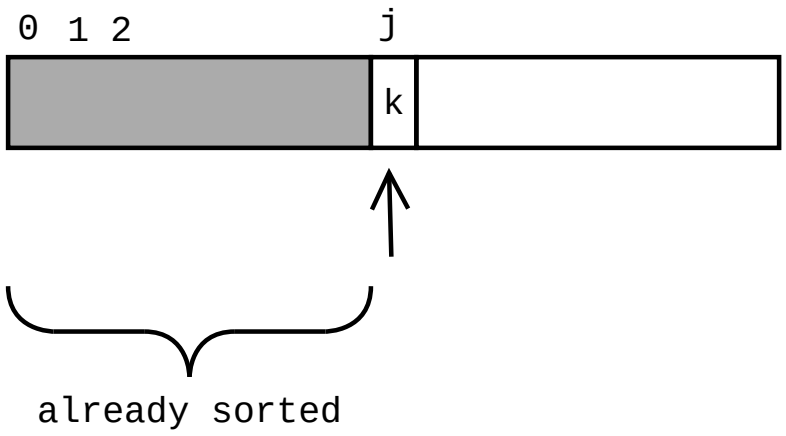
Insertion sort

- Notation (not Java!): `a[i..j]` is the part of the array from the `i`-th index to the `j`-th index.

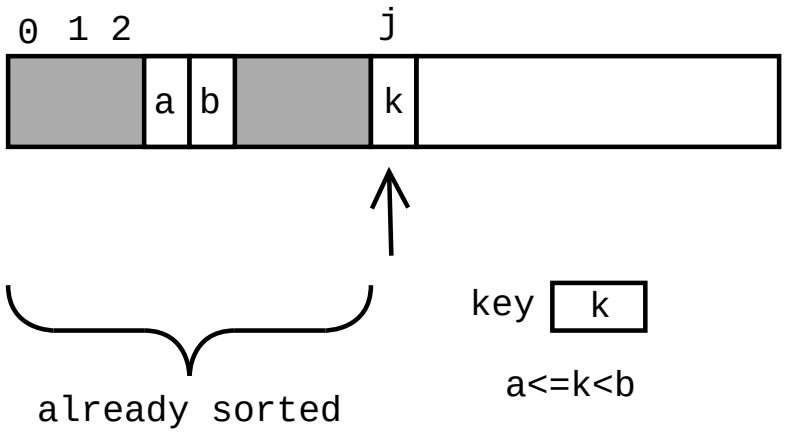
- Idea: sorting a set of cards can be done by inserting a card in the subset of the cards which are already sorted.



Insertion sort

- Notation (not Java!): `a[i..j]` is the part of the array from the `i`-th index to the `j`-th index.

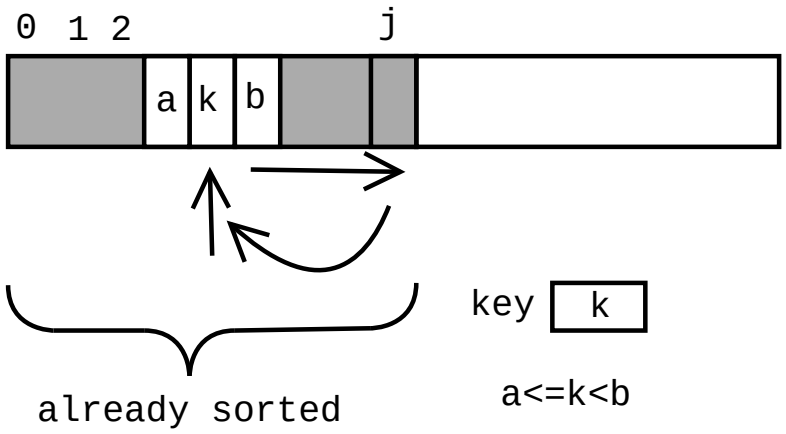
- Idea: sorting a set of cards can be done by inserting a card in the subset of the cards which are already sorted.



Insertion sort

- Notation (not Java!): `a[i..j]` is the part of the array from the `i`-th index to the `j`-th index.

