CS 1112 Final Review

What we'll do today

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

Yay!

Objects and Classes

- **Class:** A file that specified *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
- **Protected** properties: can only be directly accessed by subclasses

Direct access means being able to access a property via statements such as:

```
<reference>.propertyName
```

Indirect access could be in the form of calling a ‘get’ method:

```
<reference>.getPropertyValue()
% We need ‘getter’ methods to access private or protected properties
```

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!

Bird is a subclass of Animal and inherits all of its protected properties.
Bird has one additional property: color.

Review Question #7

```
function idxs = greatestOverlap(iArray)
% Find the biggest pairwise overlap between Intervals in iArray.
% iArray is an array (length > 1) of Interval references.
% idxs is a vector of length 2 storing indices of the two Intervals in iArray
% that overlap the most. If there is not a pair of overlapping Intervals in
% iArray, idxs is an empty vector. Write efficient code (avoid unnecessary
% iteration
```

Potentially useful methods in the Interval class:

- `getWidth(self)` returns the difference between the left and right endpoints (i.e. the width) of the Interval object referenced by `self`.
- `overlap(self, other)` returns an Interval object whose endpoints are the points between which the two Interval objects, `self` and `other`, overlap. If they do not overlap, this method returns an empty Interval object.

Review Question #7

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Find overlap between **all possible combinations** of two Interval objects (efficiently)	Use a **nested for-loop** to check all possible combinations in `iArray`
2	Determine the **maximum** overlap	Use a **`maxWidthSoFar`** variable to keep track of the *width* of the maximum overlap we've found so far
3	Store the indices from `iArray` of the Intervals which overlap the most	Update `idxs` when `maxWidthSoFar` changes
4	**If no Intervals overlap**, `idxs` is an empty vector	`idxs` should be **initialized as empty**, and only filled if the width of the overlap between any two Intervals is greater than 0

Review Question #7: **Solution**

```
function idxs = greatestOverlap(iArray)

    idxs = [];
    maxWidth = 0;
    n = length(iArray);
    for i = 1:n-1          % Notice this loop ends at n-1
        for j = i+1:n      % Notice this loop started at i+1
            olap = iArray(i).overlap(iArray(j));
            if ~isempty(olap) && olap.getWidth() > maxWidth
                maxWidth = olap.getWidth();
                idxs = [i j];
            end
        end
    end
```

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 By the *reverse* of a string `s` we mean the string obtained by reversing the order of the characters in `s`. Thus, if `s = 'abcde'`, then `'edcba'` is its reverse. (a) Write a nonrecursive function `t = Reverse(s)` that does this using a loop. (b) 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 = ReverseR(s)` that does this. (c) Compare the execution times for the two implementations.

Recursion Example: Review Question #10

P14.1.3 By the *reverse* of a string `s` we mean the string obtained by reversing the order of the characters in `s`. Thus, if `s = 'abcde'`, then `'edcba'` is its reverse. (a) Write a nonrecursive function `t = Reverse(s)` that does this using a loop. (b) 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 = ReverseR(s)` that does this. (c) Compare the execution times for the two implementations.

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’

