CS 1112 Final Review

Review Topics

- **Review of these topics:**
  - Object-oriented programming
  - Recursion
  - Sorting algorithms
  - Searching algorithms
- **Some example exam problems**

Objects and Classes

- **Class:** A file that specifies *properties* (variables) and *methods* (functions) associated with the item that the class represents
  - Contains a *constructor*, a special method that creates new objects
  - A class can have *subclasses*

- **Object:** One *instance* of a class
  - Objects of the same class have the same properties and the same methods
  - The properties of objects of the same class can have *different values*

Objects and Classes Example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end
    methods
        function aml = Animal(n, s, a, hT)
            % set properties of aml
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

Note that the `end` keyword is used to close the following:

1. The `classdef`
2. The `properties` section
3. The `methods` section
4. Each `function` inside the `methods` section

Objects and Classes: **Constructors**

Constructor: A method (function) that creates a new object

- Must have the same name as the class
- Can take in parameters to set property values
- Use `nargin` to ensure that constructor can be called without any arguments

Objects and Classes Example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end
    methods
        function aml = Animal(n, s, a, hT)
            % set properties of aml
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

Implementation of this constructor:

```
function aml = Animal(n, s, a, hT)
    if (nargin == 4)
        aml.name = n;
        aml.species = s;
        aml.age = a;
        aml.hasTail = hT;
    end
end
```

If 4 arguments are not provided, the 4 properties will be set to default values.

Objects and Classes: **Create/reference objects**

Create new objects by calling the constructor, which returns a *reference to the new object* that should be stored in a variable.

Example:

```
a = Animal('Bobbert', 'pig', 2, 1);
%   An animal object with these properties is created:
%   name = 'Bobbert', species = 'pig', age = 2, hasTail = 1
%   a is the reference to this object.
```

Create an empty array of Animal objects using `.empty()`

Example: `b = Animal.empty()`

Check if an object/object array is empty using `isempty(<reference>)`

Example: `isempty(a)` returns 0, `isempty(b)` returns 1

Objects and Classes: **Calling methods**

Each method in a class takes in a minimum of one parameter (named `'self'`), which is *a reference to the object calling the method*

Syntax for calling a method:

`<reference>.<methodName>(2nd through last input variable)`

This is equivalent (but it is better to use the above way):

`<methodName>(self, 2nd through last input variable)`

Objects and Classes Example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end
    methods
        function aml = Animal(n, s, a, hT)
            % set properties of aml
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

How to use this method (from another script, function, etc.):

```
% Object reference should be
% created first
a = Animal('Bobbert', 'pig', 2, 1);

% Call method
a.birthday();     % or: birthday(a);

% See result of method call
disp(a.age)       % 3 will be displayed
```

Objects and Classes Example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end
    methods
        function aml = Animal(n, s, a, hT)
            % set properties of aml
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

Implementation of this method:

```
function c = checkHasTail(self)
    if (self.hasTail == 1)
        c = 1;
    else
        c = 0;
    end
end
```

Objects and Classes Example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end
    methods
        function aml = Animal(n, s, a, hT)
            % set properties of aml
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

Implementation of this method:

```
function c = isOlder(self, otherAnimal)
    if (self.age > otherAnimal.age)
        c = 1;
    else
        c = 0;
    end
end
```

How to use this method:

Age of a is 2

Age of b is 1

```
a = Animal('Bobbert', 'pig', 2, 1);
b = Animal('Robbert', 'frog', 1, 0);
disp(a.isOlder(b))    % will display 1
disp(b.isOlder(a))    % will display 0
```

Objects and Classes: **Arrays of objects**

Objects of the same class can be stored in a simple vector/array.

Objects of different classes (even classes which are related by inheritance) must be stored in a cell array.

Example: Write a function that takes in a vector z of Animal objects and returns a vector of the indices from z which contain objects whose species is ‘pig’:

```
function idx = FindPigs(z)
idx = []; k = 1;
for i = 1:length(z)
    if (strcmp(z(i).species, ‘pig’))
        idx(k) = i;
        k = k+1;
    end
end
```

Objects and Classes: **Accessibility**

Keywords `public, private, protected` can be used to restrict access to properties.

- **Public** properties: can be directly accessed in any subclasses or any other files that create objects of the class
- **Private** properties: cannot be directly accessed outside the class methods
- **Protected** properties: can only be directly accessed inside class methods and subclass methods.

Private and protected can never be directly accessed from the command line

If direct access is not possible, consider writing a ‘get’ method:

```
<reference>.getPropertyValue()
```

Objects and Classes: **Inheritance**

- A class can have subclasses that share properties and methods.
  - **Private** properties are not inherited, but can be accessed through methods
  - **Protected** properties are inherited; all subclasses can access them
  - **Public** properties are inherited; all classes can access them

- **In the constructor of a subclass**, there must be a call to the superclass constructor (using “@” notation)

Objects and Classes Example: **Animal and Bird**

```
classdef Animal < handle
    properties (Access = protected)
        name; species; age; hasTail
    end
    methods
        function aml = Animal(n, s, a, hT)
            % set properties of aml
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

```
classdef Bird < Animal
    properties (Access = private)
        color
    end
    methods
        function b = Bird(n, s, a, c)
            b = b@Animal(n, s, a, 1);
            b.color = c;
        end
        function c = getColor(self)
            c = self.color;
        end
    end
end
```

hasTail = 1 for all Bird objects!

Would I be able to write a function in the bird class that accessed bird.name?

Review Question on OOP

The equation of a line can be written in the "slope-intercept form" $y = mx + b$ where m is the slope and b is the y-intercept. Given the the slope m and y-intercept b of a line, we can compute the following:

- The y value at some x-coordinate x_0 is $mx_0 + b$
- Shifting the line in the y-direction (up or down) by $\triangle y$ changes the y-intercept by $\triangle y$ but the slope remains the same.
- If line 1 with slope m_1 and y-intercept b_1 is not parallel to line 2 with slope m_2 and y-intercept b_2, then the two lines have one point of intersection (x_i, y_i) where

$$x_i = \frac{b_2 - b_1}{m_1 - m_2}, \qquad y_i = \frac{b_2 m_1 - b_1 m_2}{m_1 - m_2}$$

  If lines 1 and 2 are parallel, then there is no intersection and we say that x_i and y_i are `NaN` (Not A Number).

Implement class `NVLine` by completing the methods block as specified in the comments below.

```
classdef NVLine < handle
% A non-vertical line on the Cartesian plane with slope m and intercept b.
% This class defines NON-VERTICAL lines only.

   properties (Access=private)
      m=0;  % slope, cannot be inf (i.e., cannot be vertical line)
   end

   properties (Access=protected)
      b=0;  % y-intercept
   end

   methods
      function NVL = NVLine(s, yi)
      % Construct an NVLine object with slope s and y-intercept yi. If the number
      % of arguments passed is not 2 or if s is inf, stop program execution with
      % a descriptive error message.
```

% (Continued on next page.)

```
% methods block of class NVLine, continued

function y0 = yGivenX(self, x0)
% y0 is the y-coordinate of the NVLine referenced by self at the
% x-coordinate x0.









end

function newLine = shift(self, deltaY)
% newLine is a new NVLine object with the slope of self but shifted in the
% y-direction by deltaY.  self references an NVLine.









end % (Continued on next page.)
```

```
        function tf = parallel(self, other)
        % tf is true if self and other are parallel; otherwise false.
        % self, other each references an NVLine.

           % DO NOT IMPLEMENT.
           % ASSUME THIS METHOD IS IMPLEMENTED CORRECTLY.
        end

        function [xi, yi] = intersect(self, other)
        % xi, yi are the x- and y-coordinates of the point of intersection between
        % self and other.  self, other each references an NVLine.  If self and other
        % are parallel, then xi and yi should be NaN.  Make effective use of instance
        % method parallel.

        end
    end %methods
end %classdef
```

Review Question on OOP

Designing an algorithm

#	Thing we need to do	Task decomposition
1	Complete the constructor NVLine	`Determine number of input arguments; Display error messages; Assign property to an object.`
2	Complete methods yGivenX and shift	Use the given math formula
4	Complete the methods intersect	Make use of the given parallel function; use if conditions to determine if there is an intersection or not; compute the intersection using given math if it exists.

Review Question on OOP: **Solution**

```
properties (Access=private)
   m=0;  % slope, cannot be inf (i.e., cannot be vertical line)
end

properties (Access=protected)
   b=0;  % y-intercept
end

methods
   function NVL = NVLine(s, yi)
   % Construct an NVLine object with slope s and y-intercept yi. If the number
   % of arguments passed is not 2 or if s is inf, stop program execution with
   % a descriptive error message.


         %% EXAMPLE SOLUTION
         %
         %  if nargin~=2 || isinf(s)
         %      error('Not enough arguments or line is vertical')
         %  end
         %  NVL.m= s;
         %  NVL.b= yi;
         %%


   end
```

```
% methods block of class NVLine, continued

function y0 = yGivenX(self, x0)
% y0 is the y-coordinate of the NVLine referenced by self at the
% x-coordinate x0.


      %% EXAMPLE SOLUTION
      %
      %  y0= self.m*x0 + self.b;
      %%


end

function newLine = shift(self, deltaY)
% newLine is a new NVLine object with the slope of self but shifted in the
% y-direction by deltaY.  self references an NVLine.


      %% EXAMPLE SOLUTION
      %
      %  newLine= NVLine(self.m, self.b+deltaY);
      %%


end
```

```
        function tf = parallel(self, other)
        % tf is true if self and other are parallel; otherwise false.
        % self, other each references an NVLine.

           % DO NOT IMPLEMENT.
           % ASSUME THIS METHOD IS IMPLEMENTED CORRECTLY.
        end

        function [xi, yi] = intersect(self, other)
        % xi, yi are the x- and y-coordinates of the point of intersection between
        % self and other.  self, other each references an NVLine.  If self and other
        % are parallel, then xi and yi should be NaN.  Make effective use of instance
        % method parallel.


