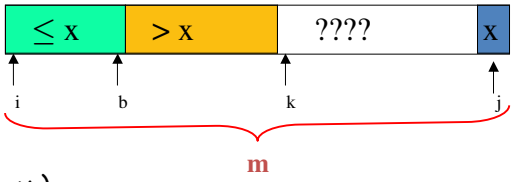

```
Partition( A , i , j , x )
    b = i-1                          } O(1)
    for k = i to j do                }
        swap( A[k] , A[b+1] )        } O(1)
        if  A[b+1] ≤ x then b++      }
    return b                         } O(1)
```

$(j-i+1) \cdot O(i) = O(j-i+1)$

Partitionieren

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Invariante:



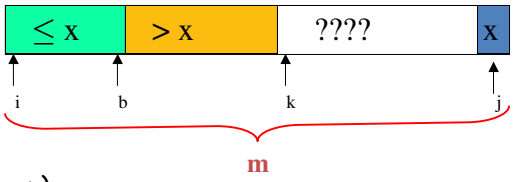

```
Partition( A , i , j , x )
    b = i-1                                  } O(1)
    for k = i to j do                        } (j-i+1) · O(i) = O(j-i+1)
        swap( A[k] , A[b+1] )                } O(1)
        if  A[b+1] ≤ x then b++              }
    return b                                 } O(1)
```

(j-i+1) = m

Partitionieren

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Invariante:



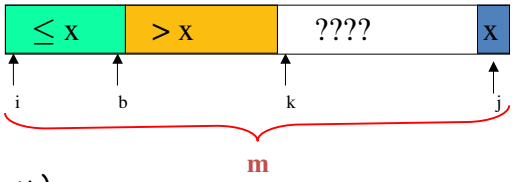

```
Partition( A , i , j , x )
    b = i-1                                  } O(1)
    for k = i to j do                        } (j-i+1) · O(i) = O(j-i+1)
        swap( A[k] , A[b+1] )                } O(1)
        if  A[b+1] ≤ x then b++              }
    return b                                 } O(1)
```

(j-i+1) = m

Gesamt: O(1)+O(m) = O(m)

Partitionieren

Invariante:



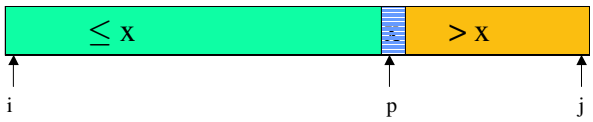

```
Partition( A , i , j , x )
    b = i-1                          } O(1)
    for k = i to j do                } (j-i+1) · O(i) = O(j-i+1)
        swap( A[k] , A[b+1] )        } O(1)
        if A[b+1] ≤ x then b++       }
    return b                         } O(1)
```

Gesamt: O(1)+O(m) = O(m)

"linear in der Teilfeldgröße m"

Quicksort

Idee: wähle „Pivotelement“ x in Feld und stelle Feld so um:



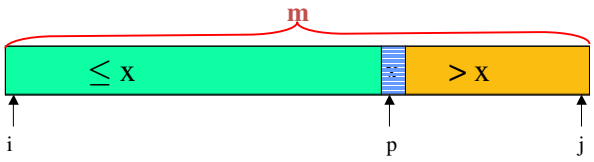
sortiere Teilfeld der „kleinen“ Elemente (≤ x) rekursiv
sortiere Teilfeld der „großen“ Elemente (> x) rekursiv

```
Quicksort( A , i , j )
    if i < j then
        p = Partition( A , i , j , A[j] )
        Quicksort( A , i , p-1 )
        Quicksort( A , p+1 , j )
```

Quicksort

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Idee: wähle „Pivotelement“ x in Feld und stelle Feld so um:



sortiere Teilfeld der „kleinen“ Elemente (≤ x) rekursiv
sortiere Teilfeld der „großen“ Elemente (> x) rekursiv

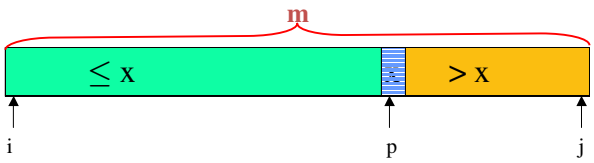
T(m) ... worst case Laufzeit von Quicksort auf einem Teilfeld der Größe m

```
Quicksort( A , i , j )
    if  i < j then
        p = Partition( A , i , j , A[j] )
        Quicksort( A , i , p-1 )
        Quicksort( A , p+1 , j )
```


SAARLAND UNIVERSITY
COMPUTER SCIENCE

Quicksort

Idee: wähle „Pivotelement“ x in Feld und stelle Feld so um:



