

Sorting an array

Johan Montelius

KTH

VT26

why, when and how

- Working with sorted data is often more efficient (searching).
- Sort data and keep sorted data sorted.
- ... todays lecture.

why, when and how

- Working with sorted data is often more efficient (searching).
- Sort data and keep sorted data sorted.
- ... todays lecture.

why, when and how

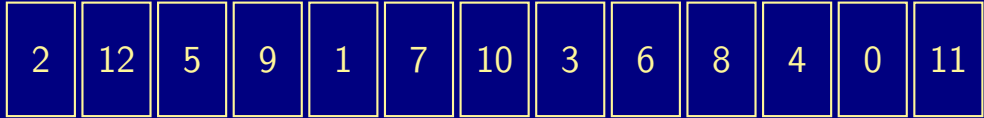
- Working with sorted data is often more efficient (searching).
- Sort data and keep sorted data sorted.
- ... todays lecture.

why, when and how

- Working with sorted data is often more efficient (searching).
- Sort data and keep sorted data sorted.
- ... todays lecture.

selection sort

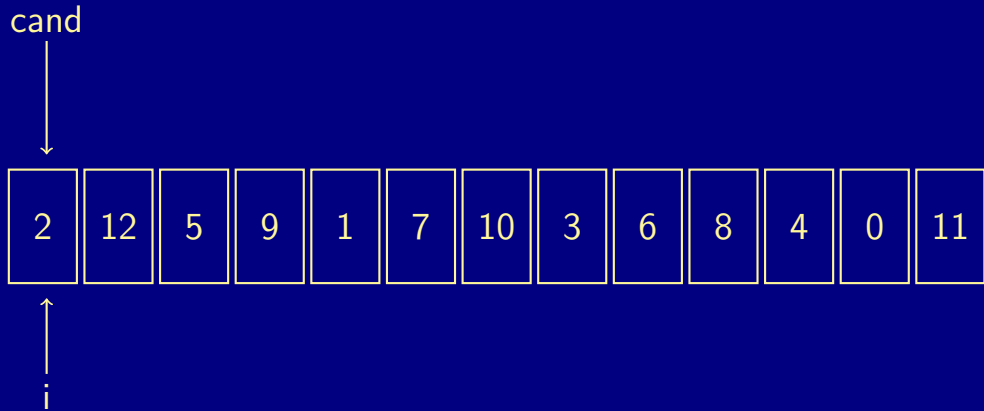
selection sort



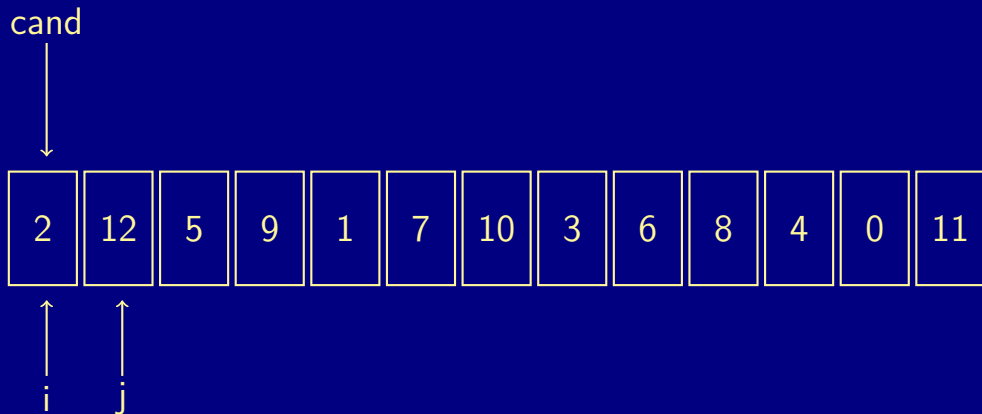
selection sort



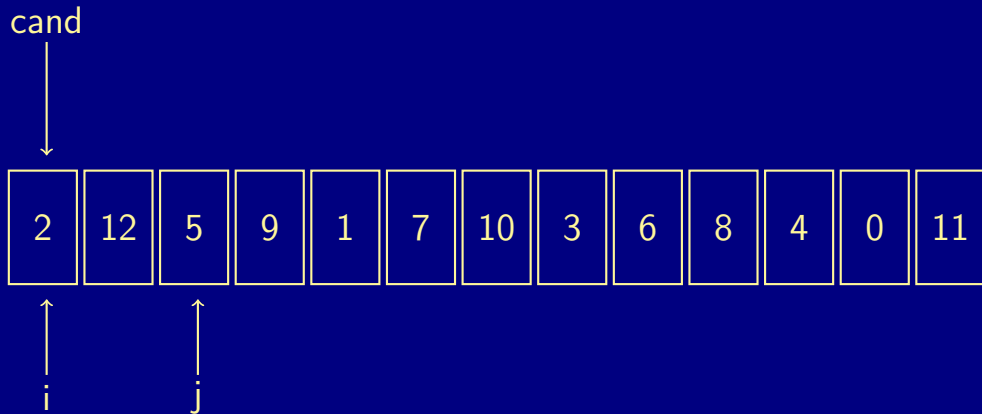
selection sort



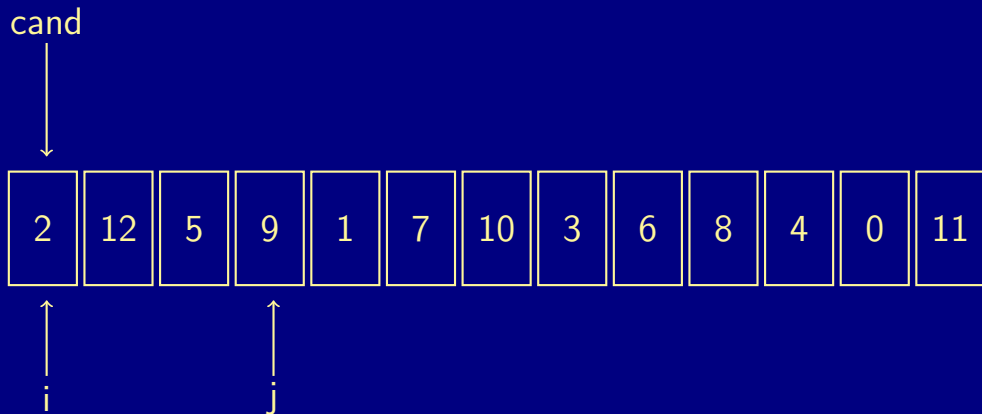
selection sort



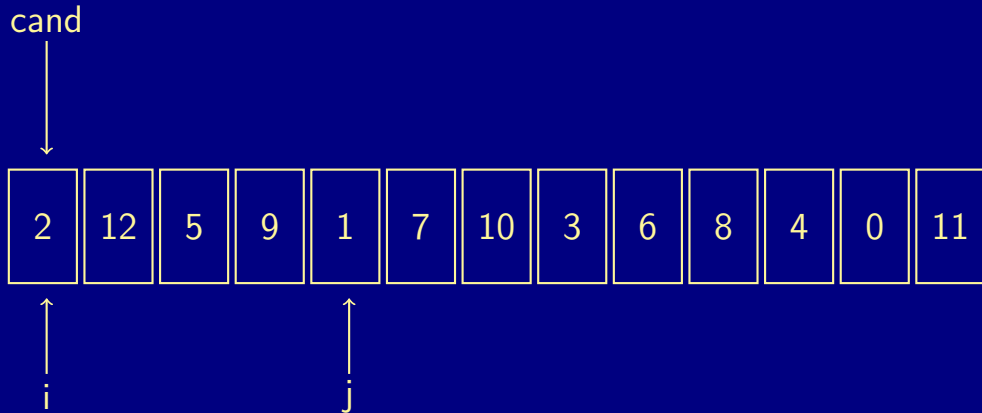
selection sort



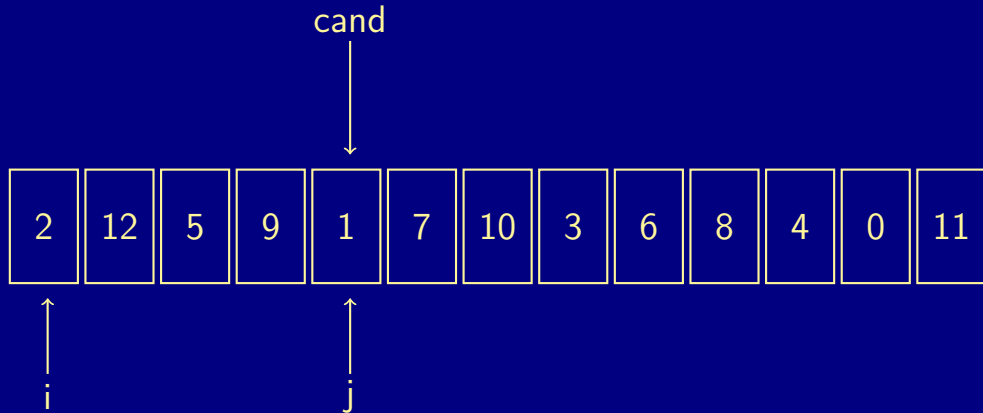
