Sortieren

Proseminar Programming Pearls, Sommersemester 2001

Christian Kuester

Sortieren

- Sortieren durch Einfuegen

Sortieren

- Sortieren durch Einfuegen
- Quicksort

Sortieren

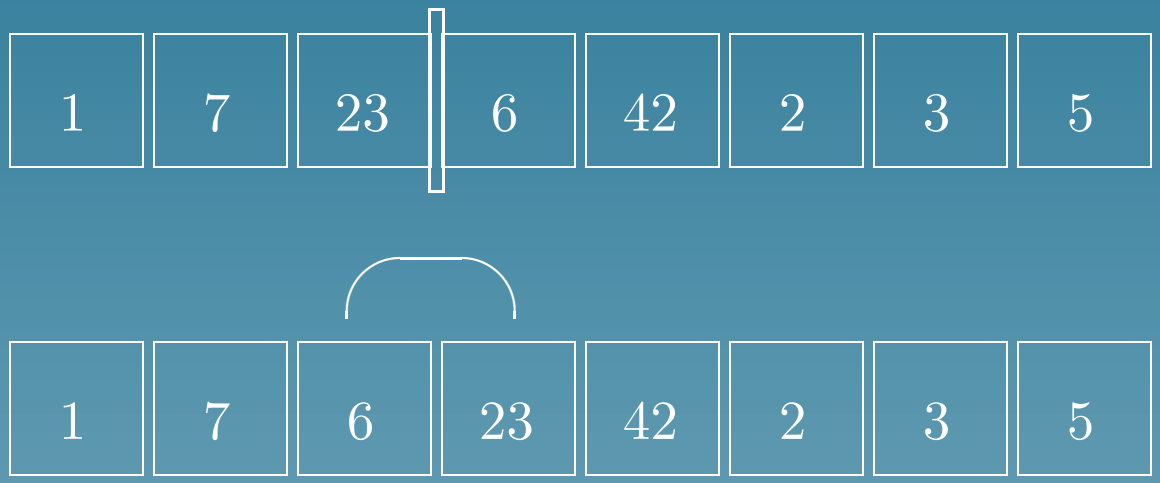
- Sortieren durch Einfuegen
- Quicksort
- Mergesort

Sortieren durch Einfuegen (InsertionSort)

Funktionsweise:

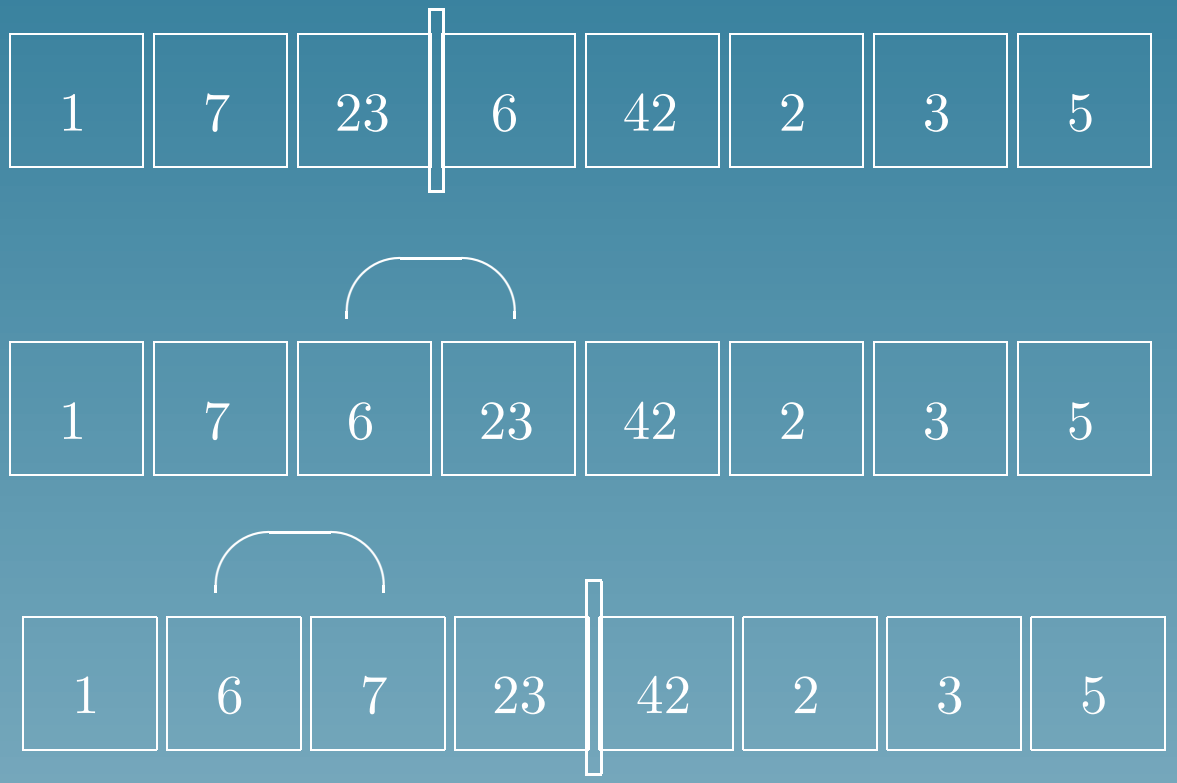
Sortieren durch Einfuegen (InsertionSort)