              %% EXAMPLE SOLUTION
              %
              %  if self.parallel(other)
              %      xi= NaN;
              %      yi= NaN;
              %  else
              %      xi= (other.b - self.b)/(self.m - other.m);
              %      yi= (other.b*self.m - self.b - other.m)/(self.m - other.m);
              %  end
              %%

        end
    end %methods
end %classdef
```

Recursion

- A **recursive function** is a function that calls itself repeatedly with a smaller input variable each time
- Recursion stops when the parameter becomes so small that it reaches the *base case* of the function

Example

Write a function that recursively computes a *factorial*.

e.g. 4! = 4 * 3!
= 4 * 3 * 2!
= 4 * 3 * 2 * 1!
= 4 * 3 * 2 * 1

```
function m = Factorial(n)
    if n == 1 % base case
        m = 1;
    else      % recursive case
        m = n * Factorial(n-1);
    end
```

Recursion Example: Review Question #10

P14.1.3 Write a function that computes the reverse of a char array s *recursively*. Thus, if `s = 'abcde'`, then `'edcba'` is its reverse. Note that if *n* is the length of `s`, then the reverse of `s` is the concatenation of the reverse of `s(2:n)` and `s(1)` in that order. Using this idea, write a recursive function `t = Reverse(s)` that does this.

Recursion Example: Review Question #10

P14.1.3 Write a function that computes the reverse of a char array s *recursively*. Thus, if `s = 'abcde'`, then `'edcba'` is its reverse. Note that if *n* is the length of `s`, then the reverse of `s` is the concatenation of the reverse of `s(2:n)` and `s(1)` in that order. Using this idea, write a recursive function `t = Reverse(s)` that does this.

Reverse([‘a’,’b’,’c’,’d’,’e’]) -> [Reverse([‘b’,’c’,’d’,’e’]), ‘a’]

Recursion Example: Review Question #10

Write a function that recursively reverses a string.

E.g. ‘abcde’ → ‘edcba’

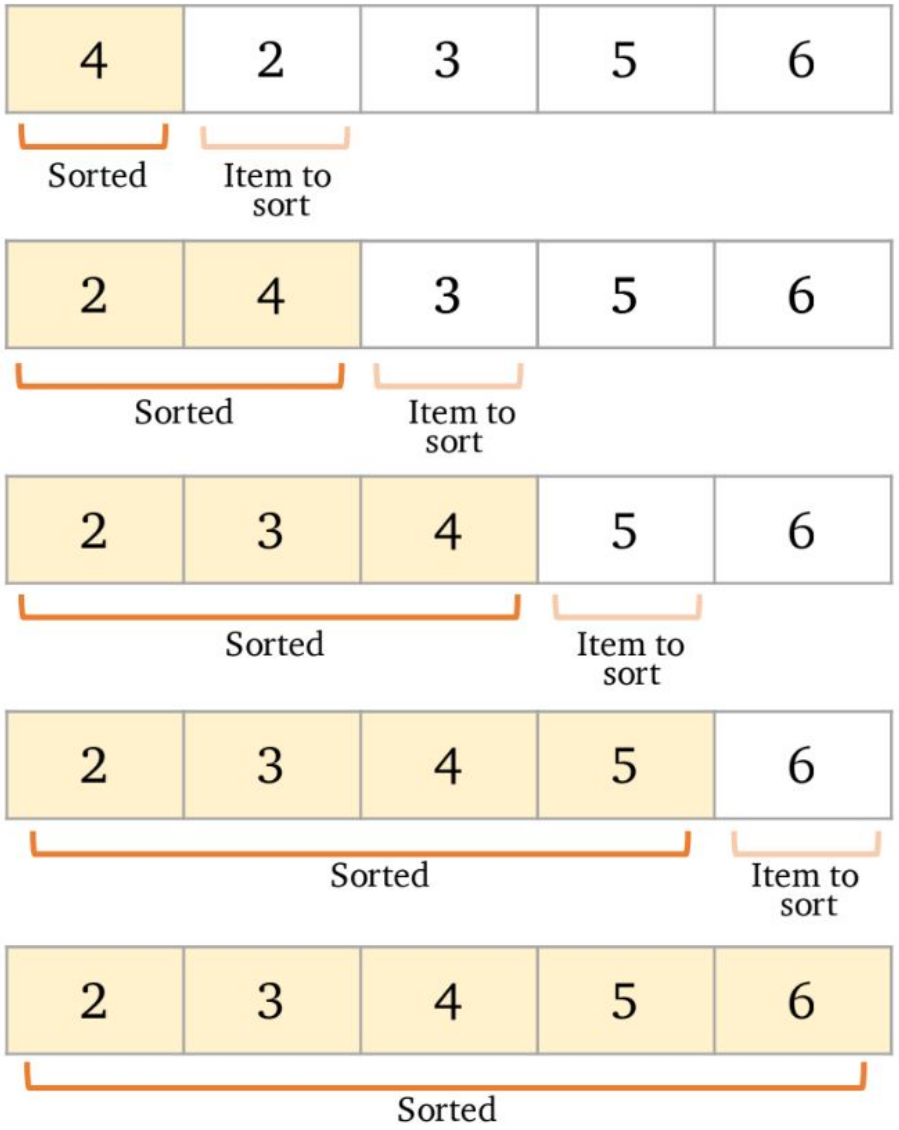
```
function t = Reverse(s)
    n = length(s);
    if n == 1     % base case: if n == 1, s is the reverse of itself
        t = s;
    else          % reverse the last n-1 characters of s, append to s(1)
        t = [Reverse(s(2:n)), s(1)];
    end
```

Sorting algorithms: **Insertion sort**

On each iteration of insertion sort, the algorithm does the following:

- Assume that the first *k* elements of the array are sorted
- Look at the $(k+1)^{th}$ element, and insert it into the correct position among the first k elements
- Now we can assume that the first (*k+1*) elements are sorted
- Repeat the above until: (*k+1*) = length of array

Sorting algorithms: **Insertion sort - Example**



Iteration 1: `x(1:1)` is sorted

Iteration 2: `x(1:2)` is sorted

Iteration 3: `x(1:3)` is sorted

Iteration 4: `x(1:4)` is sorted

Iteration 5: `x(1:5)` is sorted

Sorting algorithms: **Insertion sort**

Insertion sort algorithm: Sort a vector x