selection sort



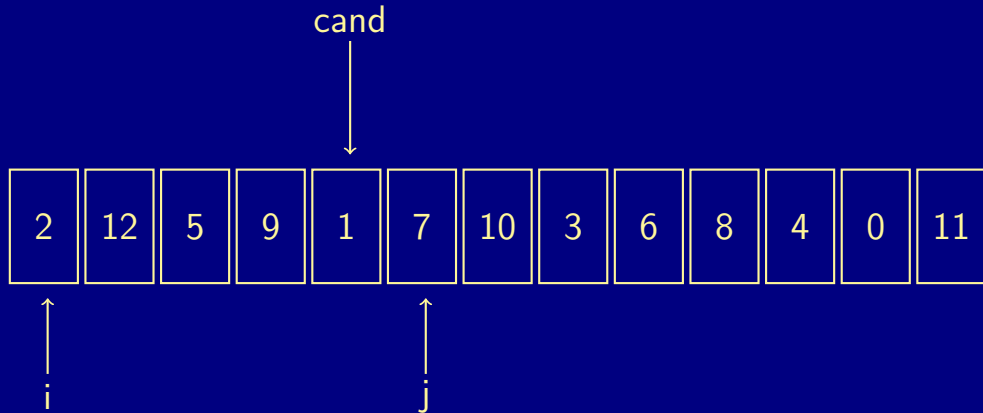
selection sort



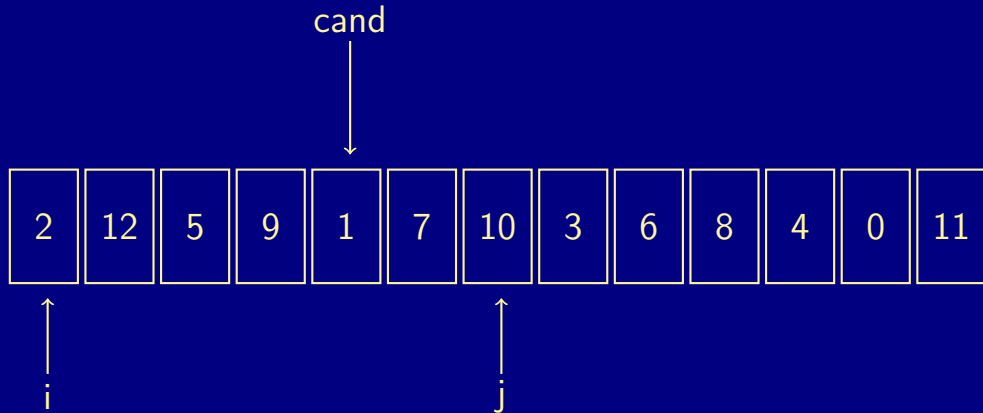
selection sort



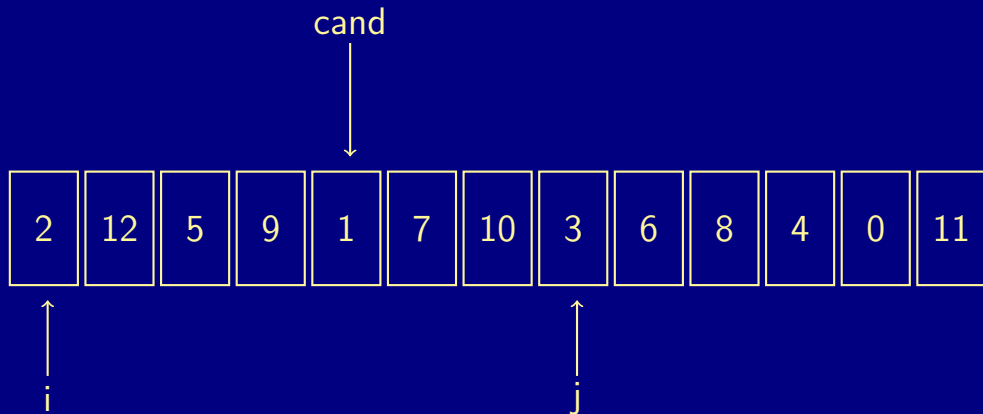
selection sort



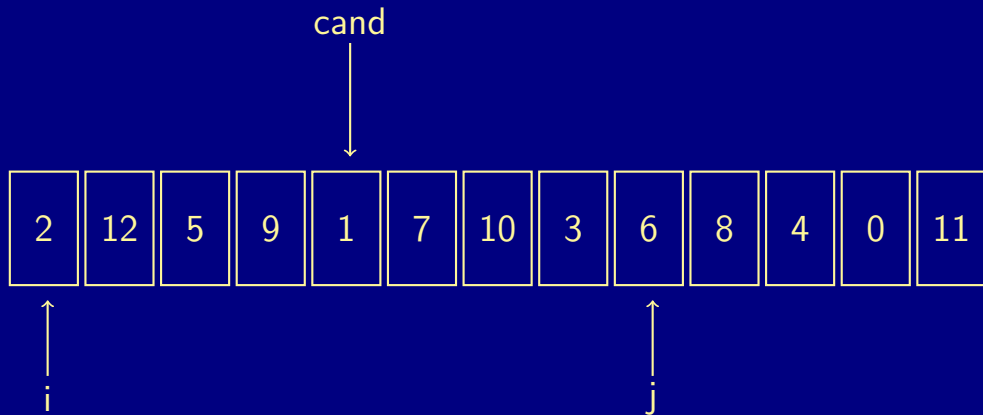
selection sort



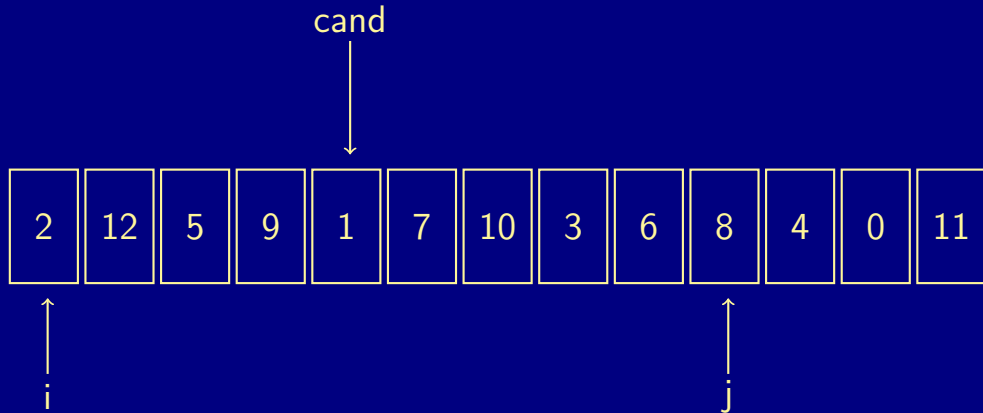
selection sort



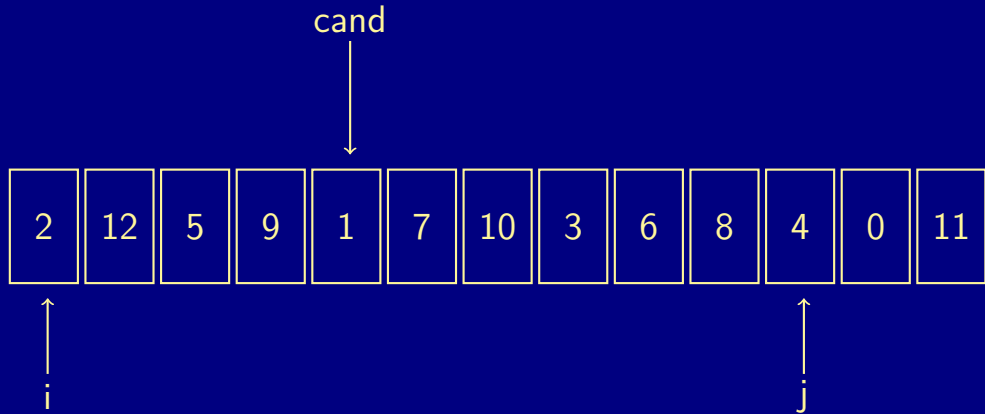
selection sort



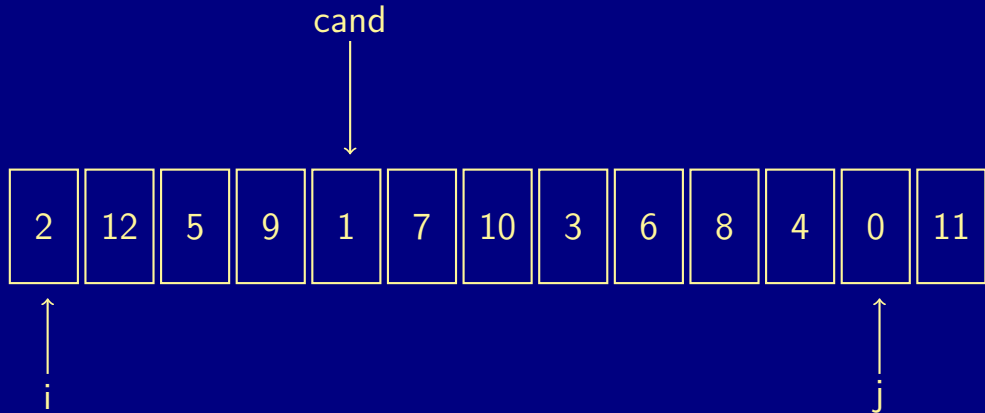
selection sort



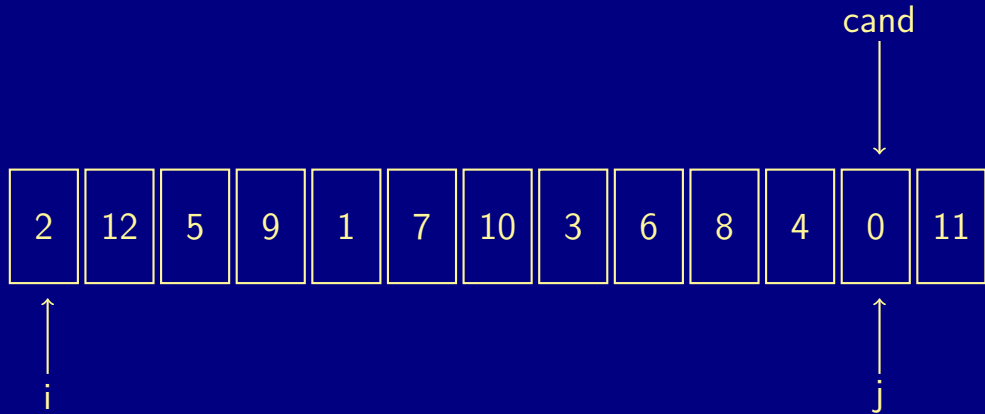
selection sort



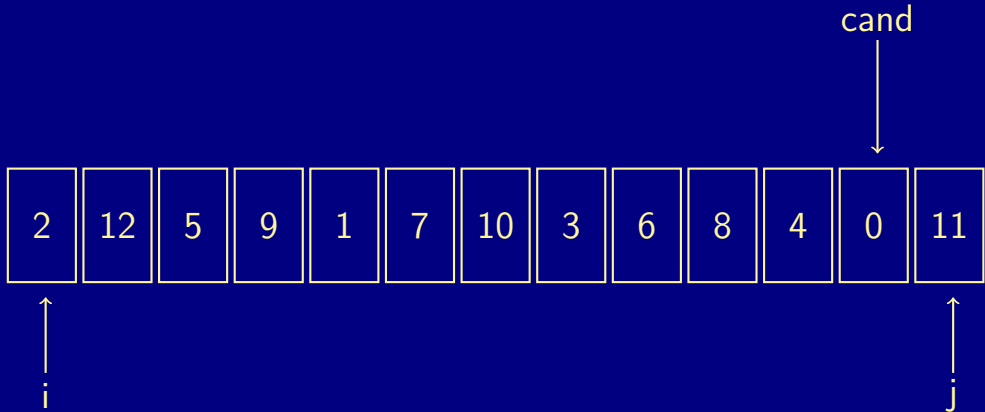
selection sort



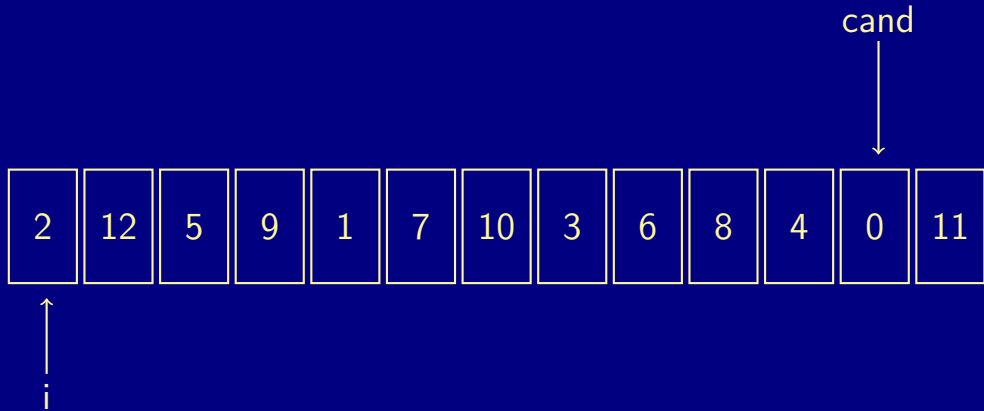
selection sort



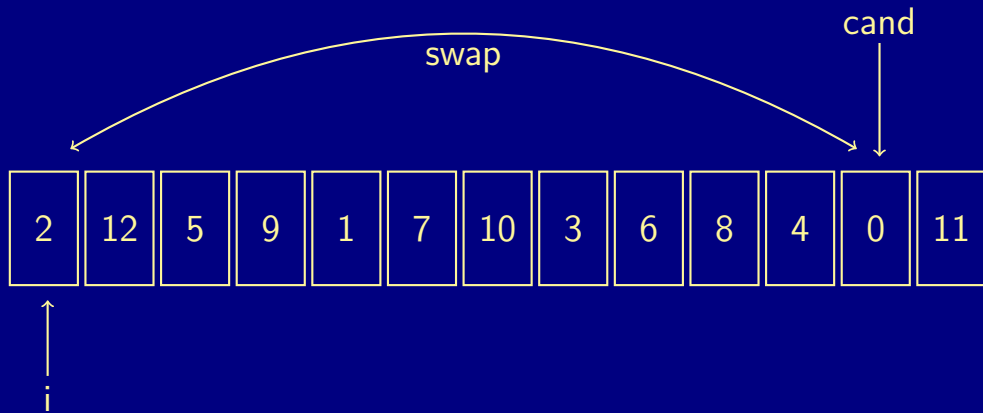
selection sort



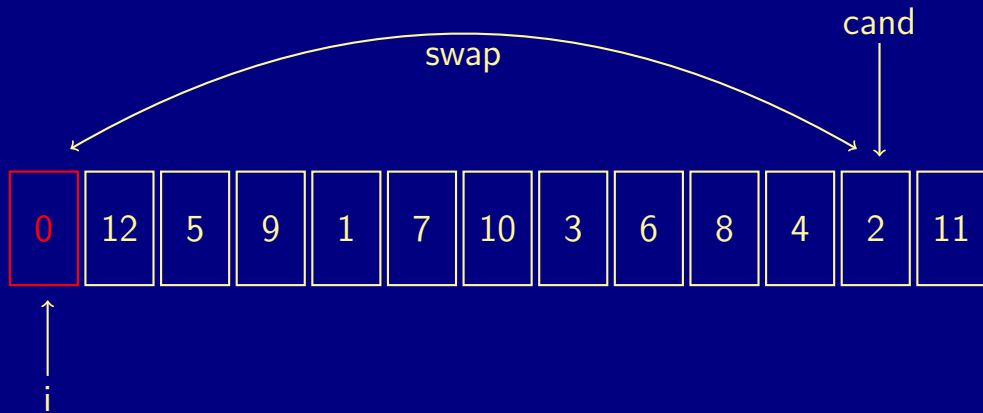
selection sort



selection sort



selection sort



selection sort



selection sort



selection sort



selection sort



selection sort



selection sort



selection sort

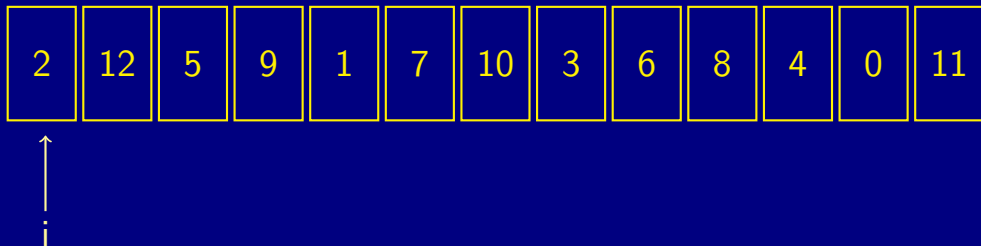
- How many compare operations?
- How many swap operations?
- Any special cases?