Funktionsweise:



Sortieren durch Einfuegen (InsertionSort)

Funktionsweise:



Sortieren durch Einfuegen

Der erste Ansatz:

```
01:    for (i = 0; i < 30000; i++)
02:        for (j = i; j > 0 && array[j-1] > array[j]; j--)
03:            this->swapint(array[j-1], array[j]);
```

Sortieren durch Einfuegen

Der erste Ansatz:

```
01:    for (i = 0; i < 30000; i++)
02:        for (j = i; j > 0 && array[j-1] > array[j]; j--)
03:            this->swapint(array[j-1], array[j]);
```

```
christian@maggie> time ./sorter i1
0m25.007s
christian@maggie>
```

Sortieren durch Einfuegen

```
01: int i, j, t;
02: for (i = 0; i < 30000; i++)
03:   for (j = i; j > 0 && array[j-1] > array[j]; j--) {
04:     t = array[j];
05:     array[j] =  array[j-1];
06:     array[j-1] = t;
07:  }
```

Sortieren durch Einfuegen

```
01: int i, j, t;
02: for (i = 0; i < 30000; i++)
03:   for (j = i; j > 0 && array[j-1] > array[j]; j--) {
04:     t = array[j];
05:     array[j] =  array[j-1];
06:     array[j-1] = t;
07:  }
```

0m15.115s

Sortieren durch Einfuegen

```
01: int i, t, j;
02: for (i = 0; i < 30000; i++) {
03:   t = array[i];
04:   for (j = i; j > 0 && array[j-1] > t; j--)
05:     array[j] = array[j-1];
06:   array[j] = t;
07: }
```

Sortieren durch Einfuegen

```
01: int i, t, j;
02: for (i = 0; i < 30000; i++) {
03:   t = array[i];
04:   for (j = i; j > 0 && array[j-1] > t; j--)
05:     array[j] = array[j-1];
06:   array[j] = t;
07: }
```

0m8.985s

Sortieren durch Einfuegen

```
01: int i, t, j;
02: for (i = 0; i < 30000; ++i) {
03:   t = array[i];
04:   for (j = i; j > 0 && array[j-1] > t; --j)
05:     array[j] = array[j-1];
06:   array[j] = t;
07: }
```