```
n = length(x)
for k = 1:n-1      % Repeat until k+1 = n
    % Sort x(1:k+1) given that x(1:k) is sorted: move the item at
    % position (k+1) backwards until it is in the correct place.
    j = k;
    need2swap = x(j+1) < x(j);  % Check if need to move (k+1)th item backwards
    while need2swap             % continue moving the item backwards until
        temp = x(j);            % it is in the correct place
        x(j) = x(j+1);
        x(j+1) = temp;
        j = j-1;
        need2swap = j>0 && x(j+1)<x(j);
    end
end
```

How much “work” is insertion sort?

- In the worst case, make *k* comparisons to insert an element in a sorted array of *k* elements. For an array of length N:

$$1 + 2 + \ldots + (N-1) = N(N-1)/2, \text{ say } N^2 \text{ for big N}$$

Sorting algorithms: **Merge sort**

Merge sort on an array of length *n* works by:

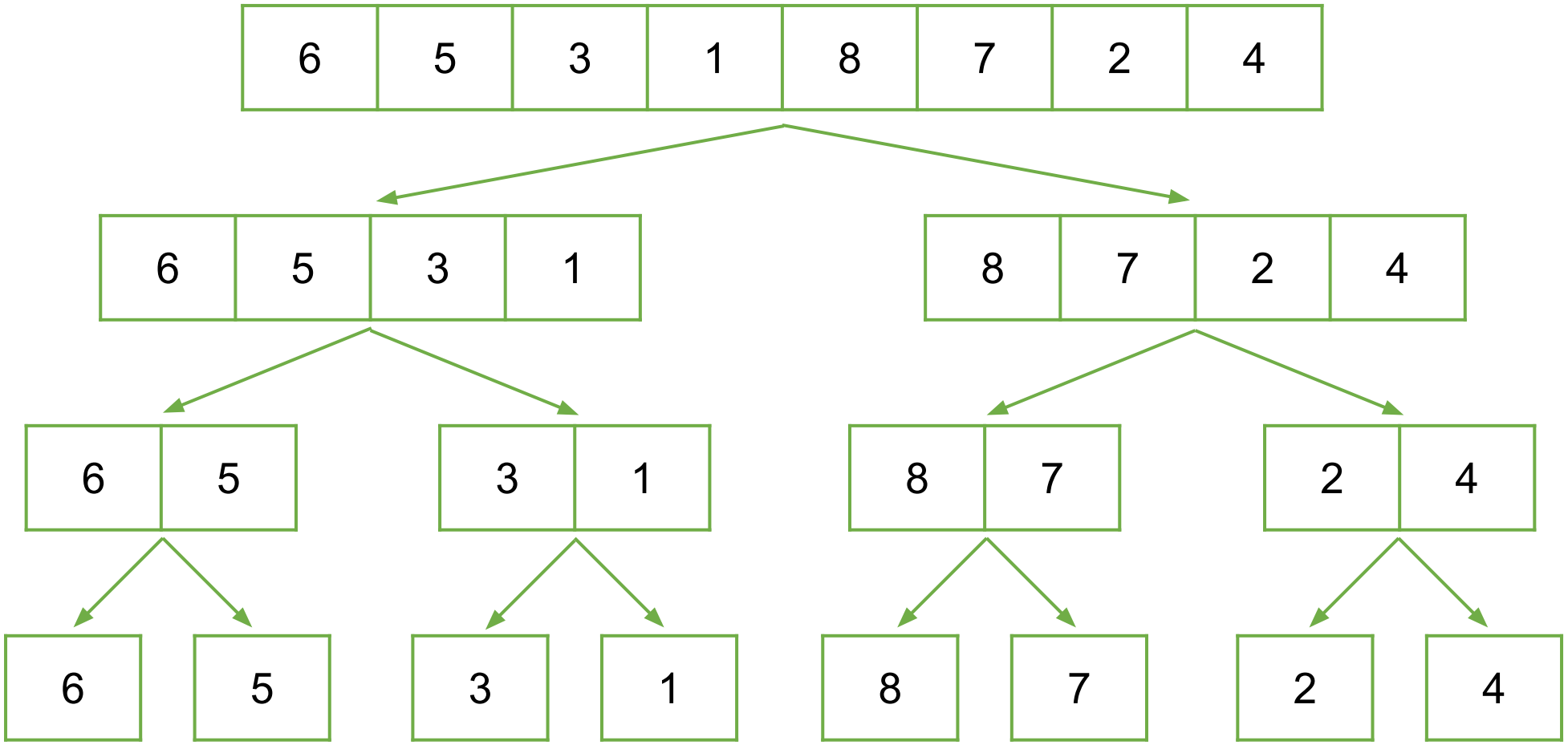
- Dividing the vector into *n* arrays of 1 component each
- Merge adjacent components in sorted order to produce `ceil(n/2)` arrays of length 2
- Merge adjacent vectors of length 2 in sorted order to produce `ceil(n/4)` arrays of length 4

… continue merging until 1 sorted array of length *n* is produced

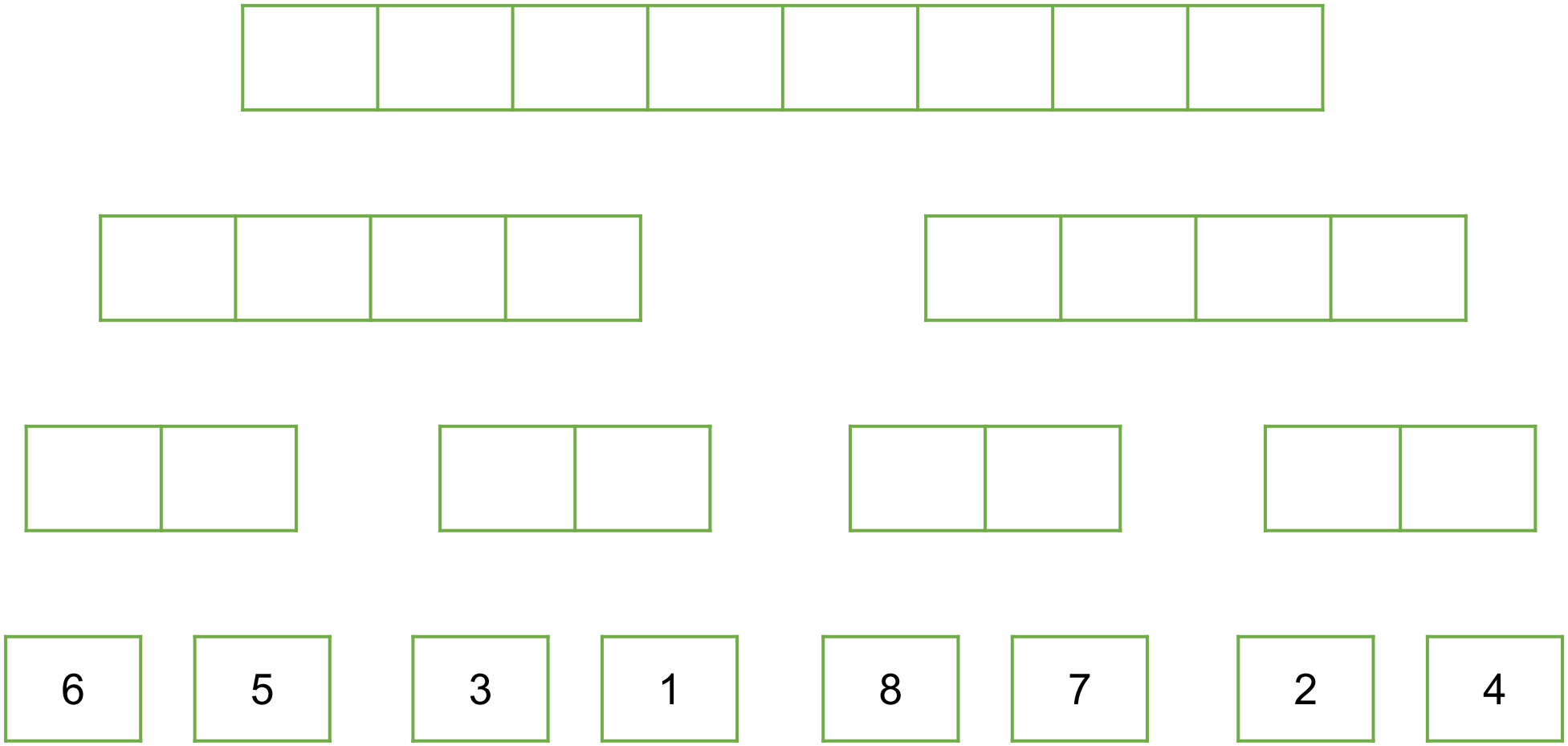
```
function y = mergeSort(x)
% x is a vector. y is a vector
% consisting of the values in x
% sorted from smallest to largest

n = length(x);
if n == 1
    y = x;
else
    m = floor(n/2);
    yL = mergeSort(x(1:m));
    yR = mergeSort(x(m+1:n));
    y = merge(yL,yR);
end
```