insertion sort

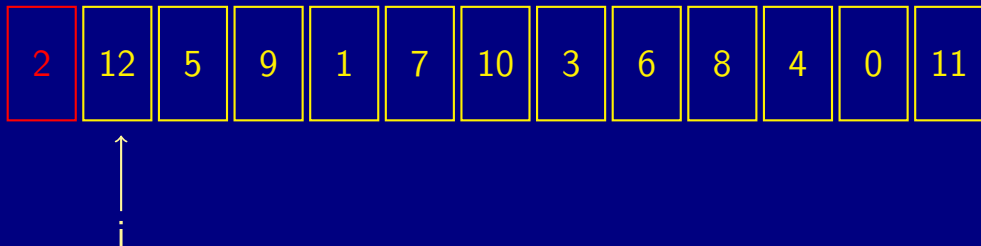
insertion sort



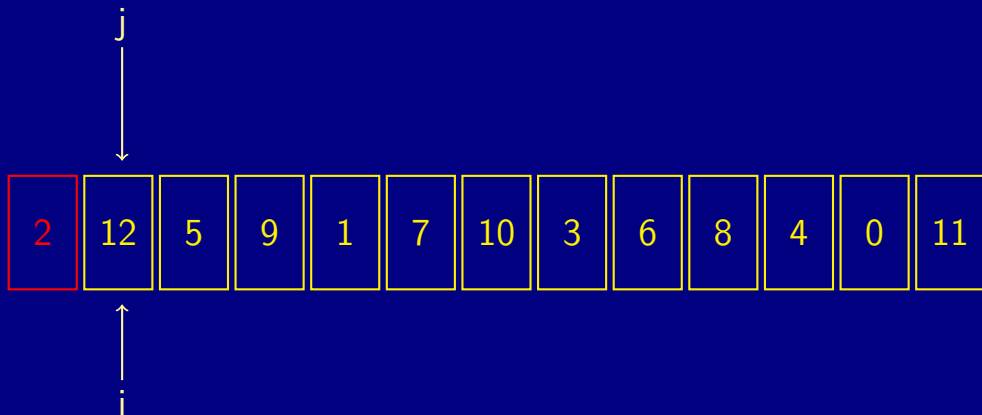
insertion sort



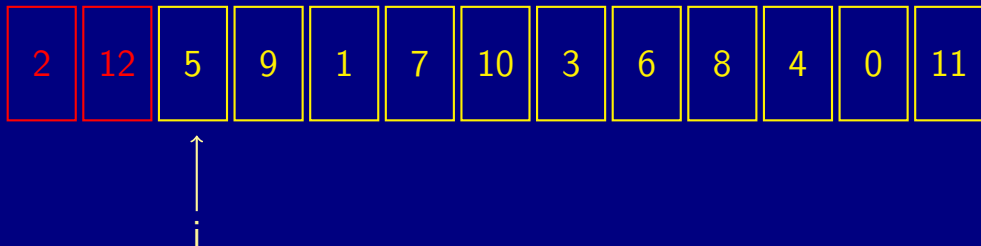
insertion sort



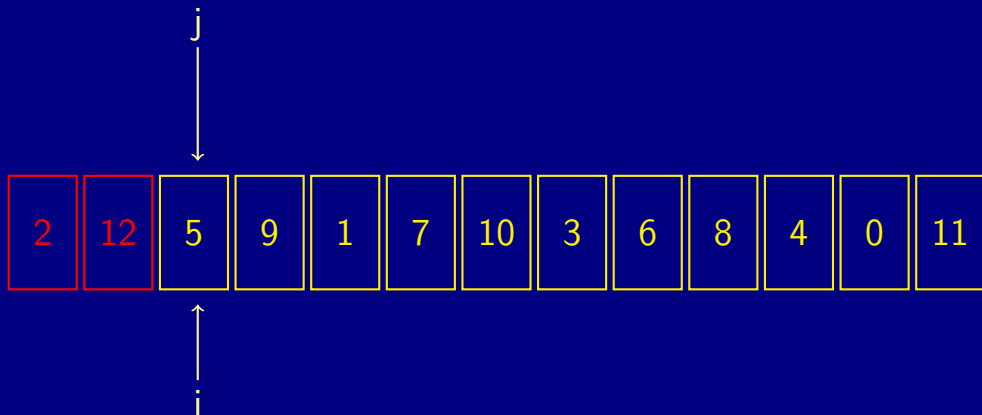
insertion sort



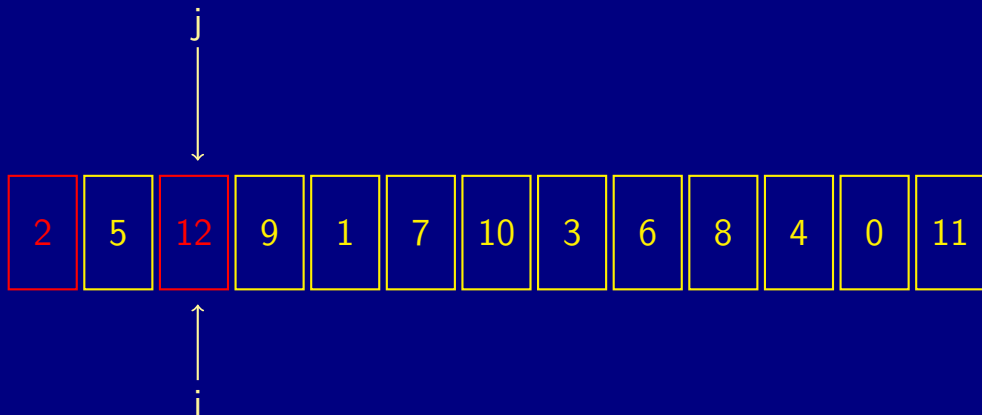
insertion sort



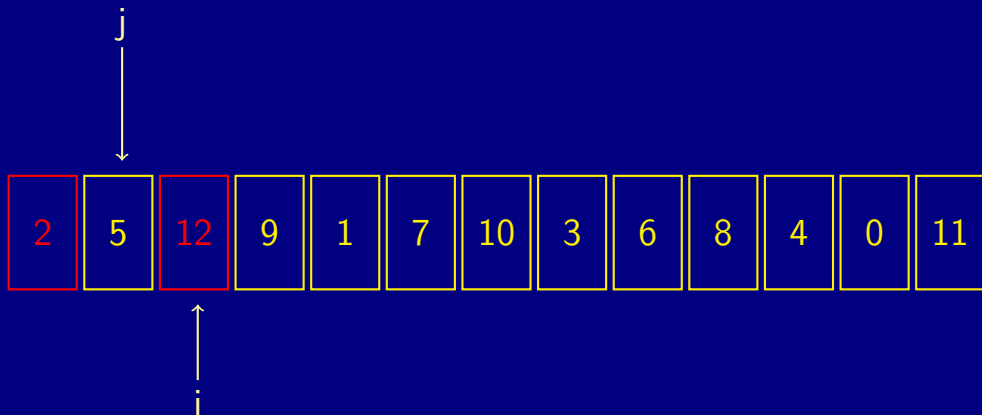
insertion sort



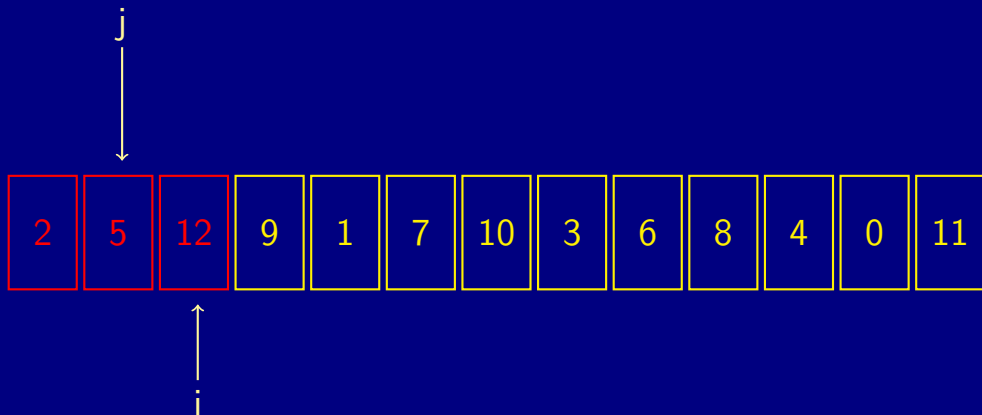
insertion sort



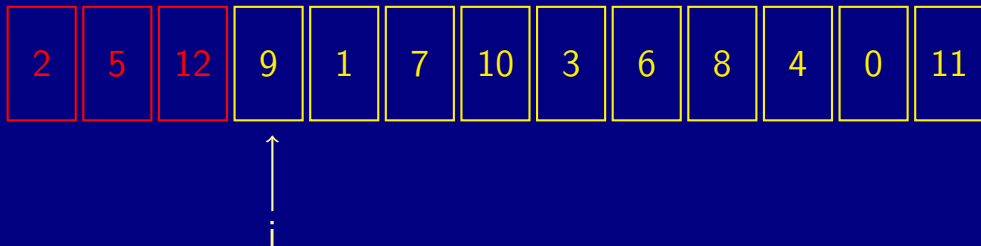
insertion sort



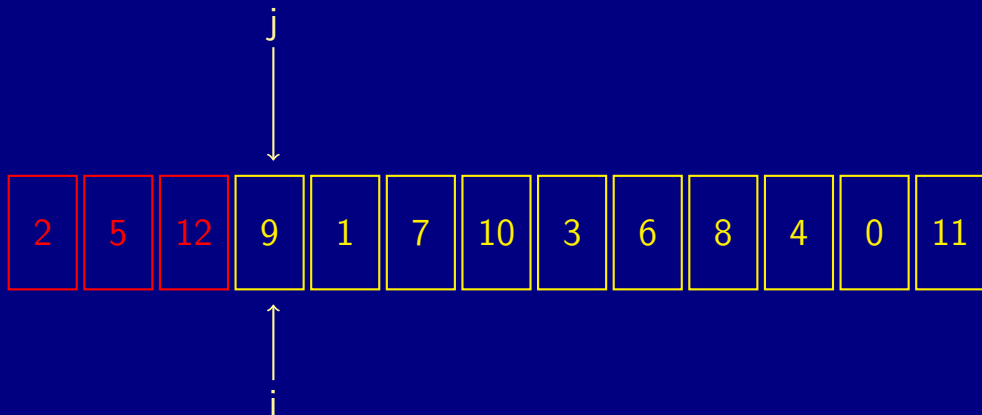
insertion sort



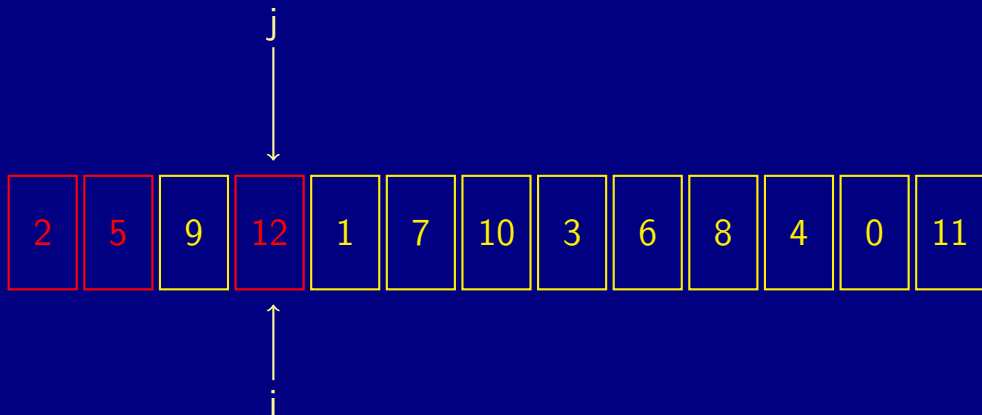
insertion sort



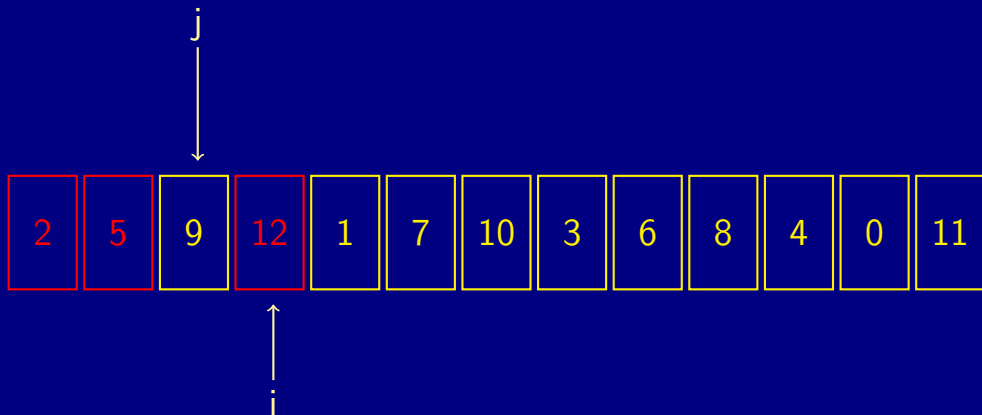
insertion sort



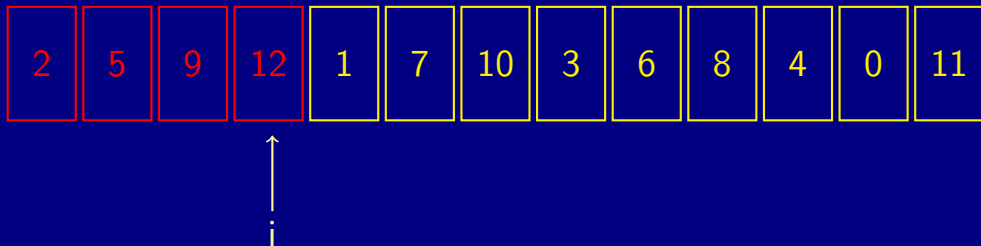
insertion sort