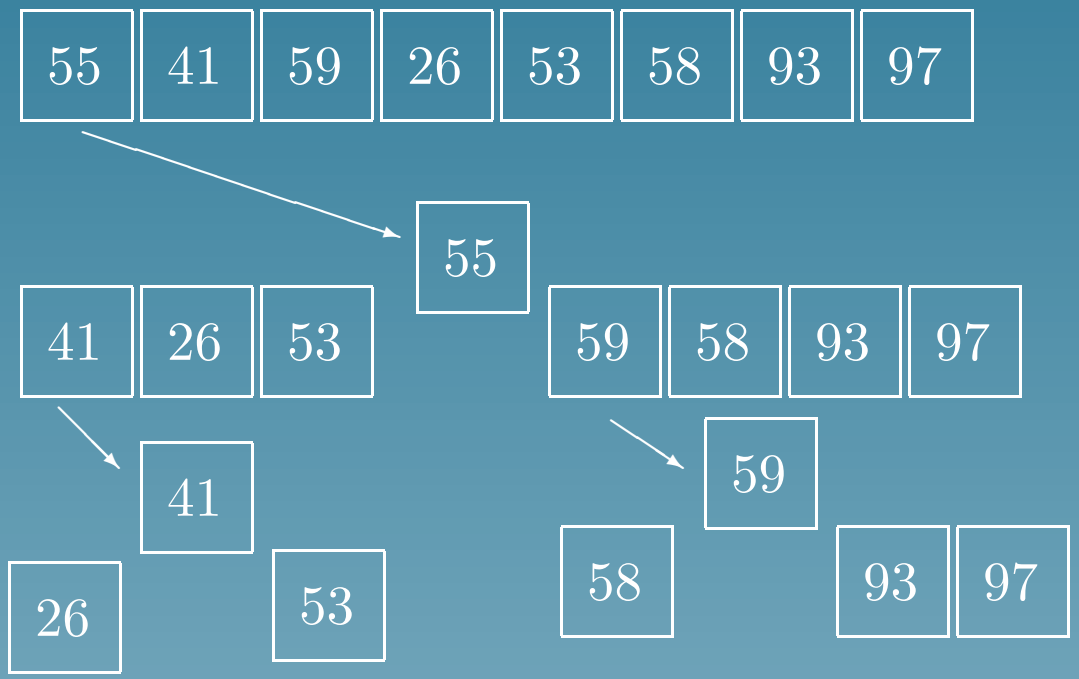
Sortieren durch Einfuegen

```
01: int i, t, j;
02: for (i = 0; i < 30000; ++i) {
03:   t = array[i];
04:   for (j = i; j > 0 && array[j-1] > t; --j)
05:     array[j] = array[j-1];
06:   array[j] = t;
07: }
```

0m8.594s

Quicksort

Funktionsweise:



Quicksort

Die erste Implementation:

```
00: void Qsort1::qsort1( int array[], int l, int u ) {
01: if (l >= u) return;
02: int m = l;
03: for (int i = l+1; i <= u; ++i) {
04:  if (array[i] < array[l])
05:    this->swapint(array[++m], array[i]);
06: }
07: this->swapint(array[l], array[m]);
08: this->qsort1(array, l, m-1);
09: this->qsort1(array, m+1, u); }
```

Quicksort

Die erste Implementation:

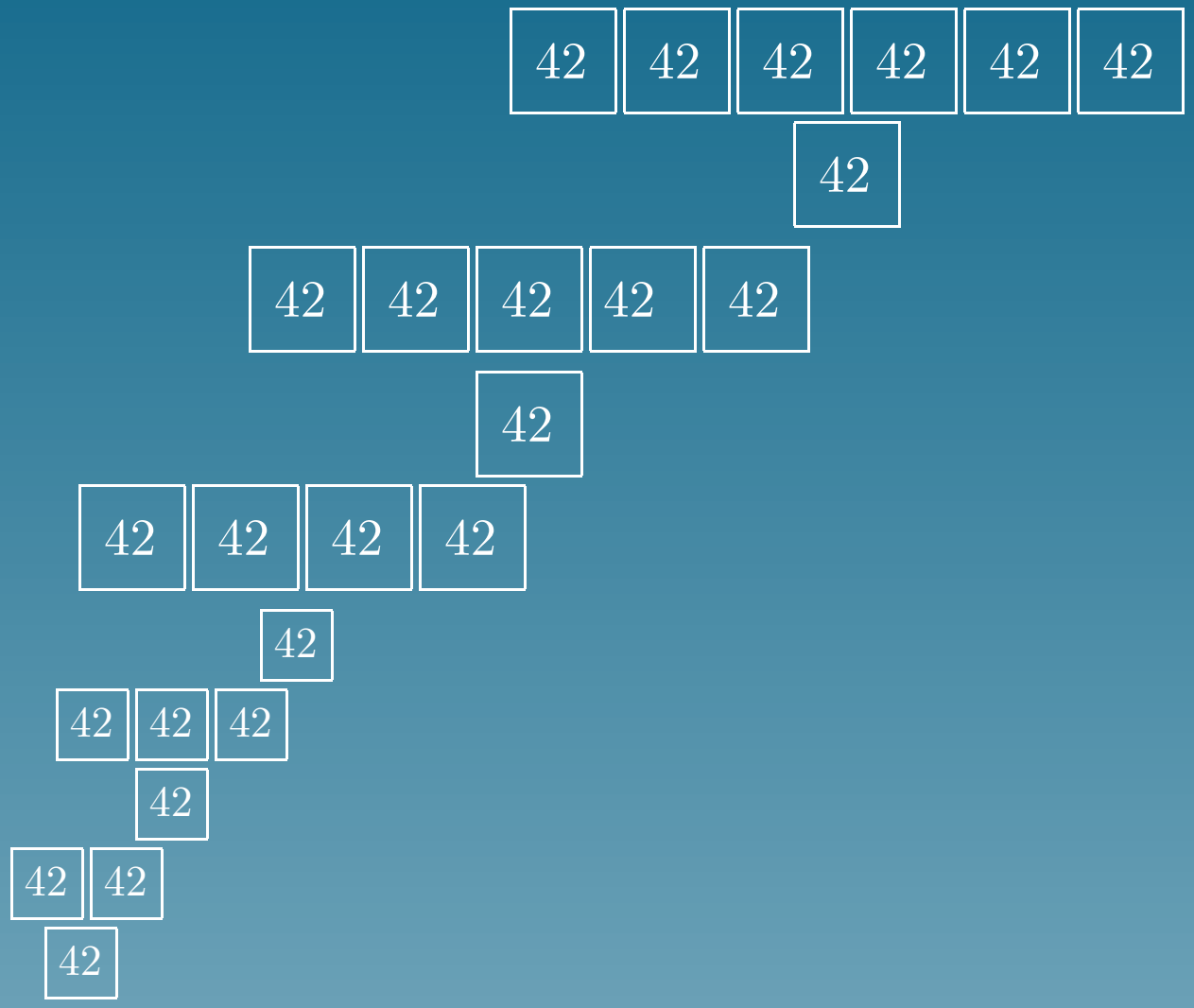
```
00: void Qsort1::qsort1( int array[], int l, int u ) {
01: if (l >= u) return;
02: int m = l;
03: for (int i = l+1; i <= u; ++i) {
04:  if (array[i] < array[l])
05:    this->swapint(array[++m], array[i]);
06: }
07: this->swapint(array[l], array[m]);
08: this->qsort1(array, l, m-1);
09: this->qsort1(array, m+1, u); }
```

0m0.129s

30.000 gleiche Elemente: 0m11.973s

Quicksort

Was ist da passiert?


42 42 42 42 42 42
42
42 42 42 42 42
42
42 42 42 42
42
42 42 42
42
42 42
42

QuickSort

```
01: if (l >= u) return;
02: int t = array[l]; int i = l; int j = u+1;
03: while(true) {
05:  do { i++; } while ( i <= u && array[i] < t );
06:  do { j--; } while ( array[j] > t );
07:  if ( i > j ) break;
08:  this->swapint( array[i], array[j] ); }
10: this->swapint( array[l], array[j] );
11: this->qsort3( array, l, j-1);
12: this->qsort3( array, j+1, u);
```