```
function z = merge(x,y)
nx = length(x); ny = length(y);
z = zeros(1, nx+ny);
ix = 1; iy = 1; iz = 1;
while ix<=nx && iy<=ny
    if x(ix) < y(iy)
        z(iz)=x(ix); ix=ix+1; iz=iz+1;
    else
        z(iz)=y(iy); iy=iy+1; iz=iz+1;
    end
end
while ix<=nx % copy remaining x-values
    z(iz)=x(ix); ix=ix+1; iz=iz+1;
end
while iy<=ny % copy remaining y-values
    z(iz)=y(iy); iy=iy+1; iz=iz+1;
end
```

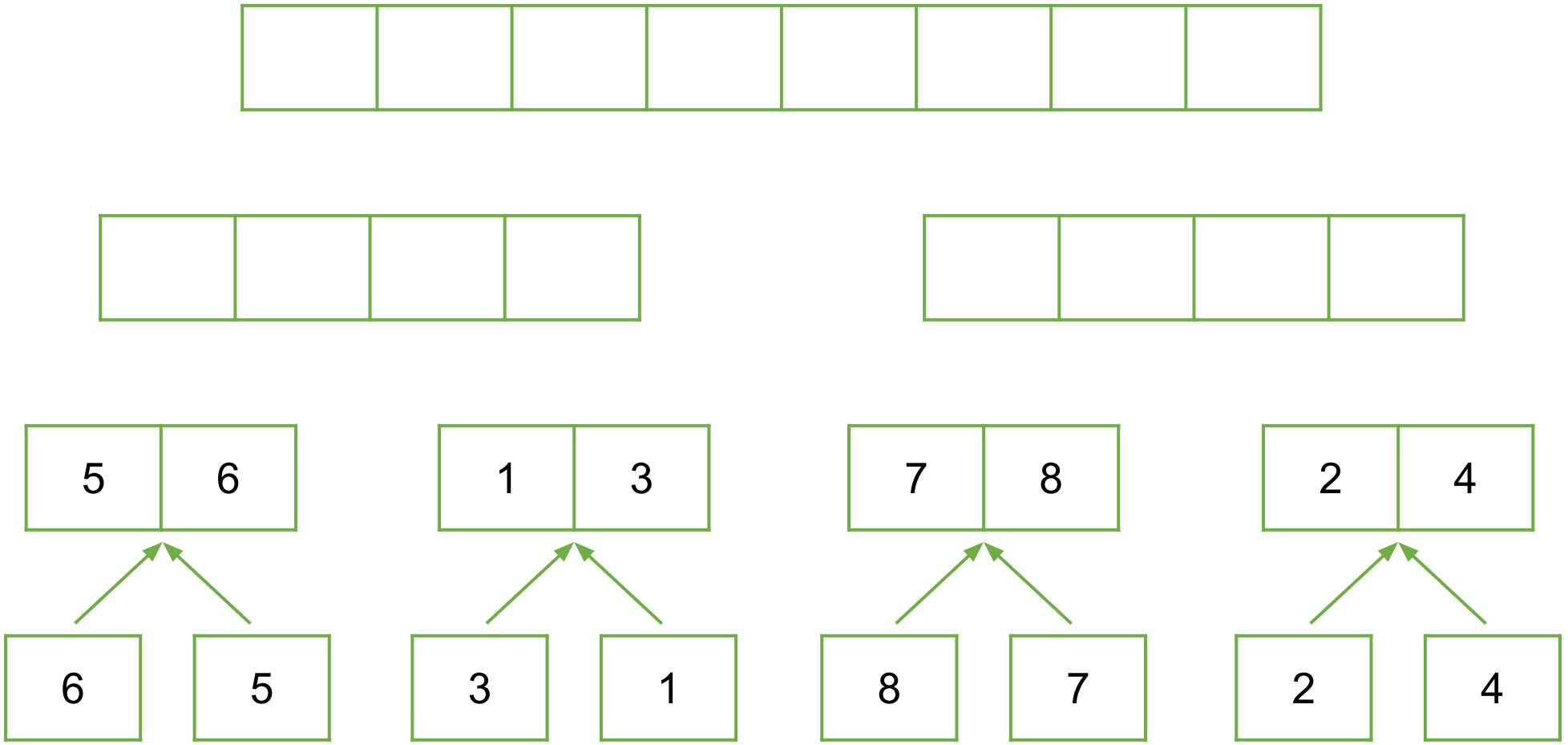
Sorting algorithms: **Merge sort - Example**



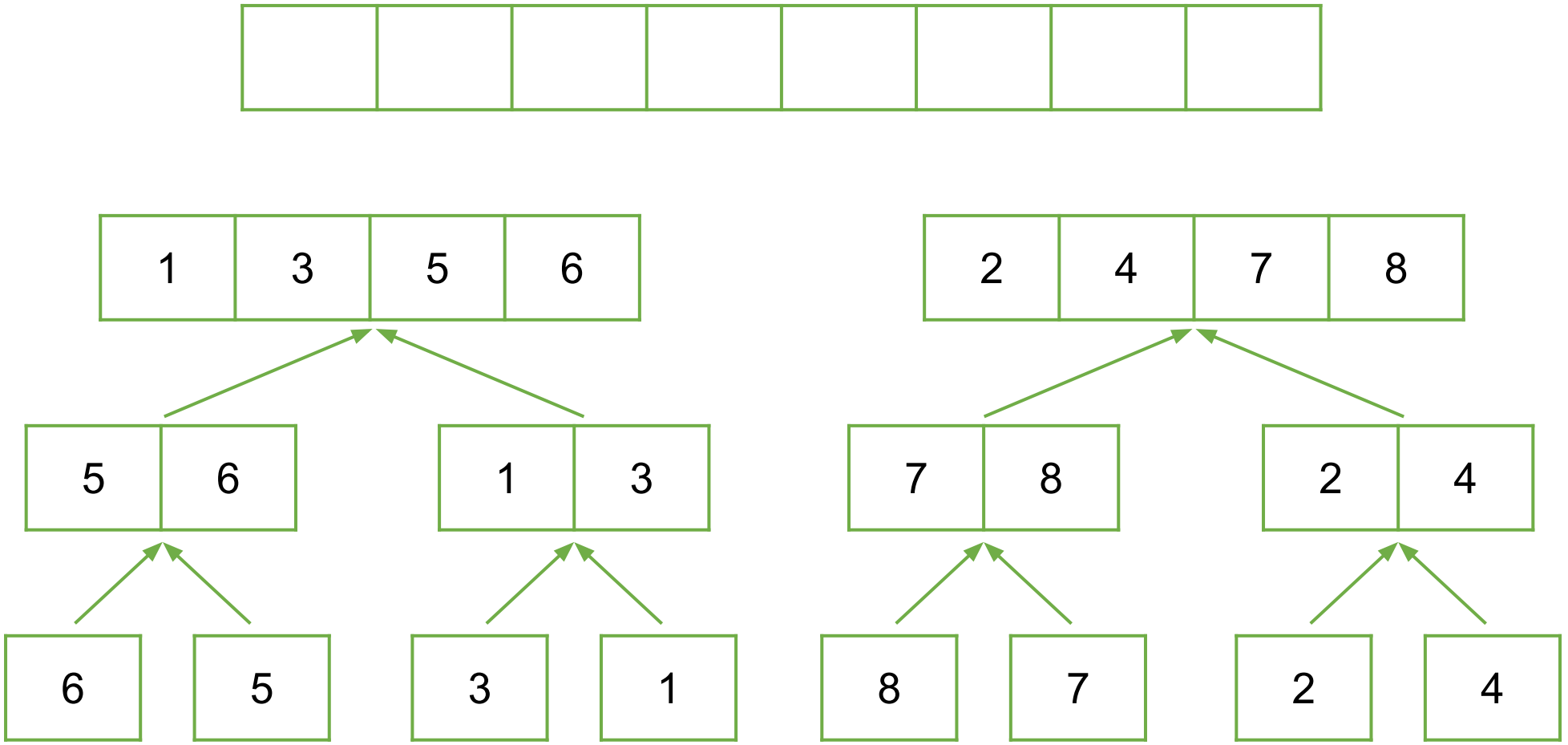