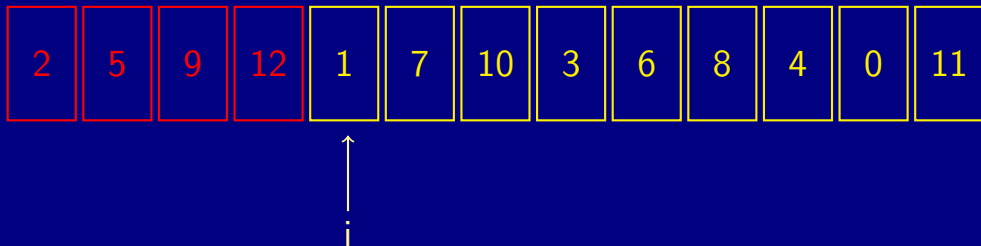
insertion sort



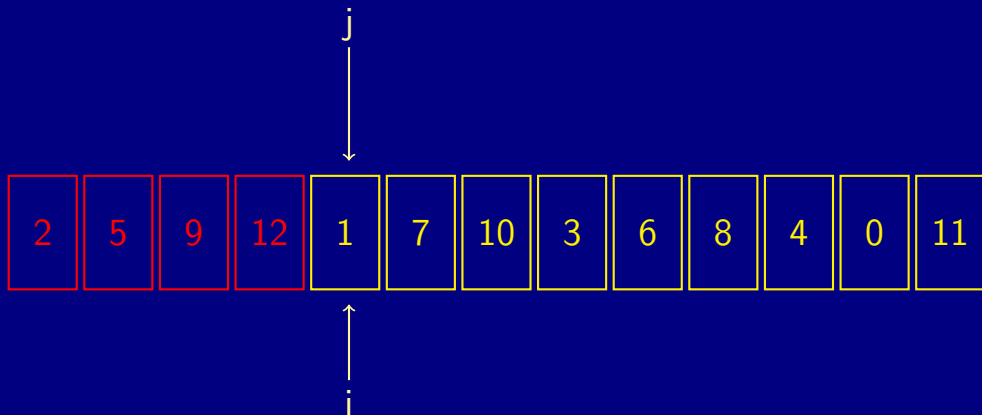
insertion sort



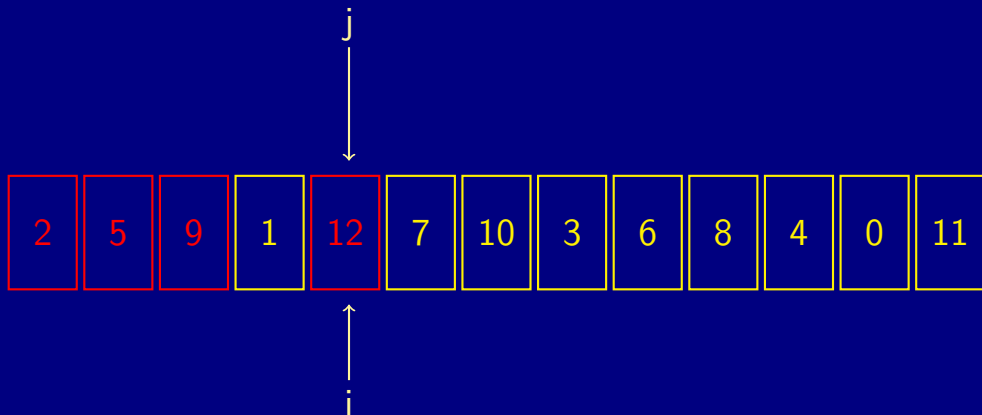
insertion sort



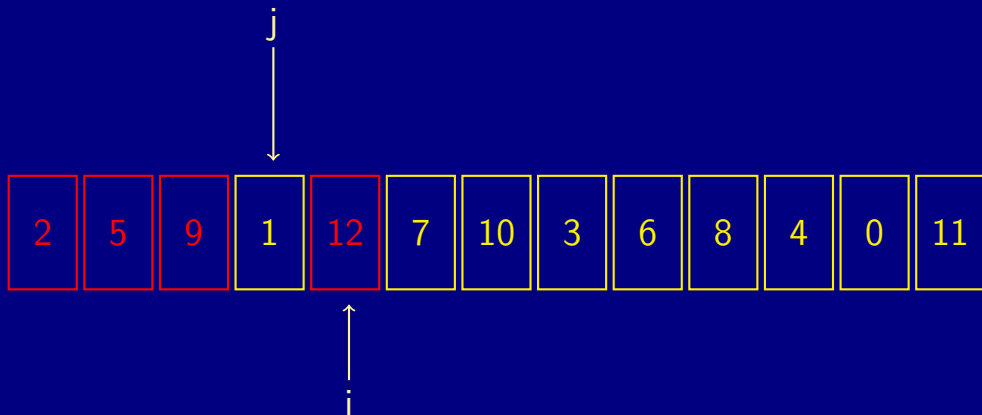
insertion sort



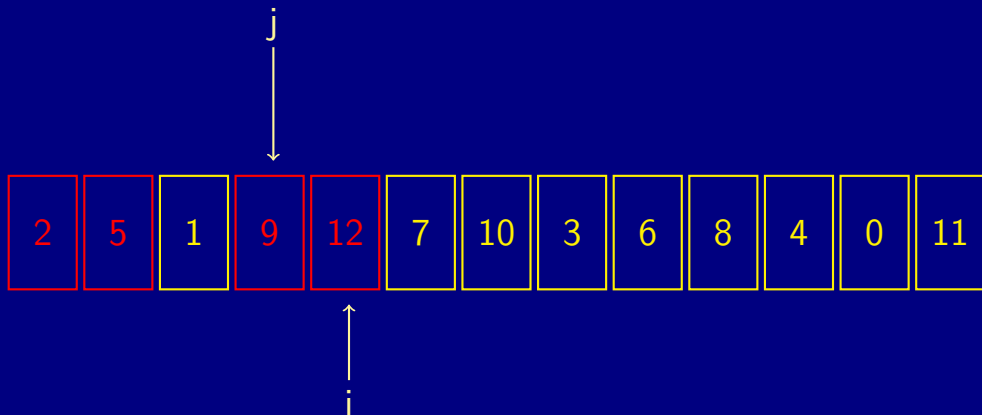
insertion sort



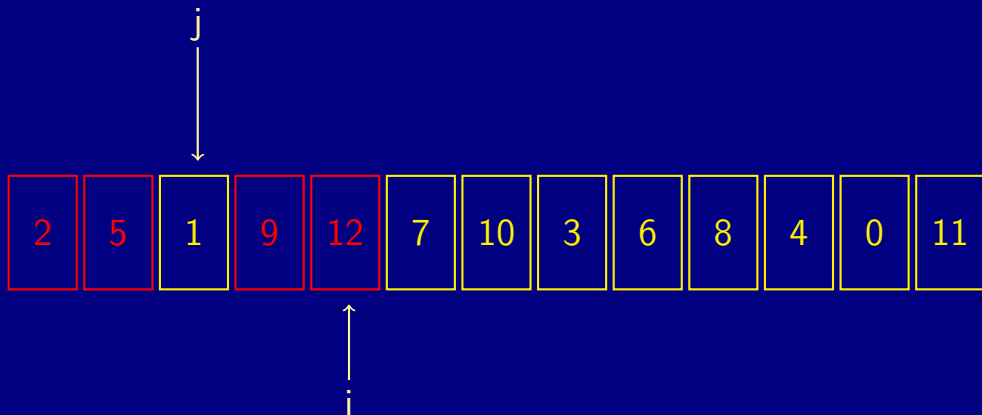
insertion sort



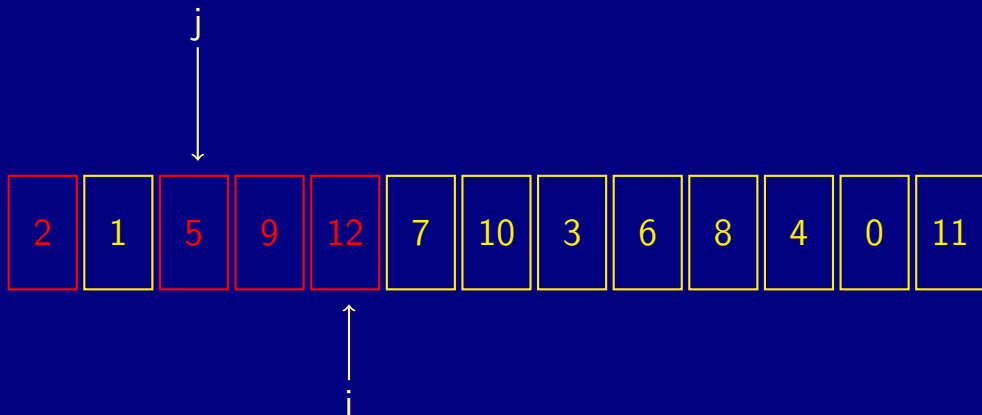
insertion sort



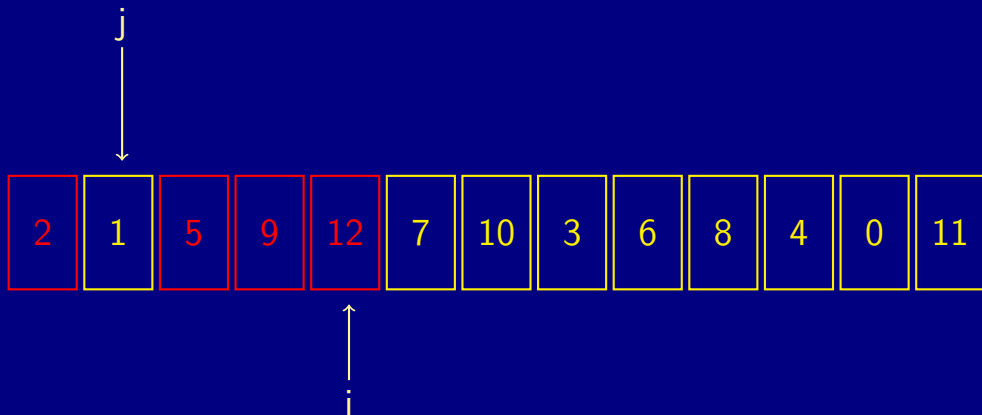
insertion sort



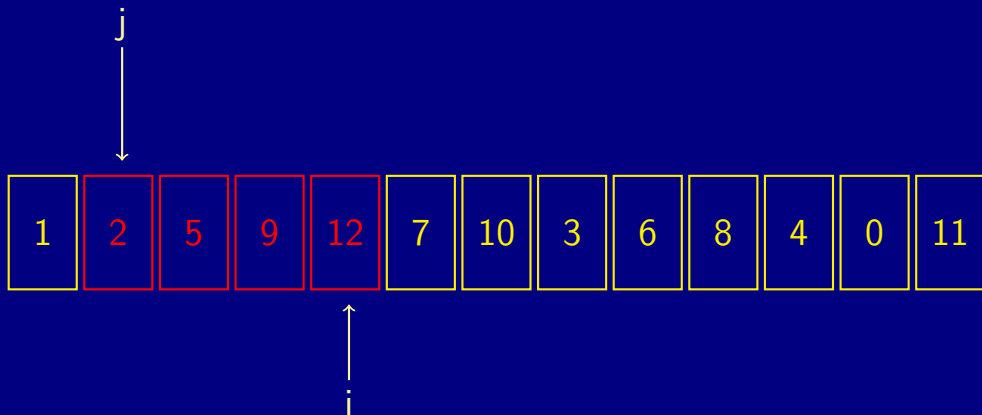
insertion sort



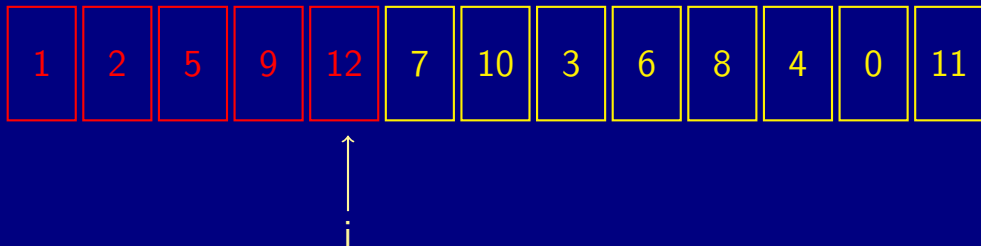
insertion sort



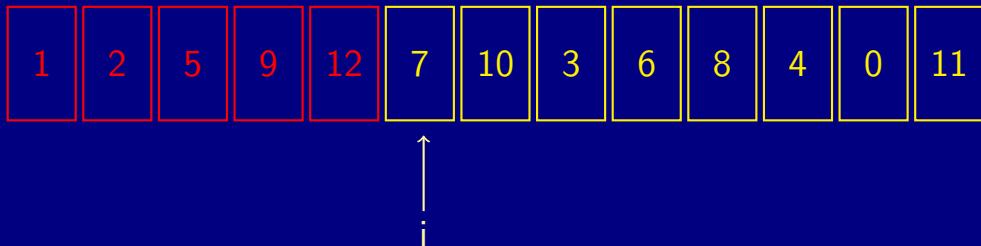
insertion sort



insertion sort



insertion sort



insertion sort

- How many compare operations?