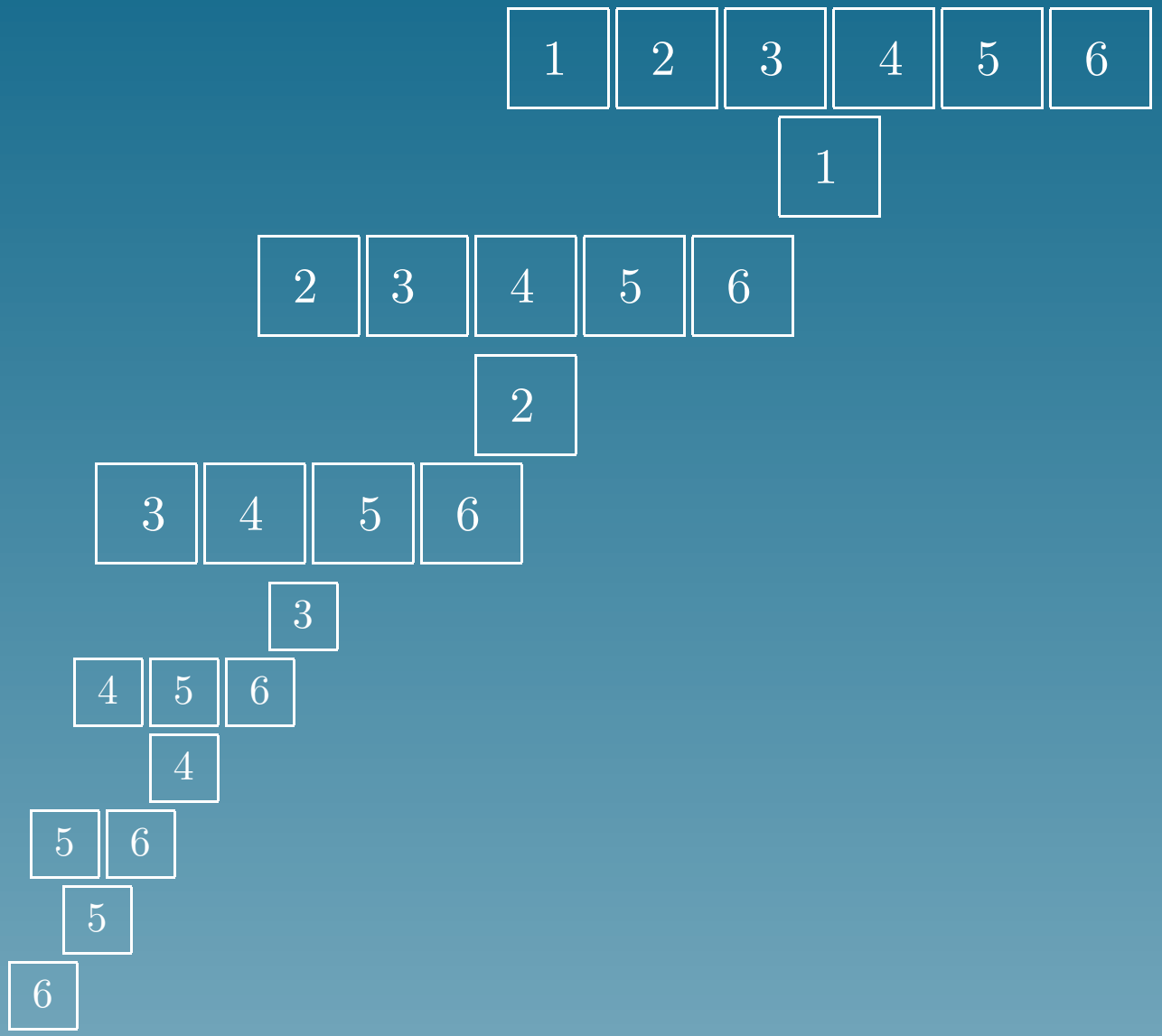
QuickSort

```
01: if (l >= u) return;
02: int t = array[l]; int i = l; int j = u+1;
03: while(true) {
05:  do { i++; } while ( i <= u && array[i] < t );
06:  do { j--; } while ( array[j] > t );
07:  if ( i > j ) break;
08:  this->swapint( array[i], array[j] ); }
10: this->swapint( array[l], array[j] );
11: this->qsort3( array, l, j-1);
12: this->qsort3( array, j+1, u);
```

```
30.000 gleiche Elemente:   0m0.144s
30.000 vorsortierte Elemente: 0m8.760s
```

QuickSort

Was ist da passiert?