Sorting algorithms: **Merge sort - Example**



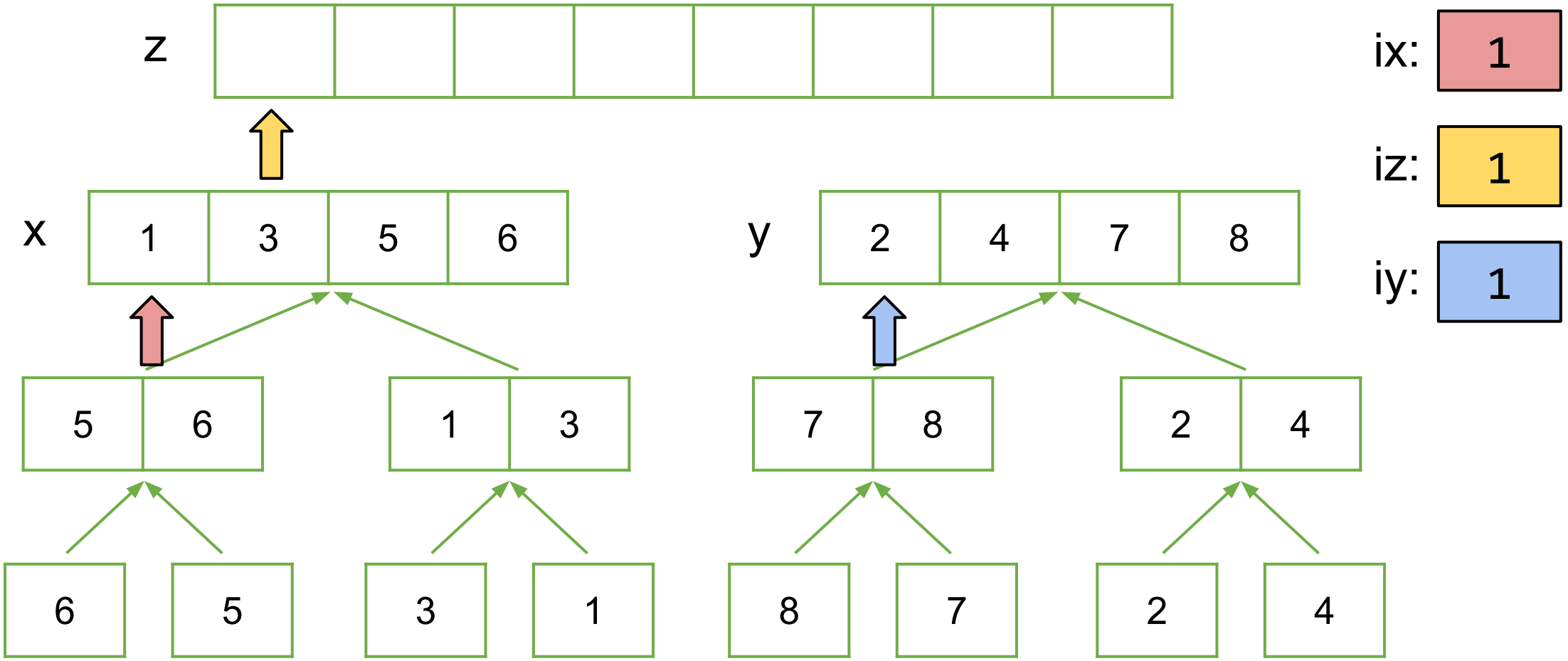
Sorting algorithms: **Merge sort - Example**



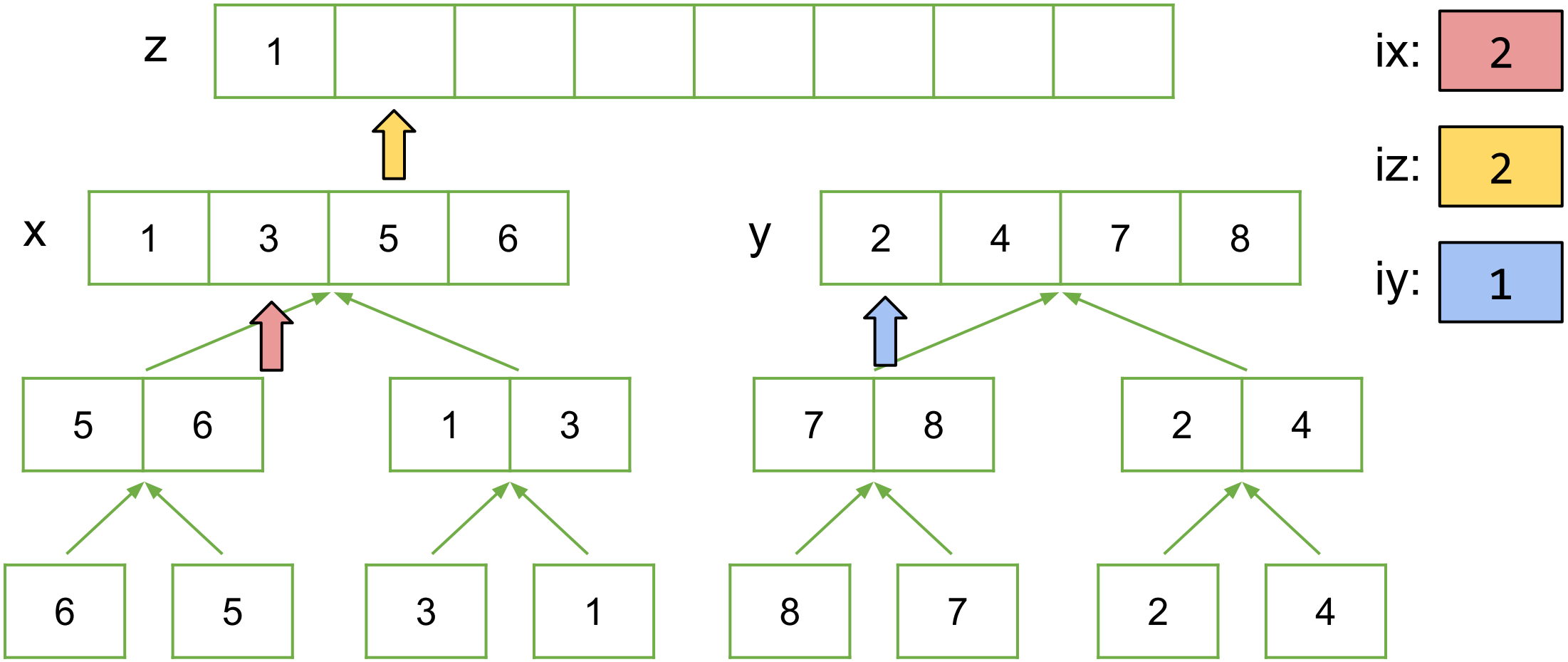
Sorting algorithms: **Merge sort - Example**



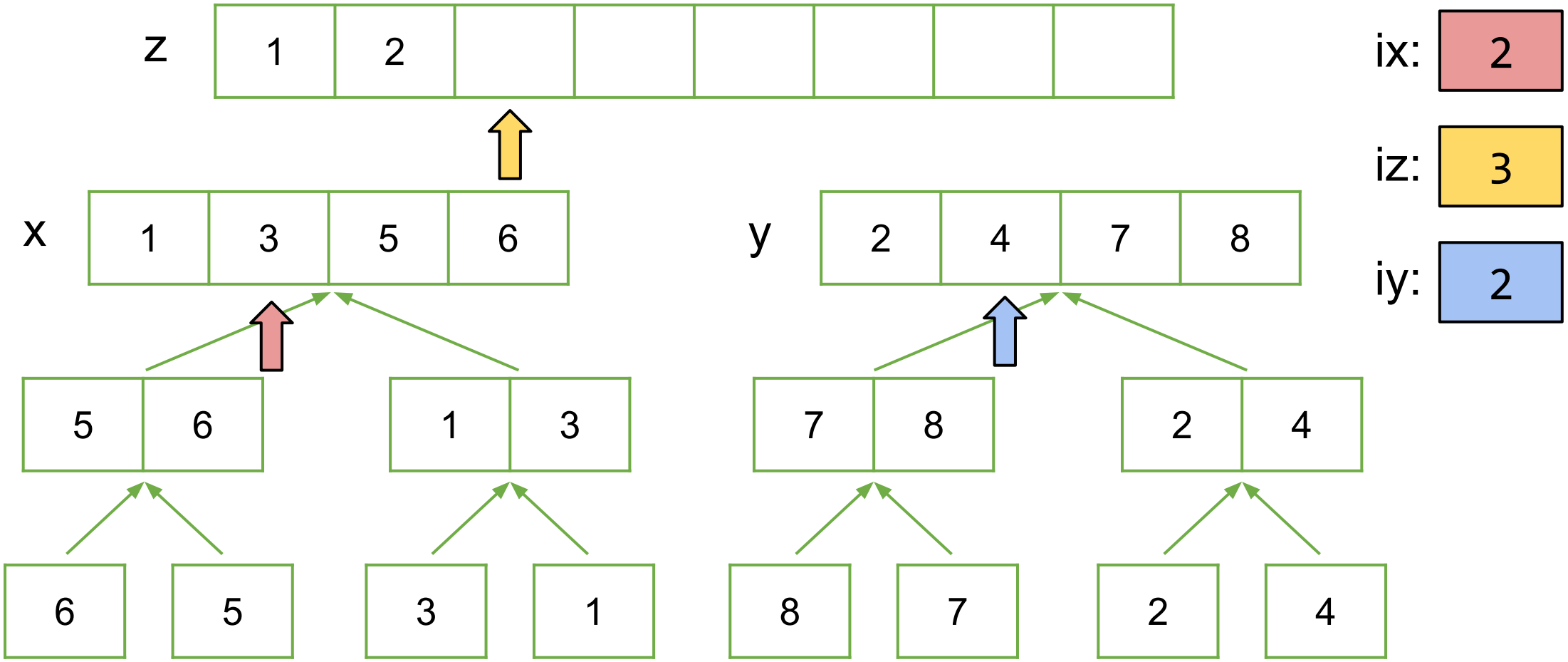
Sorting algorithms: **Merge sort - Example**



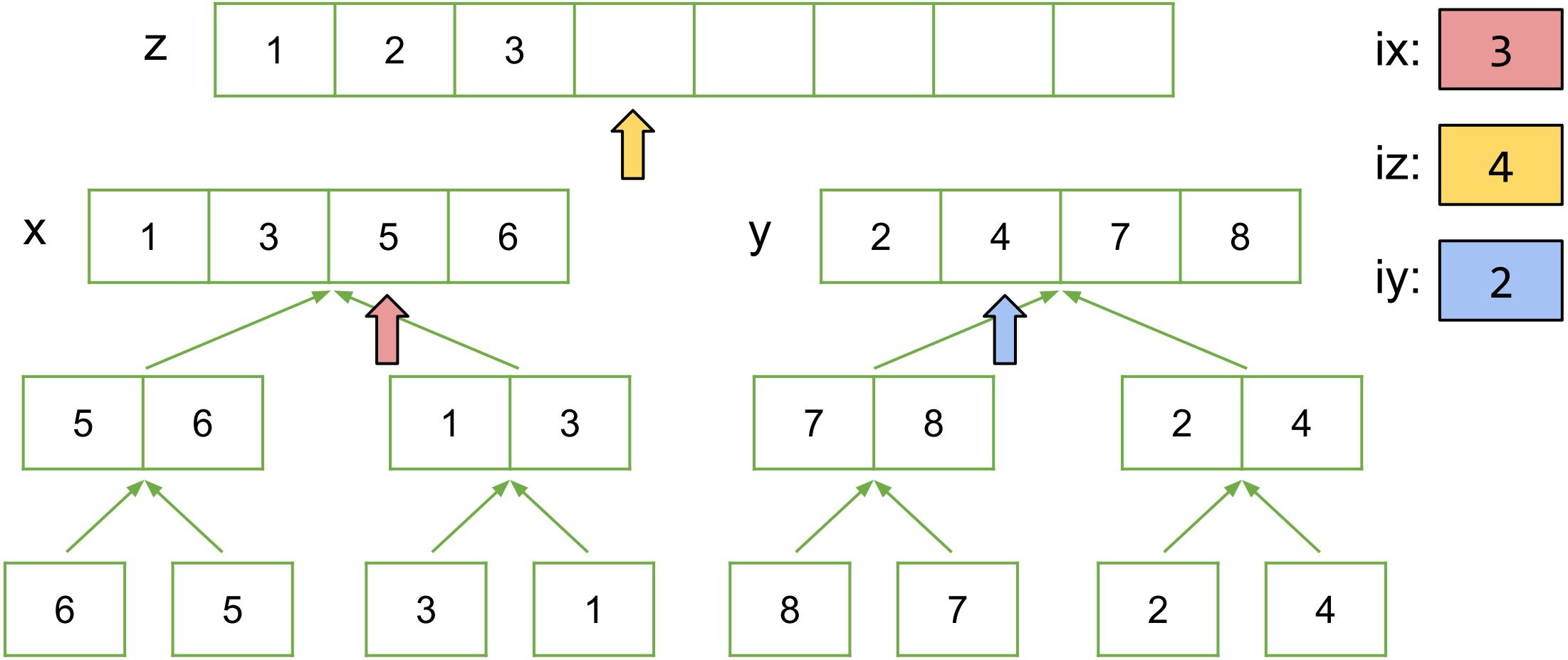
Sorting algorithms: **Merge sort - Example**



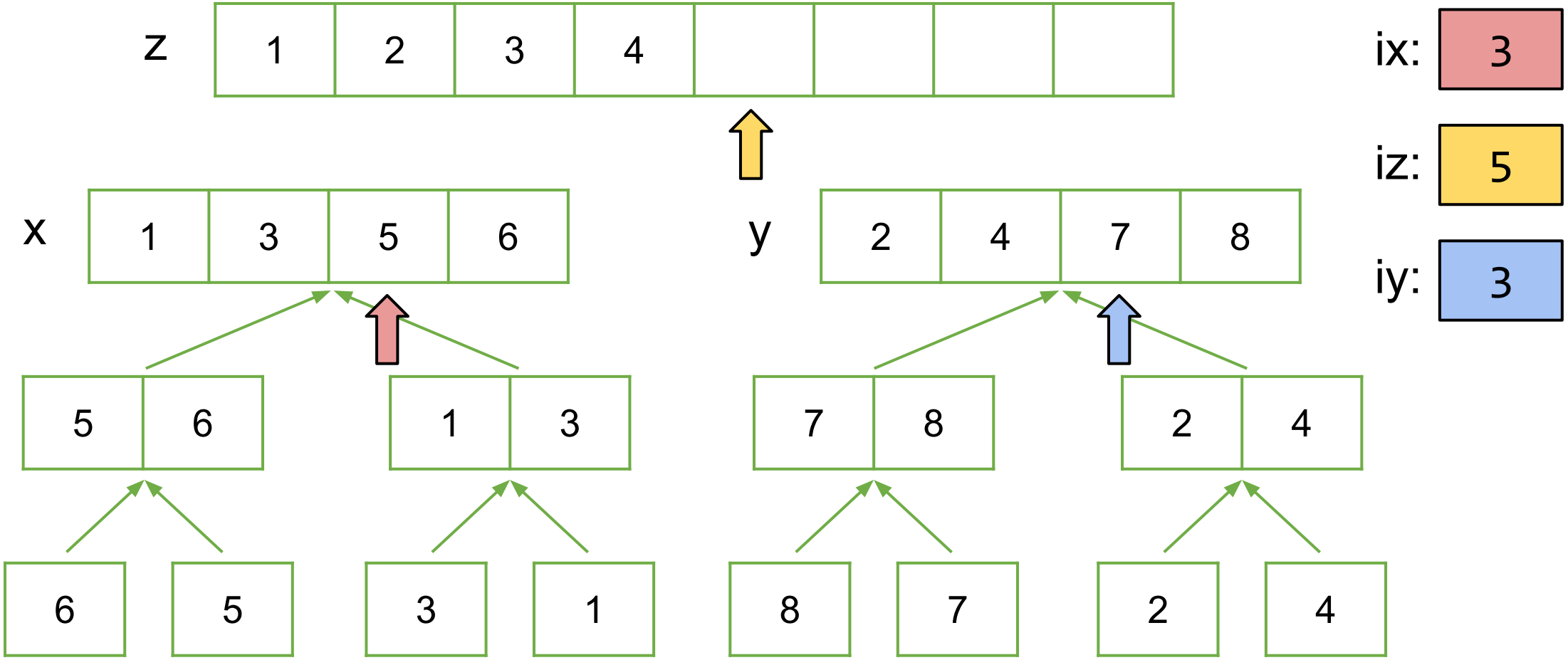
Sorting algorithms: **Merge sort - Example**



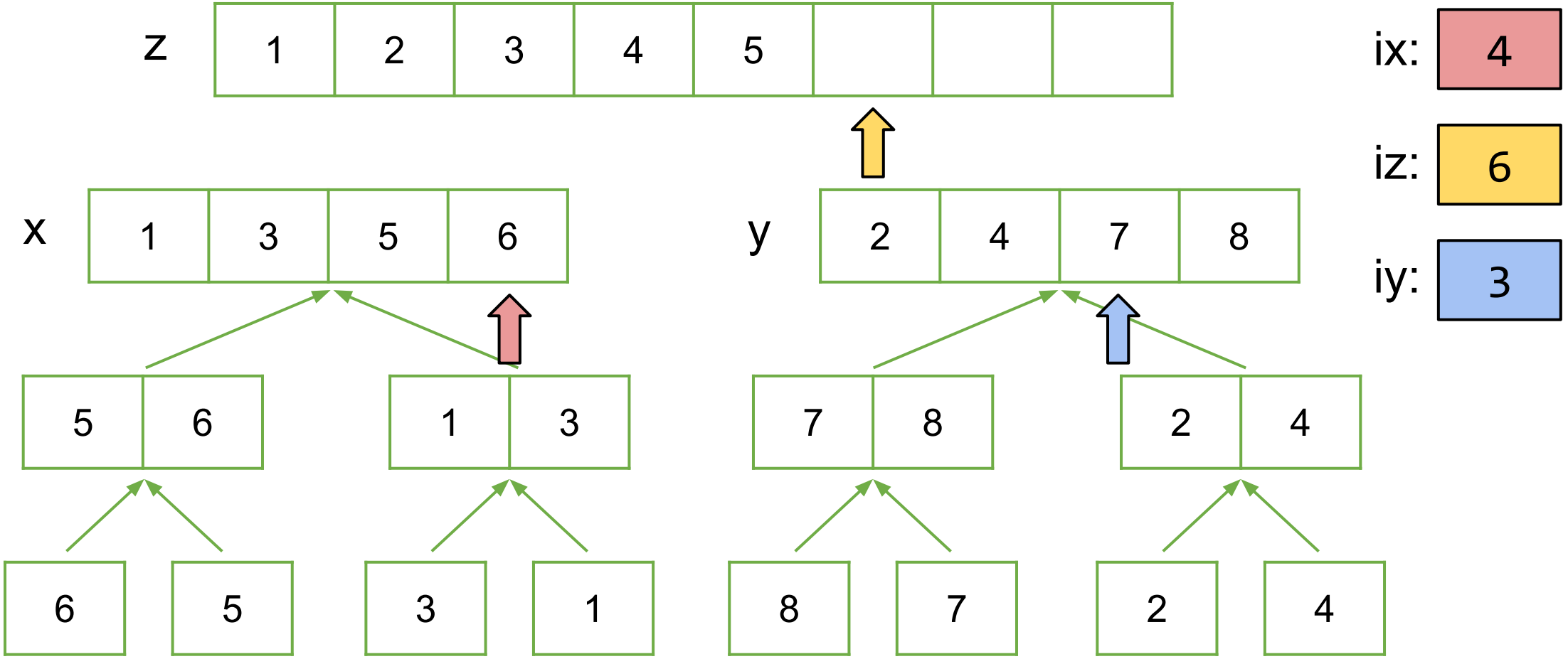