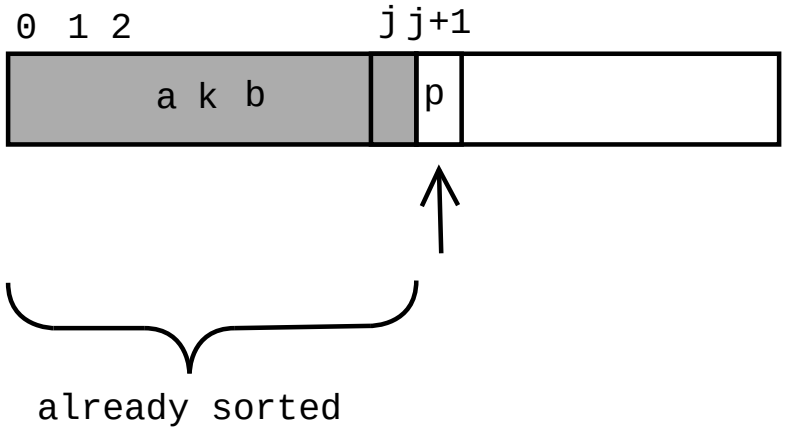
- Idea: sorting a set of cards can be done by inserting a card in the subset of the cards which are already sorted.



Insertion sort

- Notation (not Java!): `a[i..j]` is the part of the array from the `i`-th index to the `j`-th index.

- Idea: sorting a set of cards can be done by inserting a card in the subset of the cards which are already sorted.



Insertion sort

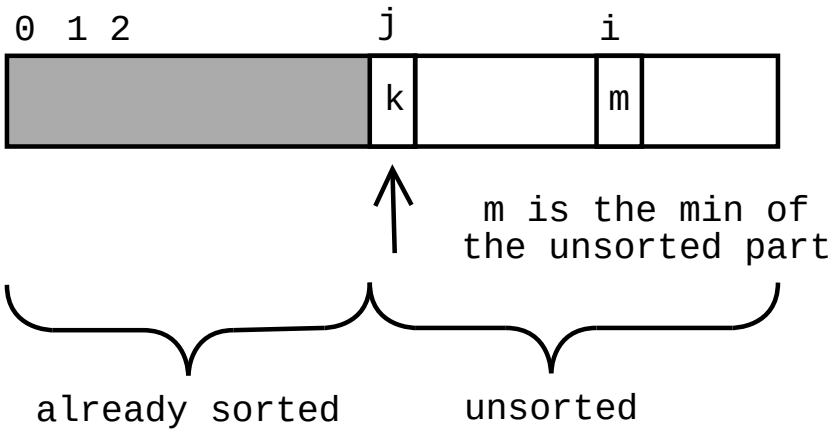
- Algorithm refined:

1. For each `j` from 1 to the length of `a-1`
   (a) Insert `a[j]` into the sorted subarray `a[0..j-1]`

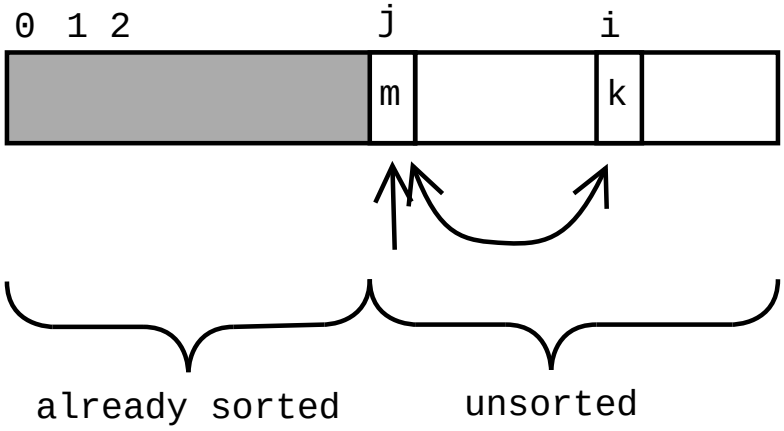
- Algorithm refined: (Full algorithm)

1. For each `j` from 1 to the length of `a-1`
   (a) Set `key` to `a[j]`
   (b) Set `i` to `j - 1`
   (c) While `i >= 0` and `a[i] > key` do
      i. Set `a[i+1]` to `a[i]`
      ii. Decrement `i` by 1
   (d) Set `a[i+1]` to `key`