1 2 3 4 5 6
1
2 3 4 5 6
2
3 4 5 6
3
4 5 6
4
5 6
5
6

QuickSort

```
01: int sw = this->randint(l, u);
02: int temp = array[sw]; array[sw] = array[l]; array[l] = temp;
03: int t = array[l]; int i = l; int j = u+1;
04: while (true) {
05:  do { i++; } while (i <= u && array[i] < t);
06:  do { j--; } while (array[j] > t);
07:  if (i > j) break;
08:  temp = array[i]; array[i] = array[j]; array[j] = temp;
09: }
10: this->swapint(array[l], array[j]);
11 :this->qsort4(array, l, j-1);
12: this->qsort4(array, j+1, u);
```

QuickSort

```
30.000 Zahlen vorsortiert:
0m0.418s
```

QuickSort

```
30.000 Zahlen vorsortiert:
0m0.418s


30.000 gleiche Zahlen:
0m0.108s
```

QuickSort

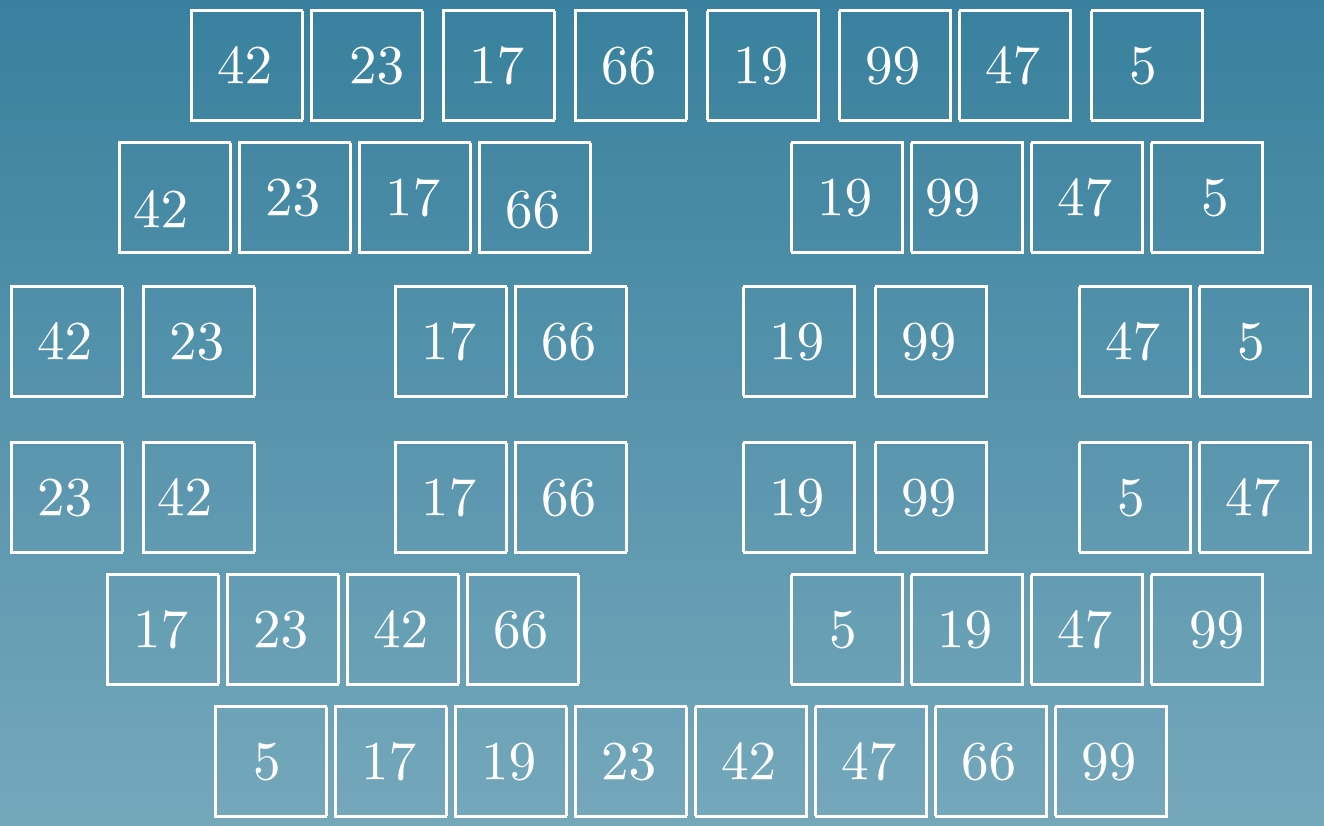
```
30.000 Zahlen vorsortiert:
0m0.418s


30.000 gleiche Zahlen:
0m0.108s


30.000 zufaellige Zahlen
0m0.171s
```