- How many swap operations?
- Any special cases?

Can we do better?

Divide and conquer.

- divide array in two
- sort first part
- sort second part
- merge the two parts

Can we do better?

Divide and conquer.

- divide array in two
- sort first part
- sort second part
- merge the two parts

Can we do better?

Divide and conquer.

- divide array in two
- sort first part
- sort second part
- merge the two parts

Can we do better?

Divide and conquer.

- divide array in two
- sort first part
- sort second part
- merge the two parts

Can we do better?

Divide and conquer.

- divide array in two
- sort first part
- sort second part
- merge the two parts

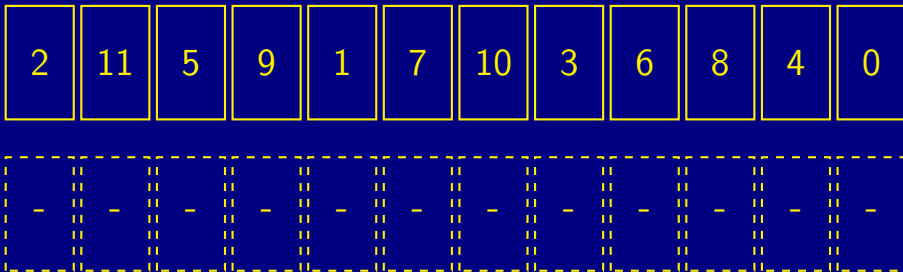
Can we do better?

Divide and conquer.