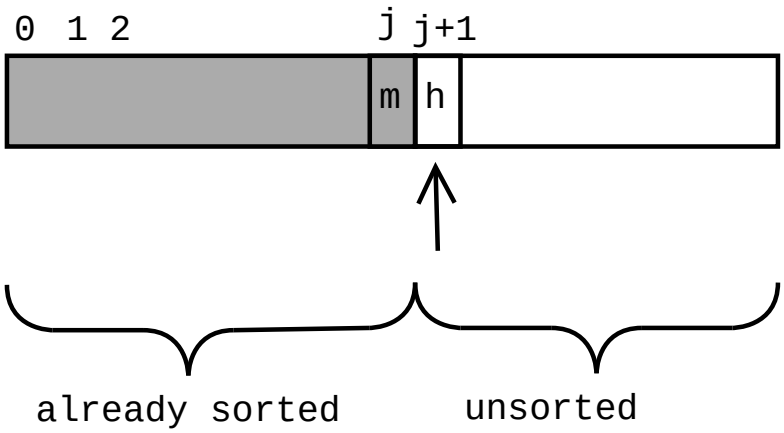
Selection sort



Selection sort



Selection sort



Selection sort

- Algorithm

1. For each j from 0 to length a - 2 do
   (a) Let min_index to be the index of the minimum in a[j+1..length a-1]
   (b) Swap a[min_index] and a[j]

- Algorithm refined

1. For each j from 0 to length a - 2 do
   (a) Let minimum be a[j]
   (b) Set min_index to j
   (c) For each i from j+1 to the length a - 1 do
      i. If a[i] < minimum then
         A. Set minimum to a[i]
         B. Set min_index to i
   (d) Swap a[min_index] and a[j]

Sorting arrays of Objects

Choose a key, which can be compared.

```
class Book {
  private String title, author;
  //...
  public String get_title() { return title; }
  public String get_author() { return author; }
  //...
}
```

Sorting arrays of Objects

- Comparing strings: Lexicographical order
- The `compareTo` method from the `String` class
- `s1.compareTo(s2)` returns a negative integer if `s1` is lexicographically before `s2`, 0 if they are equal, and a positive integer if `s1` is lexicographically after `s2`.

```
String s1 = "aacb", s2 = "aafa";
int n = s1.compareTo(s2); // n = -3;
String s3 = "aacbgg";
int m = s3.compareTo(s2); // n = -3
int k = s3.compareTo(s1); // n = 2
```

Sorting arrays of Objects

```
void insertion_sort(Book[] a)
{
  int i, j;
  String key;
  Book focus;
  for (j = 1; j < a.length; j++) {
    focus = a[j];
    key = focus.get_title();
    i = j - 1;
    while (i >= 0
      && key.compareTo(a[i].get_title()) < 0 ) {
      a[i+1] = a[i];  // copy the reference
      i--;
    }
    a[i+1] = focus;
  }
}
```

Sorting arrays of Objects

```
void selection_sort(Book[] a)
{
  int min_index;
  String minimum;
  Book temp;
  for (int j = 0; j <= a.length - 2; j++) {
    minimum = a[j].get_title();
    min_index = j;
    for (int i = j + 1; i <= a.length - 1; i++) {
      String current_key = a[i].get_title();
      if (current_key.compareTo(minimum) < 0) {
        minimum = current_key;
        min_index = i;
      }
    }
    temp = a[j];
    a[j] = a[min_index];
    a[min_index] = temp;
  }
```

Linear search

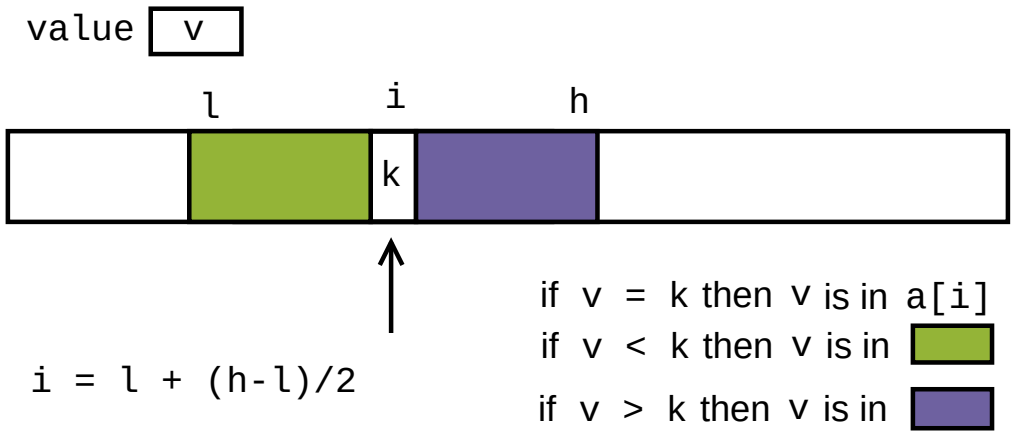
```
int linear_search(int[] a, int value)
{
  int index = 0;
  while (index < a.length) {
    if (value == a[index]) {
      return index;
    }
    index++;
  }
  return -1;  // Not found
}
```