Sorting algorithms: **Merge sort - Example**



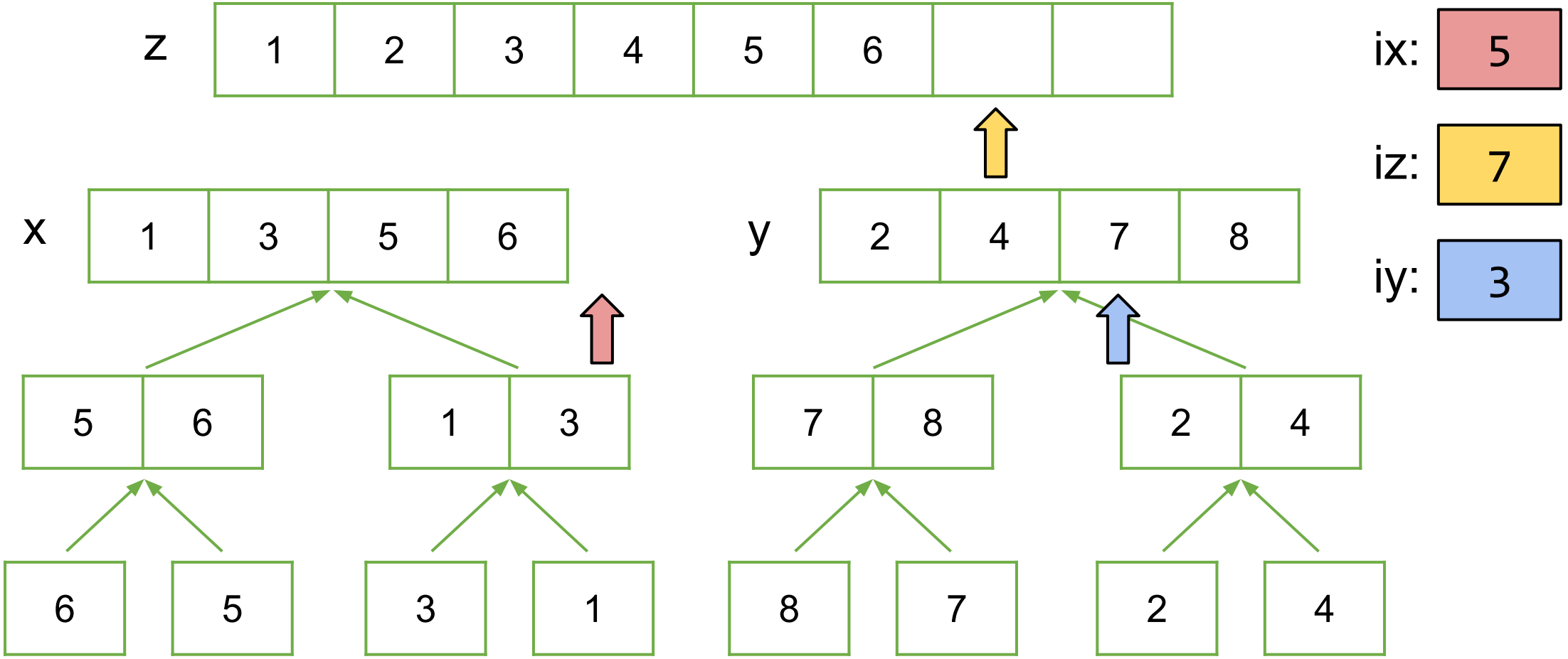
Sorting algorithms: **Merge sort - Example**



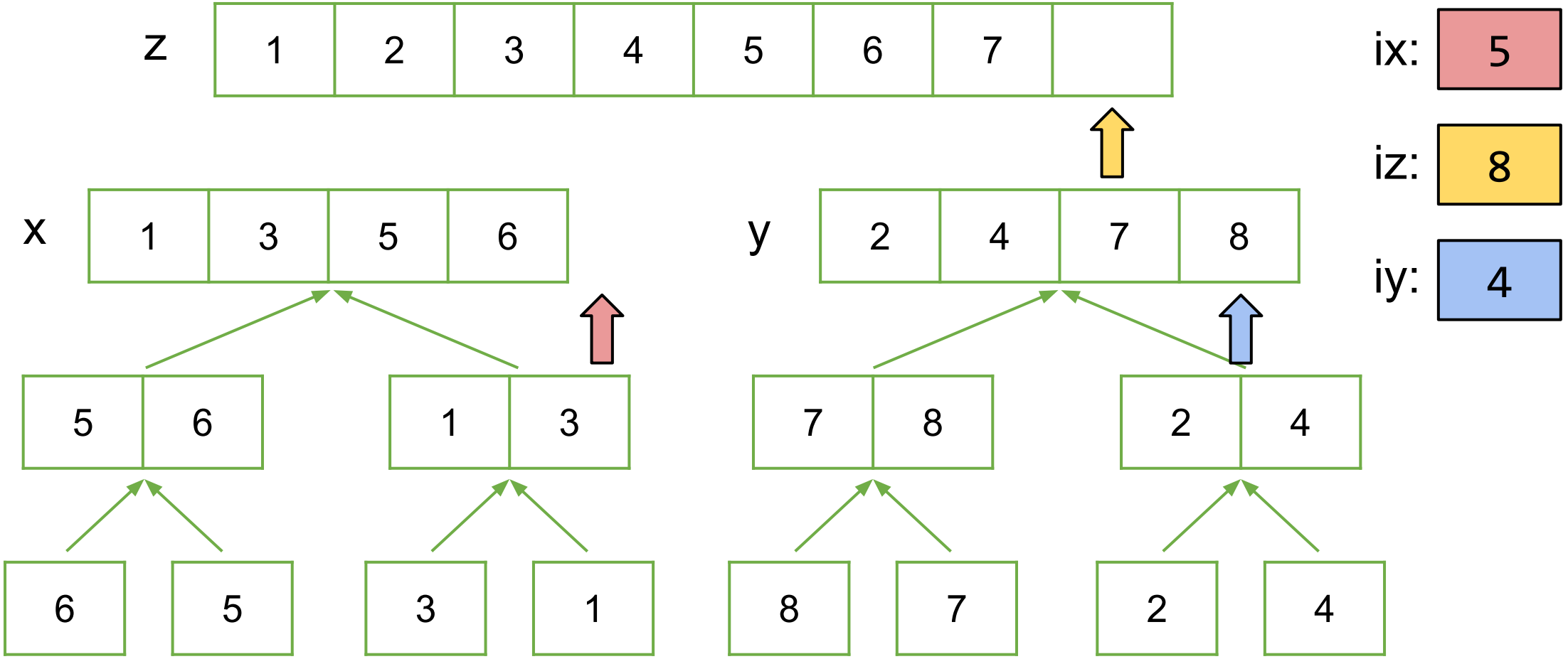
Sorting algorithms: **Merge sort - Example**



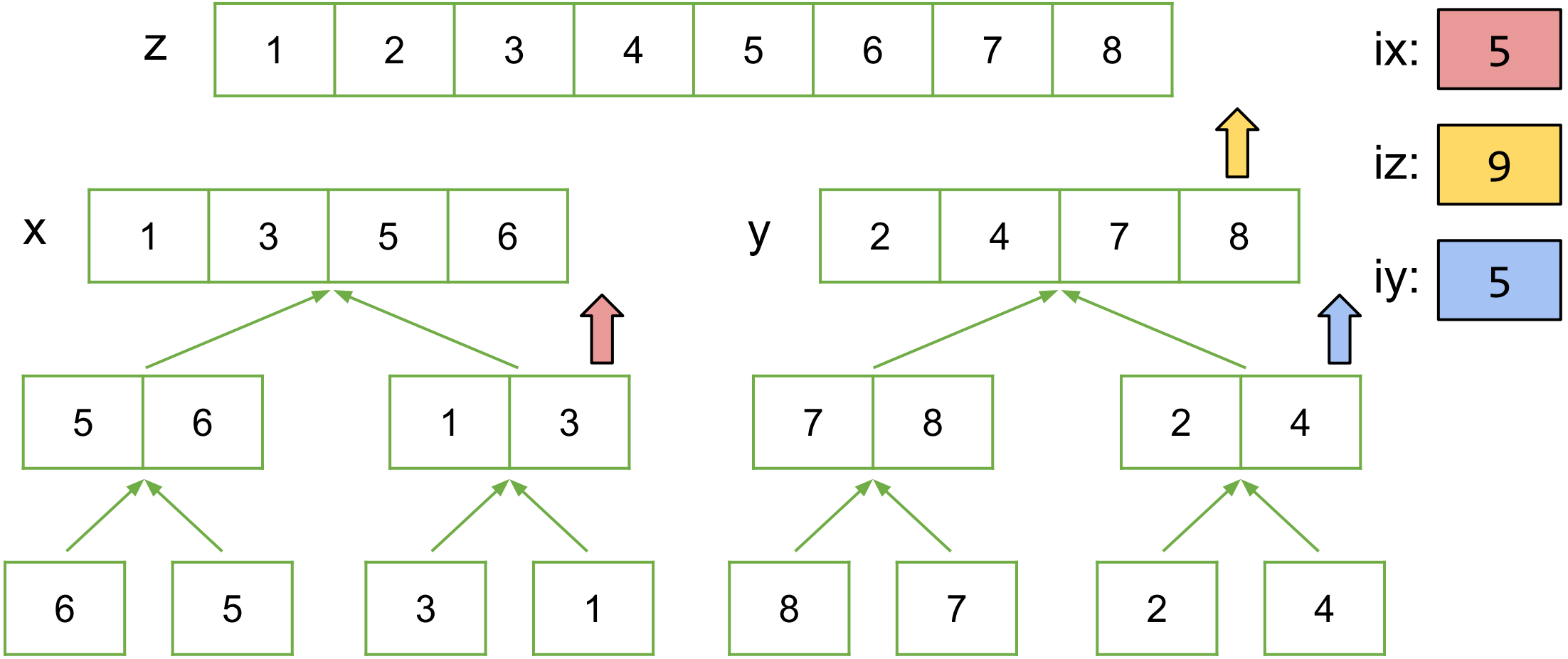
Sorting algorithms: **Merge sort - Example**



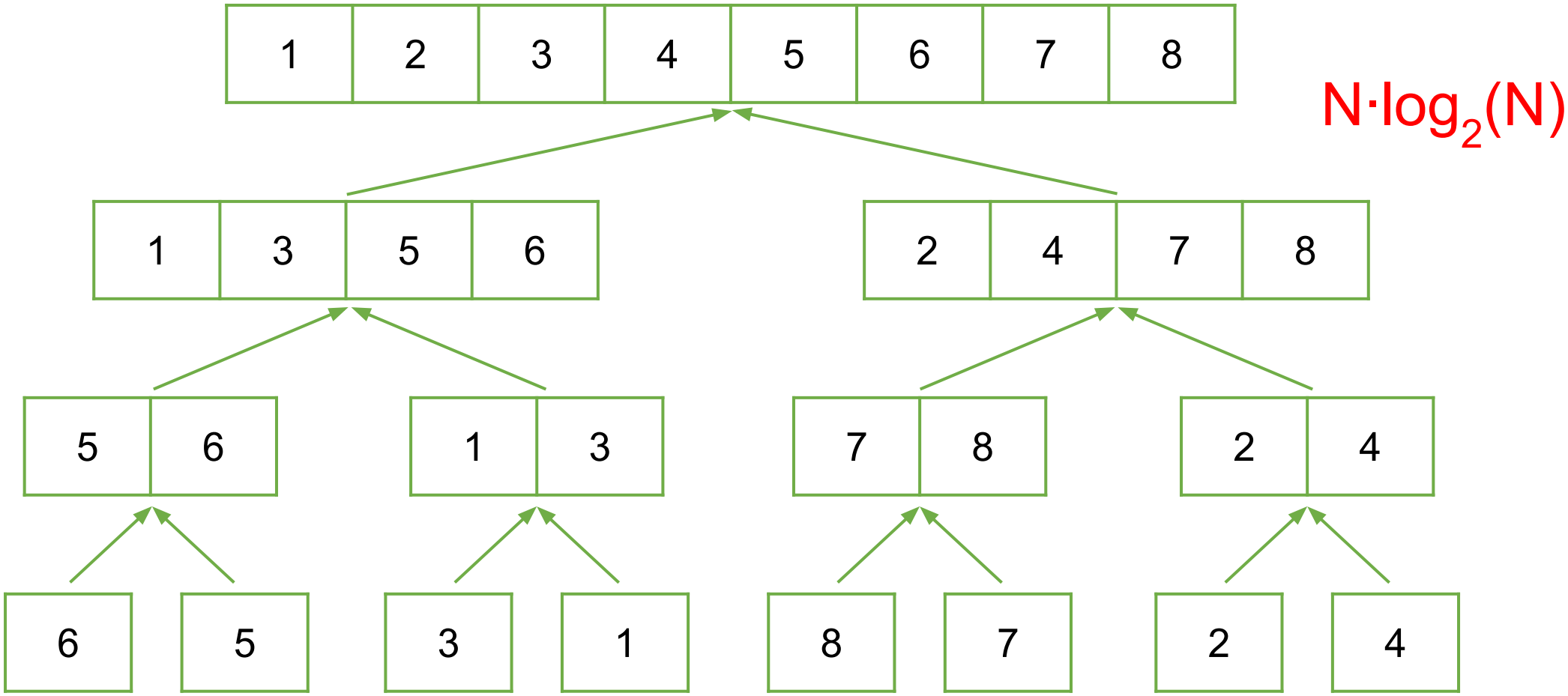
Sorting algorithms: **Merge sort - Example**



Sorting algorithms: **Merge sort - Example**



Sorting algorithms: **Merge sort - Example**



$N \cdot \log_2(N)$

Searching algorithms: **Linear Search**