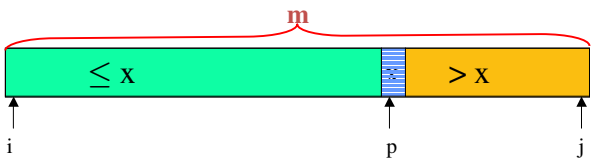
sortiere Teilfeld der „kleinen“ Elemente (≤ x) rekursiv
sortiere Teilfeld der „großen“ Elemente (> x) rekursiv

T(m) ... worst case Laufzeit von Quicksort auf einem Teilfeld der Größe m

```
Quicksort( A , i , j )
    if  i < j then
        p = Partition( A , i , j , A[j] )      } O(m)
        Quicksort( A , i , p-1 )               } T(m1)
        Quicksort( A , p+1 , j )               } T(m2)
```

Quicksort

Idee: wähle „Pivotelement“ x in Feld und stelle Feld so um:



sortiere Teilfeld der „kleinen“ Elemente (≤ x) rekursiv
sortiere Teilfeld der „großen“ Elemente (> x) rekursiv

T(m) ... worst case Laufzeit von Quicksort auf einem Teilfeld der Größe m

```
Quicksort( A , i , j )
    if  i < j then
        p = Partition( A , i , j , A[j] )      } O(m)
        Quicksort( A , i , p-1 )               } T(m1)     m1 = p-i
        Quicksort( A , p+1 , j )               } T(m2)     m2 = m-1-m1
```

$T(m) \leq O(m) + T(m_1) + T(m_2)$ **mit** $m_1 + m_2 = m-1,\ m_1, m_2 \geq 0$

T(m) ... worst case Laufzeit von Quicksort auf einem Teilfeld der Größe m

$T(m) \leq O(m) + T(m_1) + T(m_2)$ **mit** $m_1 + m_2 = m-1,\ m_1, m_2 \geq 0$

$T(m) \leq c \cdot m + T(m_1) + T(m_2)$ **mit** $m_1 + m_2 = m-1,\ m_1, m_2 \geq 0$ **für** $m>1$

$T(m) \leq c$ **für** $m \leq 1$

$c > 0$ **eine Konstante**

Behauptung: Es gilt $T(m) \leq d \cdot m^2$ für ein geeignetes, von c abhängiges d, also

$T(m) = O(m^2)$

Beweis durch Induktion über m, unter der Verwendung der Tatsache, dass

$\max\{ m_1^2 + m_2^2 \mid m_1 + m_2 = m-1,\ m_1, m_2 \geq 0 \} = (m-1)^2$

Erwartete Laufzeit von Quicksort

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Angenehme Annahme: Beim Partitionieren tritt mit gleicher Wahrscheinlichkeit 1/m ein, dass (m_1,m_2) eines der m Paare

$$(0,m-1)\ ,\ (1,m-2)\ ,\ (2,m-3)\ ,\ \cdots\ ,\ (m-2,1),\ (m-1,0)$$

t(m) ... erwartete Laufzeit von Quicksort auf Feld mit m Schlüsseln, unter der angenehmen Annahme

$t(m) \leq c\cdot m + \sum_{0\leq i<m}(1/m)(\ t(i) + t(m-1-i)\ )$ mit $\quad i \approx m_1 \quad m-1-i \approx m_2$

Behauptung: Es gilt $t(m) \leq d\cdot m\cdot \log m$ für ein geeignetes, von c abhängiges d, also

$$T(m) = O(m\cdot \log m)$$

Beweis durch Induktion über m, unter der Verwendung der Tatsache, dass

$$\sum_{1\leq i <m} i\cdot\log i \ \leq\ \int_{1\leq x\leq m} x\cdot\log x\ dx \ =\ (m^2\cdot\log m)/2 - m^2/4+1/4$$

Erwartete Laufzeit von Quicksort

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Angenehme Annahme: Beim Partitionieren tritt mit gleicher Wahrscheinlichkeit 1/m ein, dass (m_1,m_2) eines der m Paare

$$(0,m-1)\ ,\ (1,m-2)\ ,\ (2,m-3)\ ,\ \cdots\ ,\ (m-2,1),\ (m-1,0)$$

t(m) ... erwartete Laufzeit von Quicksort auf Feld mit m Schlüsseln, unter der angenehmen Annahme

Angenehme Annahme realisieren:

1) annehmen die Eingaben seien zufällig dieser Art
2) im Voraus das Eingabefeld zufällig permutieren
3) das "Pivotelement" fürs Partitionieren jedesmal zufällig wählen