- This works for unsorted arrays

Binary search

- But if we know that the array is sorted, we can improve the speed of searching by ignoring parts which do not need to look at.

- If we are looking for a value `v` in an array `a`, and we have already narrowed down the search space to `a[l..h]`, then



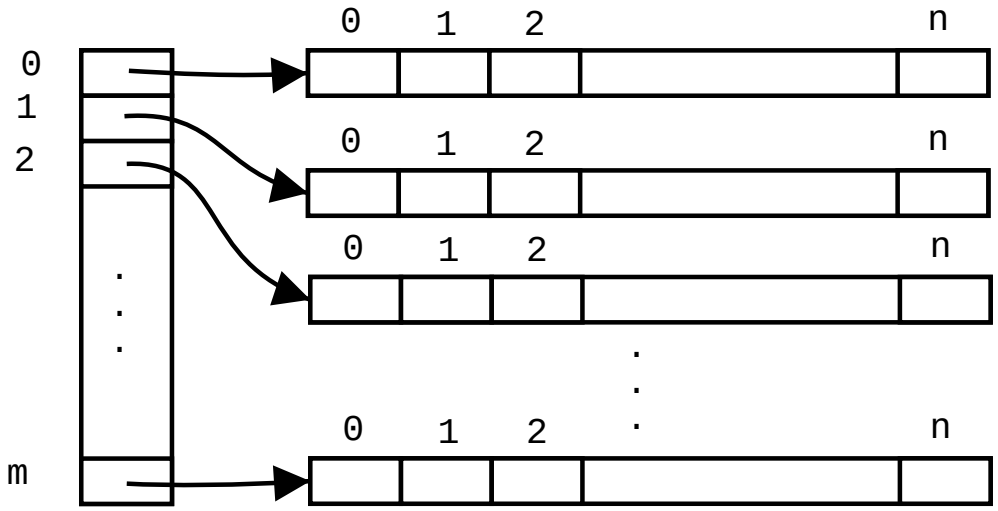
Binary search

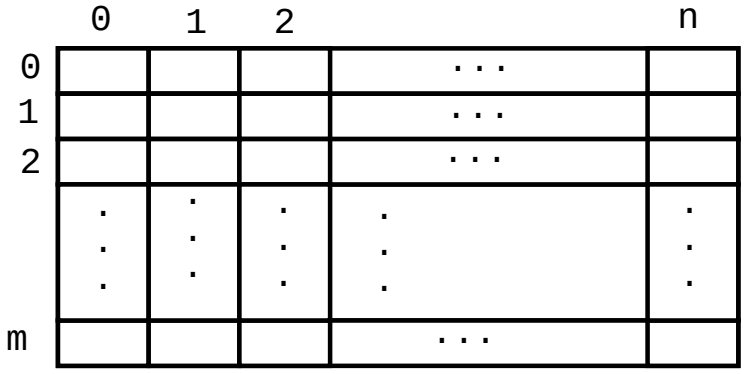
```
int binary_search(int[] a, int value)
{
  int lower = 0, higher = a.length - 1, index;
  while (lower <= higher) {
    index = lower + (higher - lower) / 2;
    if (value == a[index]) {
      return index;
    }
    else if (value < a[index]) {
      higher = index - 1;
    }
    else {  // value > a[index]
      lower = index + 1;
    }
  }
  return -1;  // Not found
}
```