MergeSort

Funktionsweise:



MergeSort

```
00:Msort1::RekMergeSort( int my_array[], int Lo, int Hi ) {
01:  if (Lo < Hi) {
02:    int split = (Lo + Hi + 1)/2;
03:    this->RekMergeSort(my_array, Lo, split-1);
04:    this->RekMergeSort(my_array, split, Hi);
05:    int alen = (Hi-Lo+1); int *temp_arr = new int[alen];
06:    for (int i = 0, j = Lo,  k = split; i < alen; i++)
07:      if ((k > Hi)||(j < split) && (my_array[j] < my_array[k])){
08:        temp_arr[i] = my_array[j]; j++; }
09:      else { temp_arr[i] = my_array[k]; k++; }
10:    for (int i=0; i < alen; i++) my_array[Lo+i] = temp_arr[i];
11:  } }
```

MergeSort

```
30.000 zufaellige Zahlen:
0m0.152s
```

MergeSort

```
30.000 zufaellige Zahlen:
0m0.152s


30.000 gleiche Zahlen:
0m0.129s
```

MergeSort

```
30.000 zufaellige Zahlen:
0m0.152s


30.000 gleiche Zahlen:
0m0.129s


30.000 vorsortierte Zahlen:
0m0.191s
```

MergeSort

```
01: if (Lo >= Hi) return;
02: if ((Hi - Lo) <= 2){ int mid = (Hi+Lo)/2;
03:   if (array[Lo]  > array[mid]) this->swapint(array[Lo], array[mid]);
04:   if (array[mid] > array[Hi])  this->swapint(array[mid], array[Hi]);
05:   if (array[Lo]  > array[mid]) this->swapint(array[Lo], array[mid]); }
06: else{ int split = (Lo + Hi +1)/2;
07:         int len = Hi-Lo+1;
08:         this->RekMergeSort(array, Lo, split-1);
09:         this->RekMergeSort(array, split, Hi);
10:         for (int i=0, j=Lo, k=split; i < len; i++){
11:           if ((k > Hi)||(j < split) && (array[j] < array[k])){
12:             this->temp_arr[i] = array[j]; j++; }
13:           else{ this->temp_arr[i] = array[k]; k++; }
14:    }
15:   for (int i=0; i < len; i++) array[Lo+i] = this->temp_arr[i]; } }
```

MergeSort

```
30.000 zufaellige Zahlen
0m0.118s
```

MergeSort

```
30.000 zufaellige Zahlen
0m0.118s


30.000 gleiche Zahlen
0m0.089s
```

MergeSort

```
30.000 zufaellige Zahlen
0m0.118s


30.000 gleiche Zahlen
0m0.089s


30.000 vorsortierte Zahlen
0m0.171s
```

Literatur

Jon Bentley, Programming Pearls
Longman

Literatur

Jon Bentley, Programming Pearls
Longman

Donald Knuth, The Art of Computer Programming,
Volume 3
Addison-Wesley

Literatur

Jon Bentley, Programming Pearls
Longman

Donald Knuth, The Art of Computer Programming,
Volume 3
Addison-Wesley

Robert Sedgewick, Algorithmen
Addison-Wesley

Mehr Literatur

Scott Meyers, Effective C++
Addison-Wesley