```
% Linear Search
% f is index of first occurrence of value x in vector v.
% f is -1 if x not found.
k = 1;
while k<=length(v) && v(k)~=x
    k = k+1;
end
if k>length(v)
    f = -1; % signal for x not found
else
    f = k;
end
```

`n` comparisons against the target are needed in worst case, `n = length(v)`.

Searching algorithms: **Binary Search**

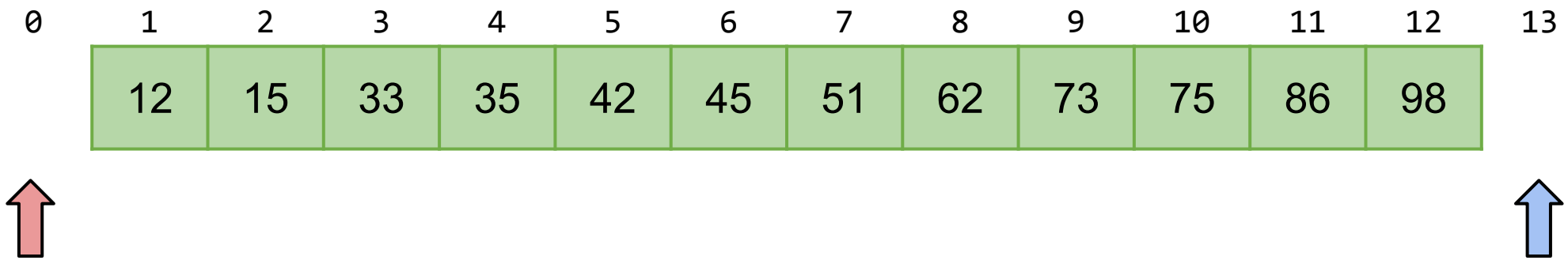
only works on sorted arrays!

An item in a sorted array of length n can be located with just $\log_2 n$ comparisons.

Searching algorithms: **Binary Search**