Binary search

```
int binary_search(Book[] a, String title)
{
  int lower = 0, higher = a.length - 1, index;
  String current_title;
  int comparison;
  while (lower <= higher) {
    index = lower + (higher - lower) / 2;
    current_title = a[index].get_title();
    comparison = title.compareTo(current_title);
    if (comparison == 0) {
      return index;
    }
    else if (comparison < 0) {
      higher = index - 1;
    }
    else {  // comparison > 0
      lower = index + 1;
    }
  }
  return -1;  // Not found
}
```

Multidimensional arrays





Multidimensional arrays

- A two-dimensional array is an array of arrays.

```
int[][] table = new int[5][10];

for (int row=0; row < table.length; row++)
  for (int col=0; col < table[row].length; col++)
    table[row][col] = row * 10 + col;
```

- A multidimensional array is an n-dimensional array, i.e. an array of arrays of arrays of ...

- Processing nested arrays is commonly done with nested loops.

Multidimensional arrays

- A two-dimensional array is an array of arrays.

```
int[][] table = new int[5][10];

for (int row=0; row < table.length; row++)
  for (int col=0; col < table[row].length; col++)
    table[row][col] = row * 10 + col;

for (int row = 0; row < table.length; row++) {
  for (int col=0; col < table[row].length; col++)
    System.out.print(table[row][col]+"\t");
  System.out.println();
}
```

- A multidimensional array is an n-dimensional array, i.e. an array of arrays of arrays of ...

- Processing nested arrays is commonly done with nested loops.

Multidimensional arrays

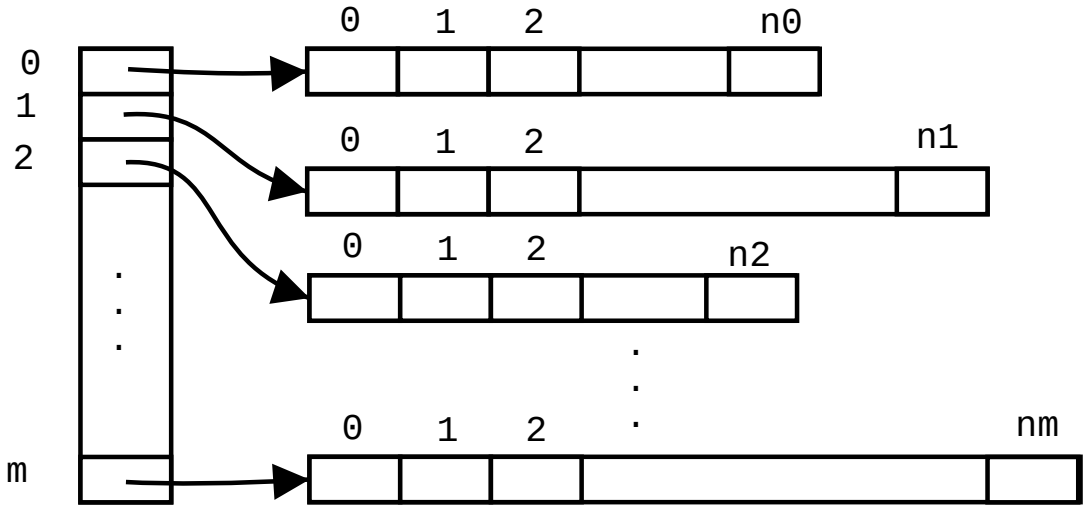
- A two-dimensional array can be an array of objects

```
class A { int x; }

// and in the client
A[][] table = new A[5][10];
for (int row=0; row < table.length; row++)
  for (int col=0; col < table[row].length; col++)
    table[row][col] = new A();
    table[row][col].x = row * 10 + col;
  }

for (int row = 0; row < table.length; row++) {
  for (int col=0; col < table[row].length; col++)
    System.out.print(table[row][col].x+"\t");
  System.out.println();
}
```

Multidimensional arrays



Multidimensional arrays

- Each row can have different length

```
class A { int x; }

// and in the client
A[][] table = new A[5][];
for (int row=0; row < table.length; row++) {
  table[row] = new A[row];
  for (int col=0; col < table[row].length; col++)
    table[row][col] = new A();
    table[row][col].x = row * 10 + col;
  }
}

for (int row = 0; row < table.length; row++) {
  for (int col=0; col < table[row].length; col++)
    System.out.print(table[row][col].x+"\t");
  System.out.println();
}
```

The end