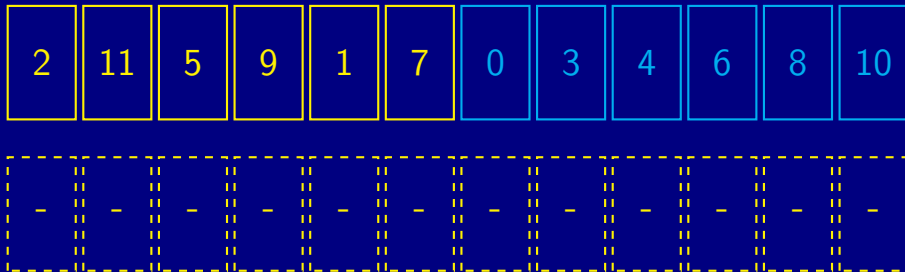
- divide array in two
- sort first part
- sort second part
- merge the two parts

merge-sort

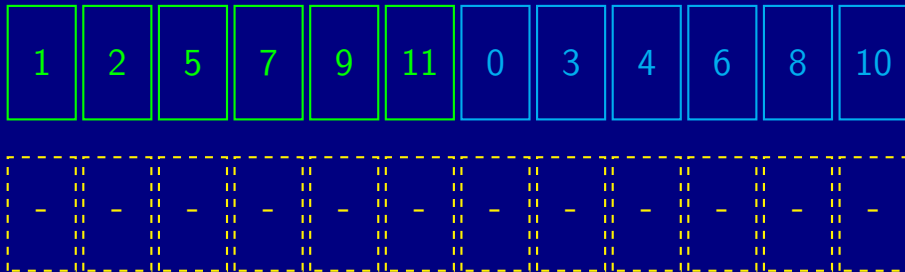
merge-sort



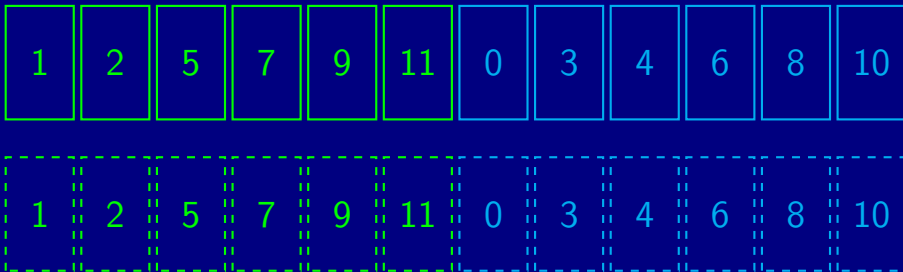
merge-sort



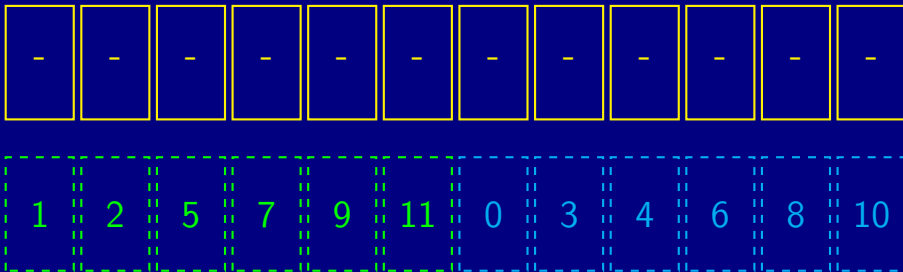
merge-sort



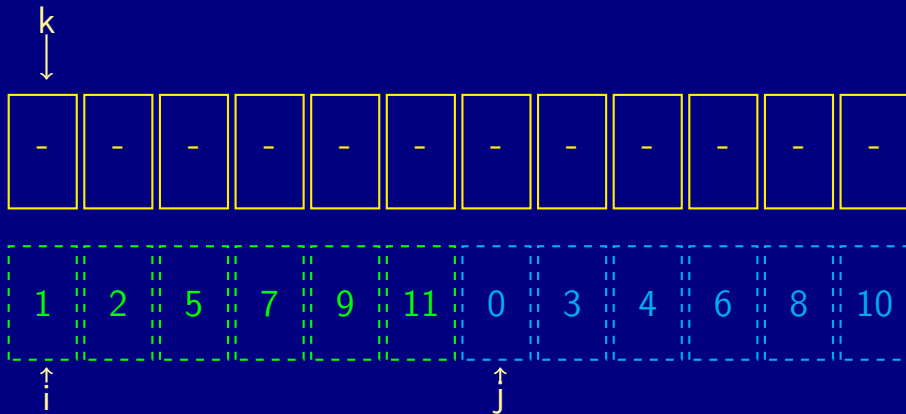
merge-sort



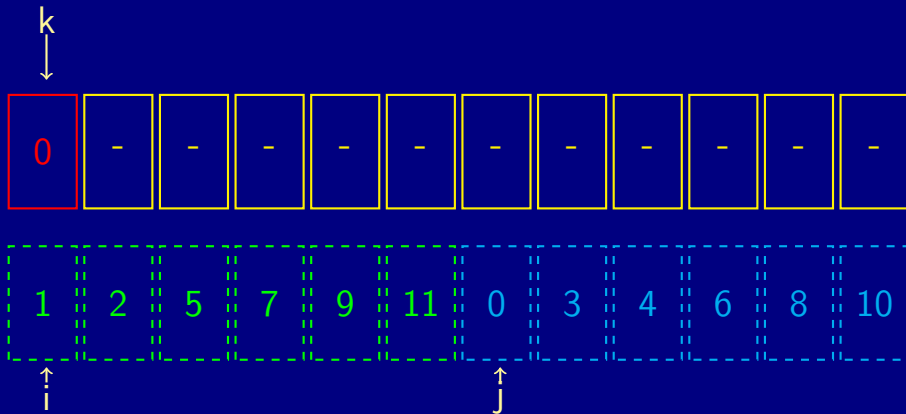
merge-sort



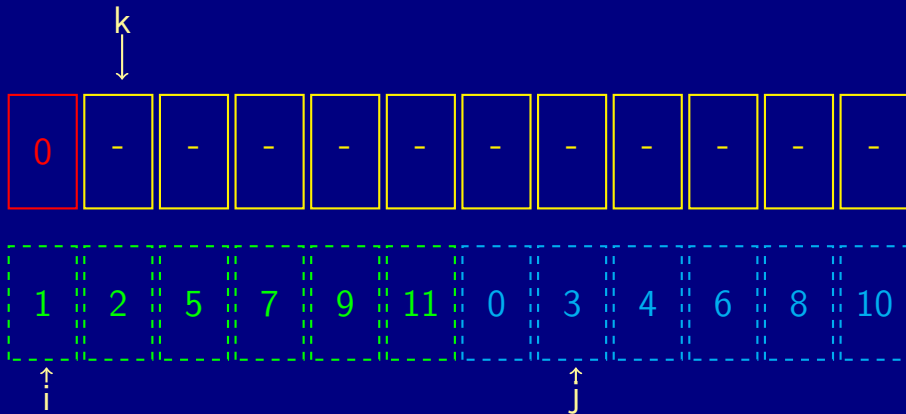
merge-sort



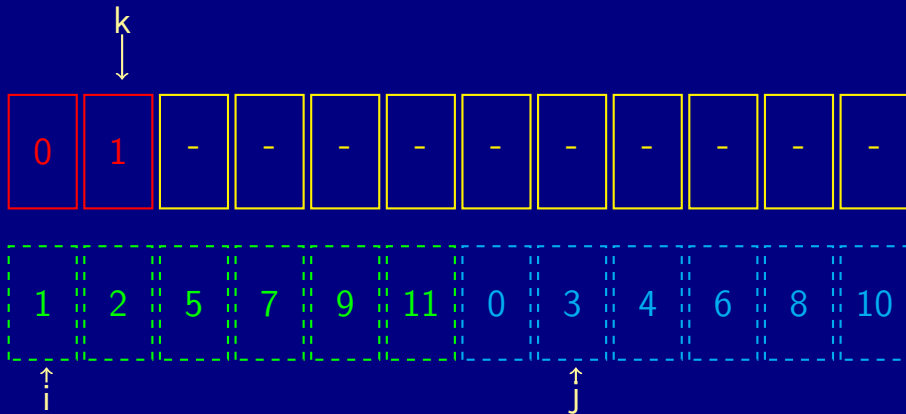
merge-sort



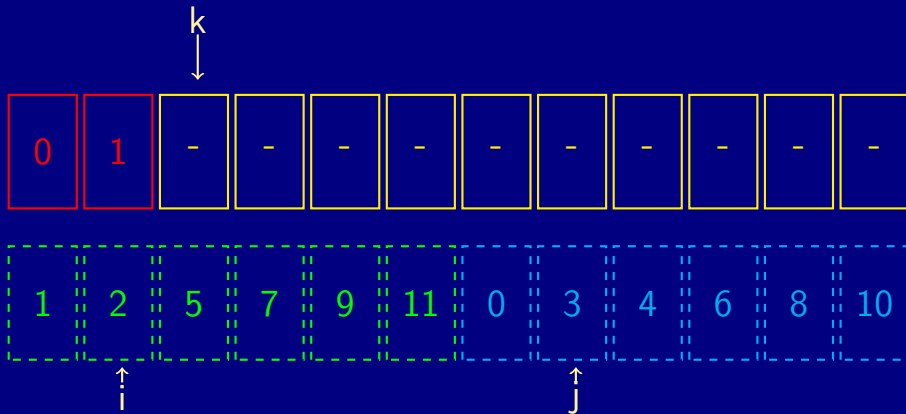
merge-sort



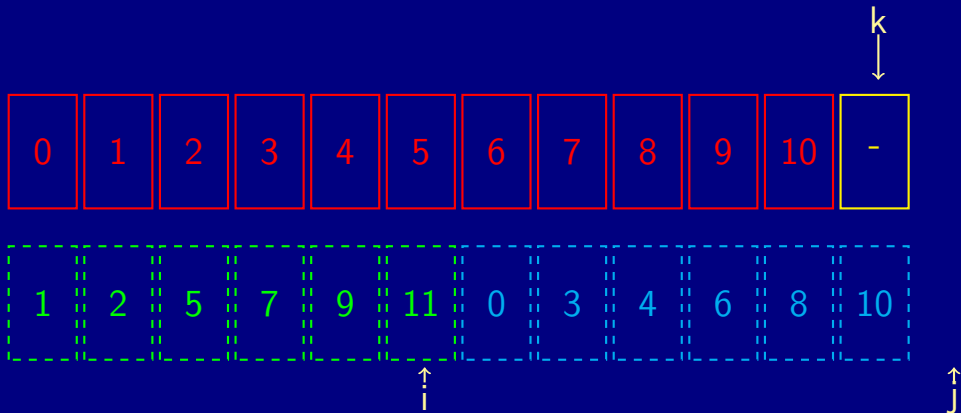
merge-sort



merge-sort



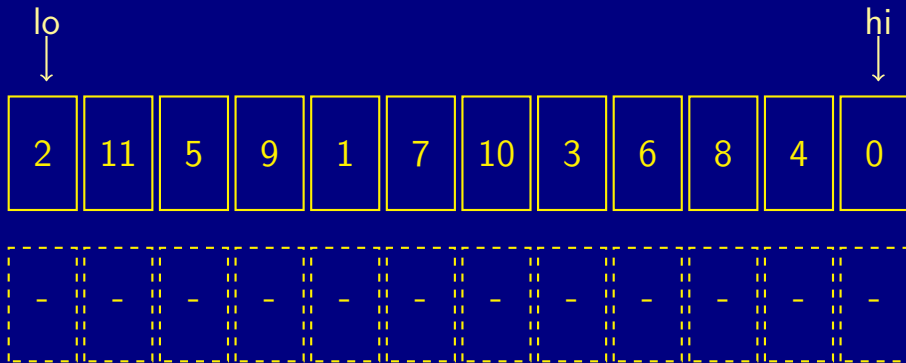
merge-sort