```
function L = binarySearch(x, v)
% Find position after which to insert x. v(1)<...<v(end)
% L is the index such that v(L)<=x<v(L+1), L=0 if x<v(1).
% If x>v(end), L=length(v) but x~=v(L).

% Maintain a search window [L..R] such that v(L)<=x<v(R)
% Since x may not be in v, initially set …
L = 0; R = length(v) + 1;

% Keep halving [L..R] until R-L is 1, always keeping v(L)<=x<v(R)
while R ~= L+1
    m = floor((L+R)/2); % middle of search window
    if v(m) <= x
        L = m;
    else
        R = m:
    end
end
```

Searching algorithms: **Binary Search - Example**



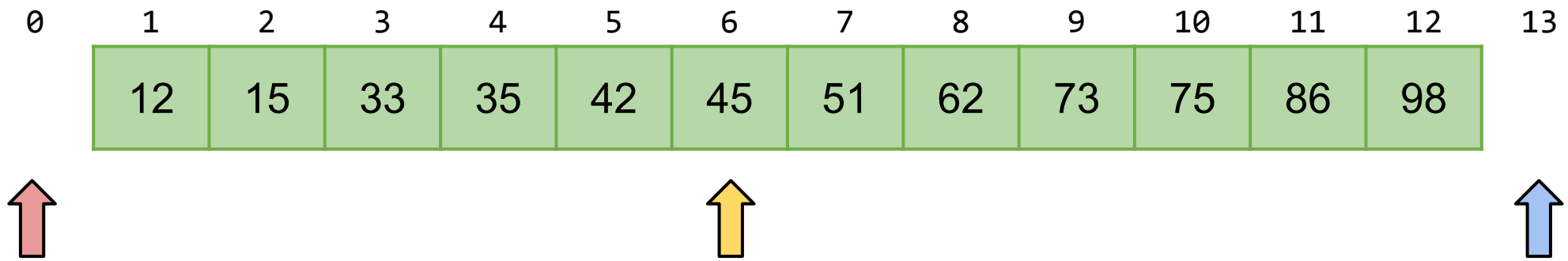
L: 0

m:

R: 13

Target x = 70

Searching algorithms: **Binary Search - Example**



L: 0

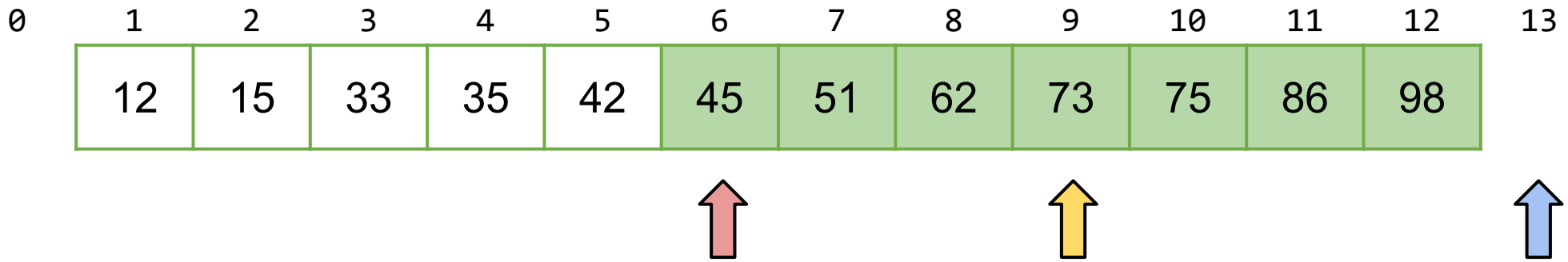
m: 6

R: 13

Target `x = 70`

`v(m) <=` x, so throw away the left half

Searching algorithms: **Binary Search - Example**



L: 6

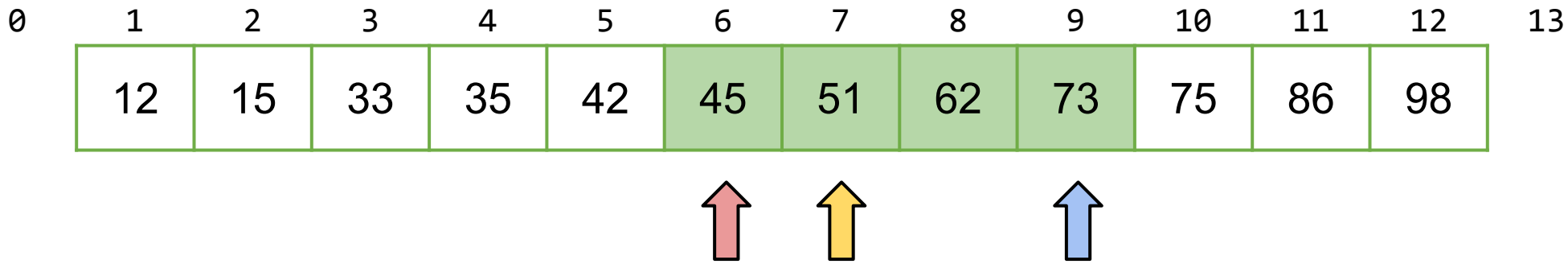
m: 9

R: 13

Target `x = 70`

`v(m) > x`, so throw away the right half

Searching algorithms: **Binary Search - Example**



L: 6

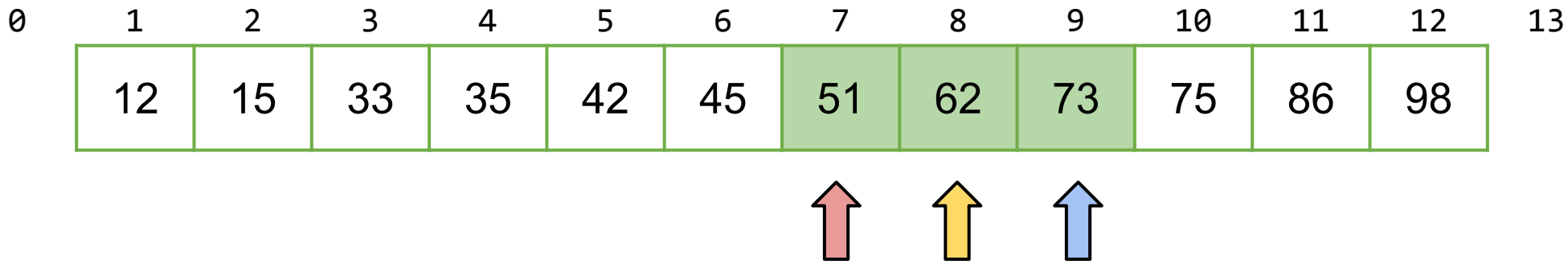
m: 7

R: 9

Target `x = 70`

`v(m) <=` x, so throw away the left half

Searching algorithms: **Binary Search - Example**



L: 7

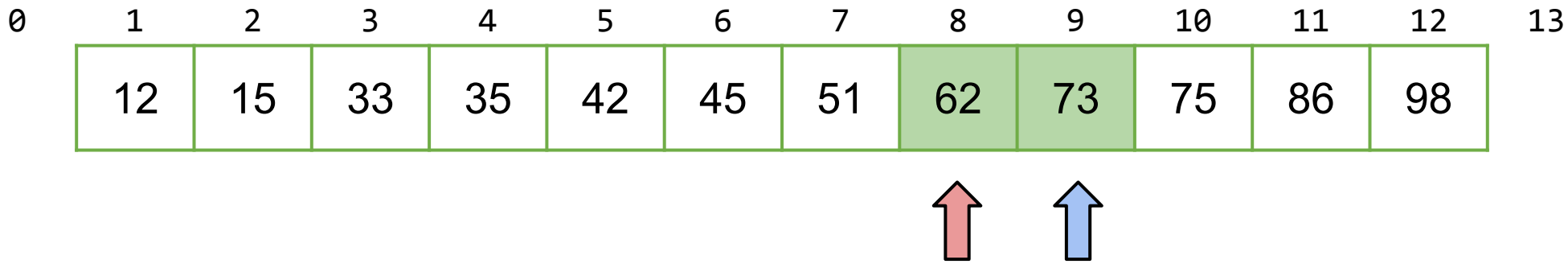
m: 8

R: 9

Target `x = 70`

`v(m) <=` x, so throw away the left half

Searching algorithms: **Binary Search - Example**



L: 8

m: 8

R: 9

Since `R-L = 1`, we're done!

Review Questions

Assume you are given a 1D cell array containing handles to Student objects. Assume now that the cell array of students is already sorted by score in ascending order. Complete the following function to find

the index of the first student whose score is at least x using a binary search strategy.

```
function k = scoreSearch ( students , x )
% Return the index `k` of the first student in `students ` whose score is at
% least as large as x . `students ` is a 1 D cell array of handles to Student
% objects , sorted in ascending order by their scores . If no students have
% a score >= x , then `k` will be 1 larger than the number of students
function s = getScore ( self ) %gets the score of the student
```

Solution

```
lb = 1; % Smallest possible index of target
k = length ( students ) + 1; % Largest possible index of target
while ( lb < k )
m = floor (( lb + k )/2);
if ( students { m }. getScore () < x )
lb = m + 1;
else
k = ( m );
end
end
```