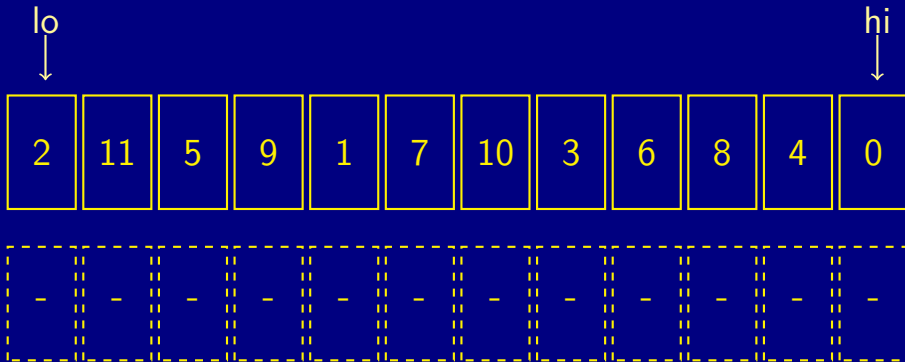
merge-sort



one small problem

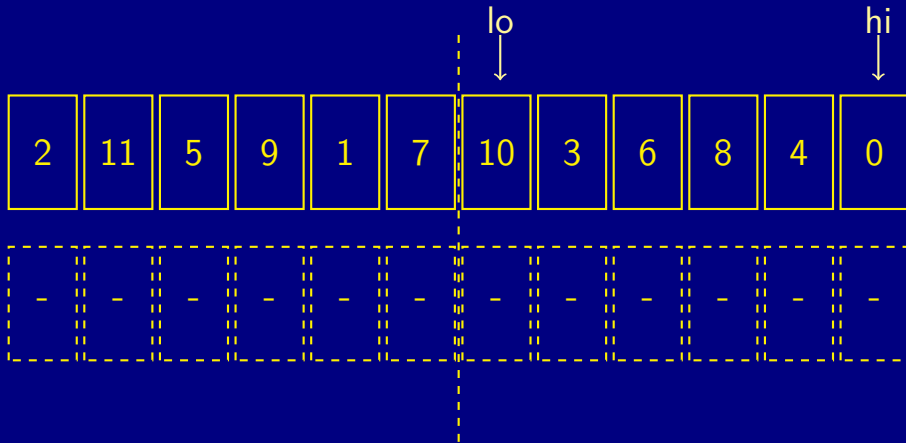


one small problem



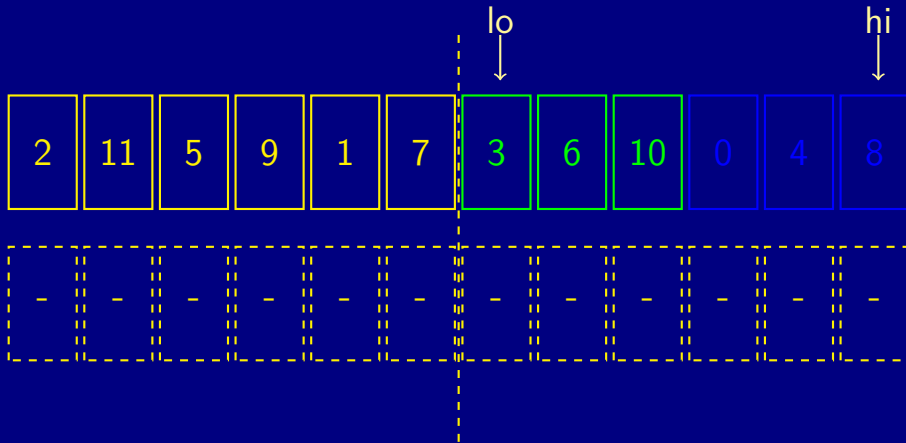
How do we sort the two smaller arrays?

one small problem



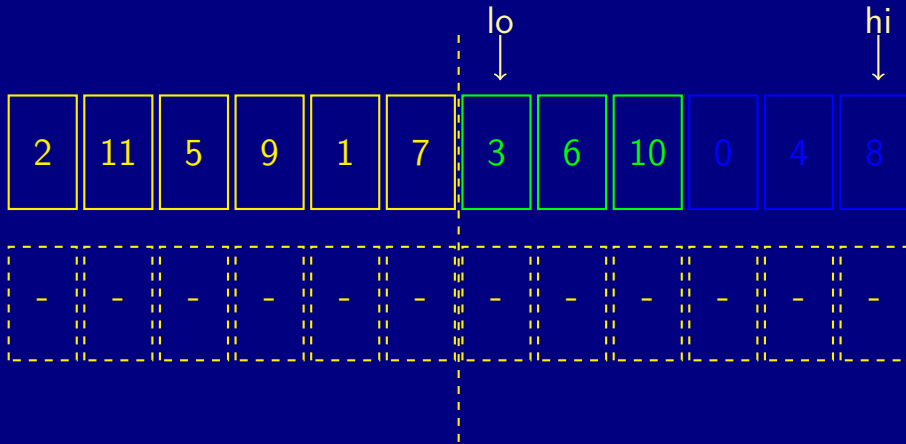
How do we sort the two smaller arrays?

one small problem



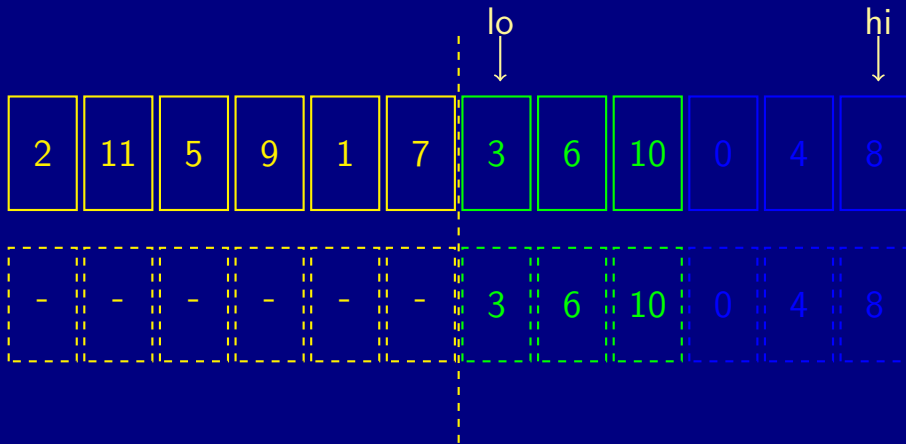
How do we sort the two smaller arrays?

one small problem



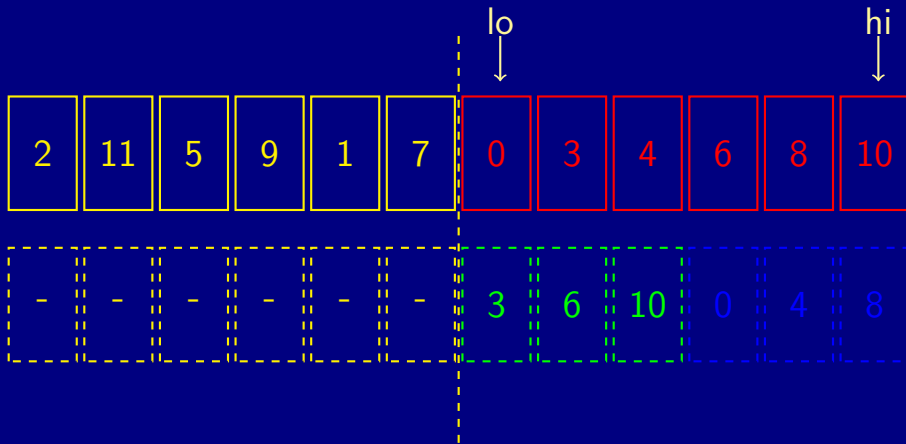
How do we sort the two smaller arrays?

one small problem



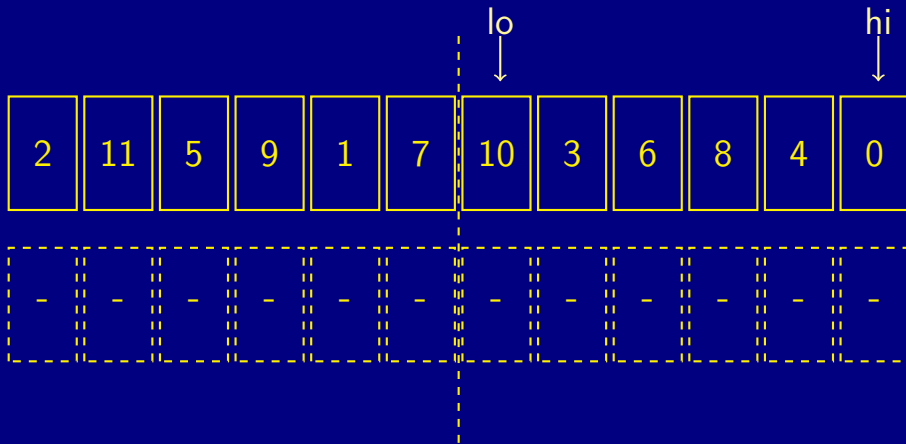
How do we sort the two smaller arrays?

one small problem

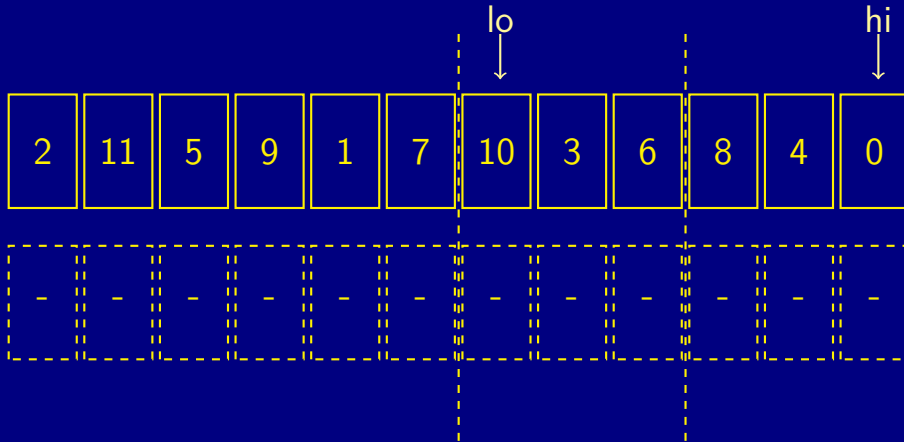


How do we sort the two smaller arrays?

.... one small problem

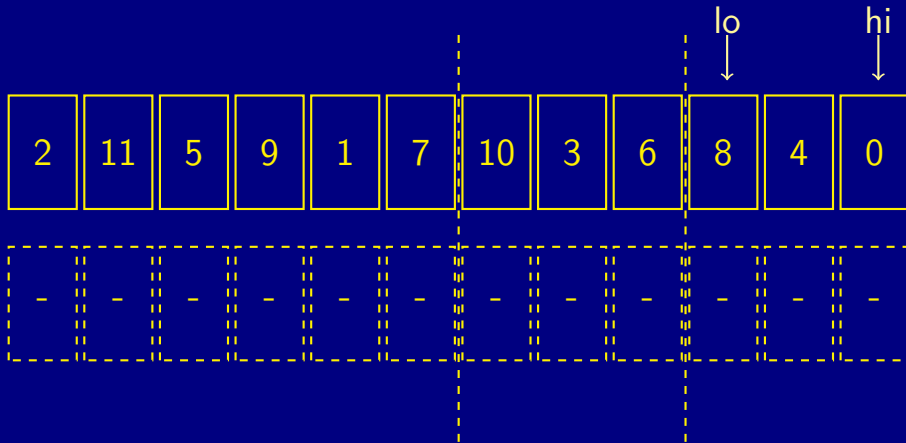


.... one small problem



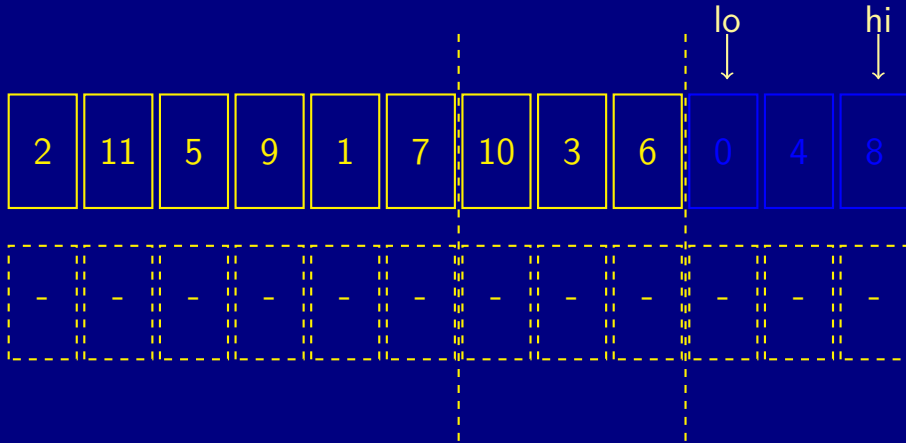
.... how do we sort the two smaller arrays?

.... one small problem



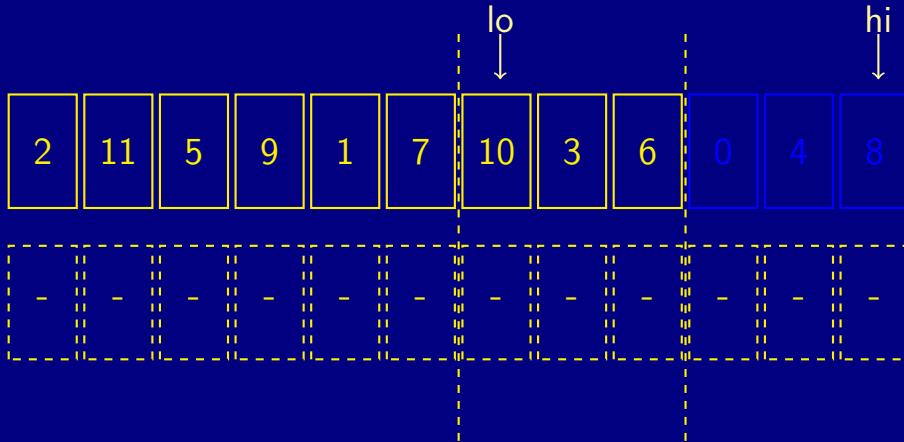
.... how do we sort the two smaller arrays?

.... one small problem



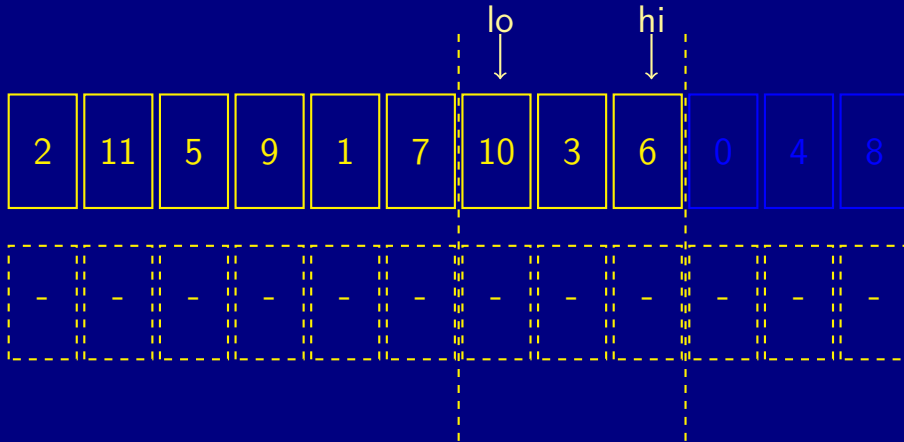
.... how do we sort the two smaller arrays?

.... one small problem



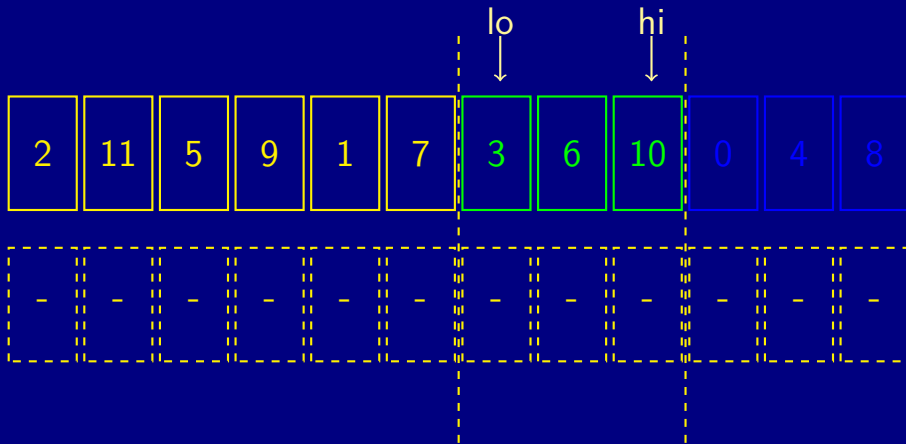
.... how do we sort the two smaller arrays?

.... one small problem



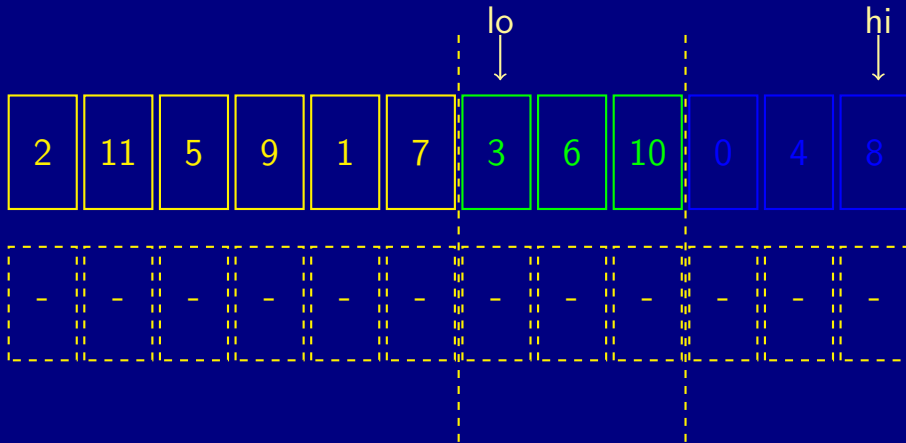
.... how do we sort the two smaller arrays?

.... one small problem



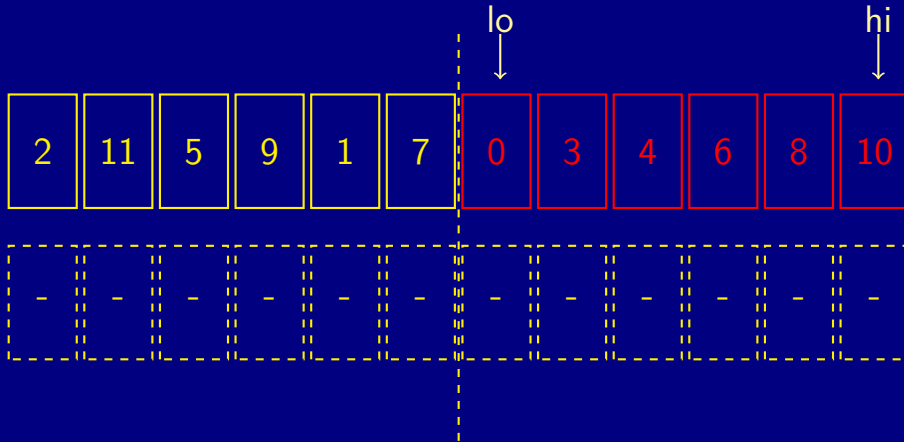
.... how do we sort the two smaller arrays?

.... one small problem



.... how do we sort the two smaller arrays?

.... one small problem



.... how do we sort the two smaller arrays?

Recursion

Recursion

- If you can sort an array with up to n elements

Recursion

- If you can sort an array with up to n elements
- and can merge two sorted arrays

Recursion

- If you can sort an array with up to n elements
- and can merge two sorted arrays
- then you can sort an array of up to $2 \times n$ elements.

Recursion

- If you can sort an array with up to n elements
- and can merge two sorted arrays
- then you can sort an array of up to $2 \times n$ elements.
- Since an array with only one element is sorted

Recursion

- If you can sort an array with up to n elements
- and can merge two sorted arrays
- then you can sort an array of up to $2 \times n$ elements.
- Since an array with only one element is sorted
- then you can sort an array of arbitrary length.

sum of elements

```
int sum(int *arr, int length) {  
    int sum = 0;  
    for(int i = 0; i < length, i++)  
        sum += arr[i];  
    return sum;  
}
```

sum of elements

```
int sum(int *arr, int length) {  
    return sum(arr, 0, length-1);  
}
```

```
sum(int *arr, int from, int to) {  
    if (from == to )  
        return arr[from];  
    else  
        return arr[from] + sum(arr, from+1, to);  
}
```

sum of elements

```
int sum(int *arr, int from, int to) {  
  
    if (from == to )  
        return arr[from];  
    else {  
        int mid = (from + to)/2;  
        return sum(arr, from, mid) + sum(arr, mid+1, to);  
    }  
}
```

binary search revisited

```
bool binary_search(int array[], int length, int key) {  
    int min = 0;  
    int max = length - 1;  
  
    while (true) {  
        int mid = (min+max)/2;  
        if (array[mid] == key) return true;  
        if ((array[mid] > key) & (mid > min))  
            { max = mid - 1; continue; }  
        if ((array[mid] < key) & (mid < max))  
            { min = mid + 1; continue; }  
        return false;  
    }  
}
```

binary search revisited

```
boolean search(int *array, int length, int key) {  
    search(array, key, 0, length - 1);  
}
```

binary search revisited

```
boolean search(int *array, int key, int min, int max) {  
    int mid = (min + max)/2;  
  
    if (array[mid] == key) return true;  
  
    if ((array[mid] > key) & (mid > min))  
        return search(array, key, min, mid - 1);  
    else if (mid < max)  
        return search(array, key, mid + 1, max);  
    else  
        return false;  
}
```

recursive procedures

Use with care.

recursive procedures

Use with care.

Every method call will push a frame on the stack.

recursive procedures

Use with care.

Every method call will push a frame on the stack.

The stack is not unlimited.