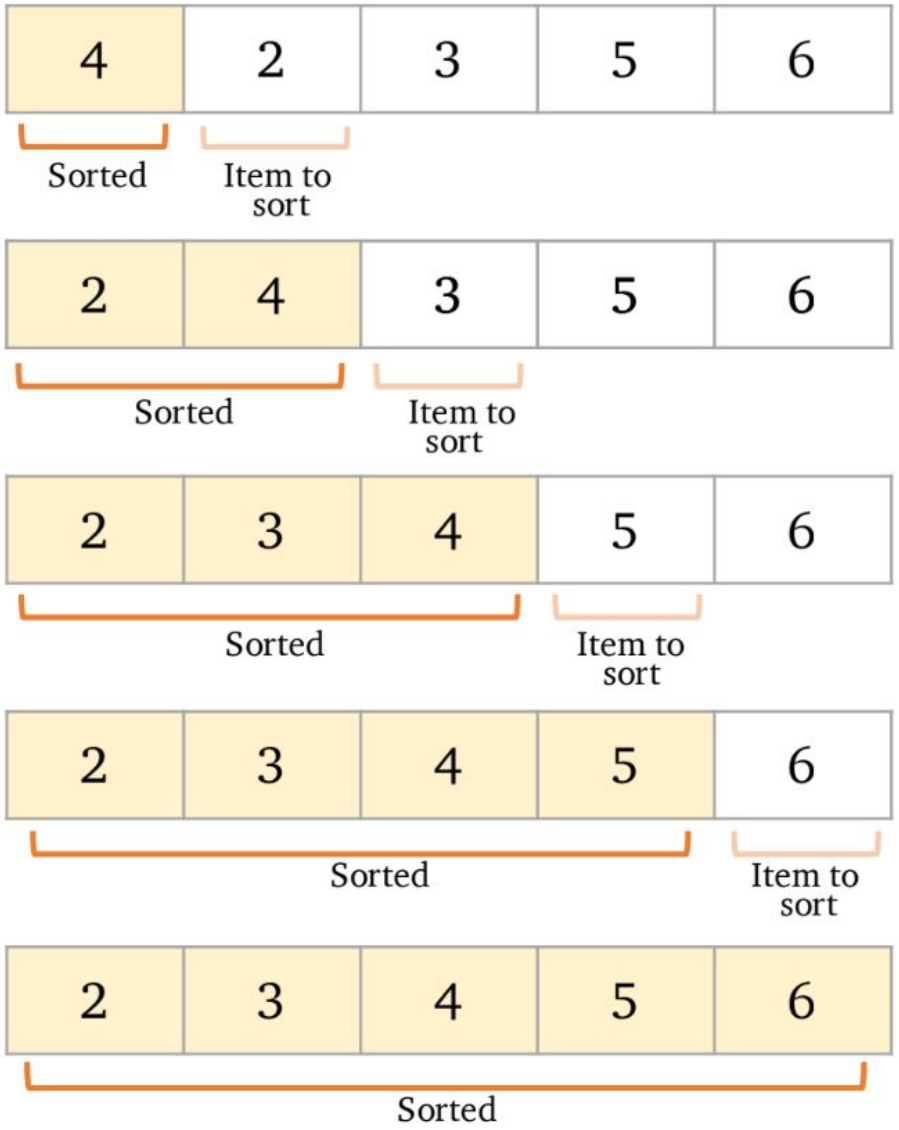
```
function t = Reverse(s)
    n = length(s);
    if n == 1     % base case: if n == 1, s is the reverse of itself
        t = s;
    else          % reverse the last n-1 characters of s, append to s(1)
        t = [Reverse(s(2:n)), s(1)];
    end
```

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:

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

Review Question #1b

```
1.  (Point struct and sorting algorithm)

% Given a structure array Pts where each structure has two fields, x and y,
% sort Pts so that the structures are in the order of
% increasing distance from (0,0)
% Write two different scripts to solve this problem:
%  (a) Make effective use of built-in function sort.
%  (b) Use the INSERTION SORT algorithm; do not use built-in function sort.
```

Review Question #1b: **Solution**

```
n = length(Pts);


for i = 1:n-1


    j = i;
    need2swap =
    while need2swap


        % swap elements in Pts array
        tempP = Pts(j); Pts(j) = Pts(j+1); Pts(j+1) = tempP;
        j = j-1;
        need2swap =
    end
end
```

Review Question #1b: **Solution**

```
n = length(Pts);
dis = zeros(1,n); % dis stores distance of each point from origin
dis(1) = sqrt(Pts(1).x^2 + Pts(1).y^2);
for i = 1:n-1
    % Sort dis(1:i+1) given that dis(1:i) is sorted
    dis(i+1) = sqrt(Pts(i+1).x^2 + Pts(i+1).y^2);
    j = i;
    need2swap =
    while need2swap


        % swap elements in Pts array
        tempP = Pts(j); Pts(j) = Pts(j+1); Pts(j+1) = tempP;
        j = j-1;
        need2swap =
    end
end
```

Review Question #1b: **Solution**

```
n = length(Pts);
dis = zeros(1,n); % dis stores distance of each point from origin
dis(1) = sqrt(Pts(1).x^2 + Pts(1).y^2);
for i = 1:n-1
    % Sort dis(1:i+1) given that dis(1:i) is sorted
    dis(i+1) = sqrt(Pts(i+1).x^2 + Pts(i+1).y^2);
    j = i;
    need2swap = dis(j+1) < dis(j);
    while need2swap
        % swap elements in dis array
        tempD = dis(j); dis(j) = dis(j+1); dis(j+1) = tempD;
        % swap elements in Pts array
        tempP = Pts(j); Pts(j) = Pts(j+1); Pts(j+1) = tempP;
        j = j-1;
        need2swap = j>0 && dis(j+1)<dis(j);
    end
end
```

Review Question #1b: **Solution**

```
n = length(Pts);
dis = zeros(1,n); % dis stores distance of each point from origin
dis(1) = sqrt(Pts(1).x^2 + Pts(1).y^2);
for i = 1:n-1
    % Sort dis(1:i+1) given that dis(1:i) is sorted
    dis(i+1) = sqrt(Pts(i+1).x^2 + Pts(i+1).y^2);
    j = i;
    need2swap = dis(j+1) < dis(j);
    while need2swap
        % swap elements in dis array
        tempD = dis(j); dis(j) = dis(j+1); dis(j+1) = tempD;
        % swap elements in Pts array
        tempP = Pts(j); Pts(j) = Pts(j+1); Pts(j+1) = tempP;
        j = j-1;
        need2swap = j>0 && dis(j+1)<dis(j);
    end
end
```

Sorting algorithms: **Merge sort**

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

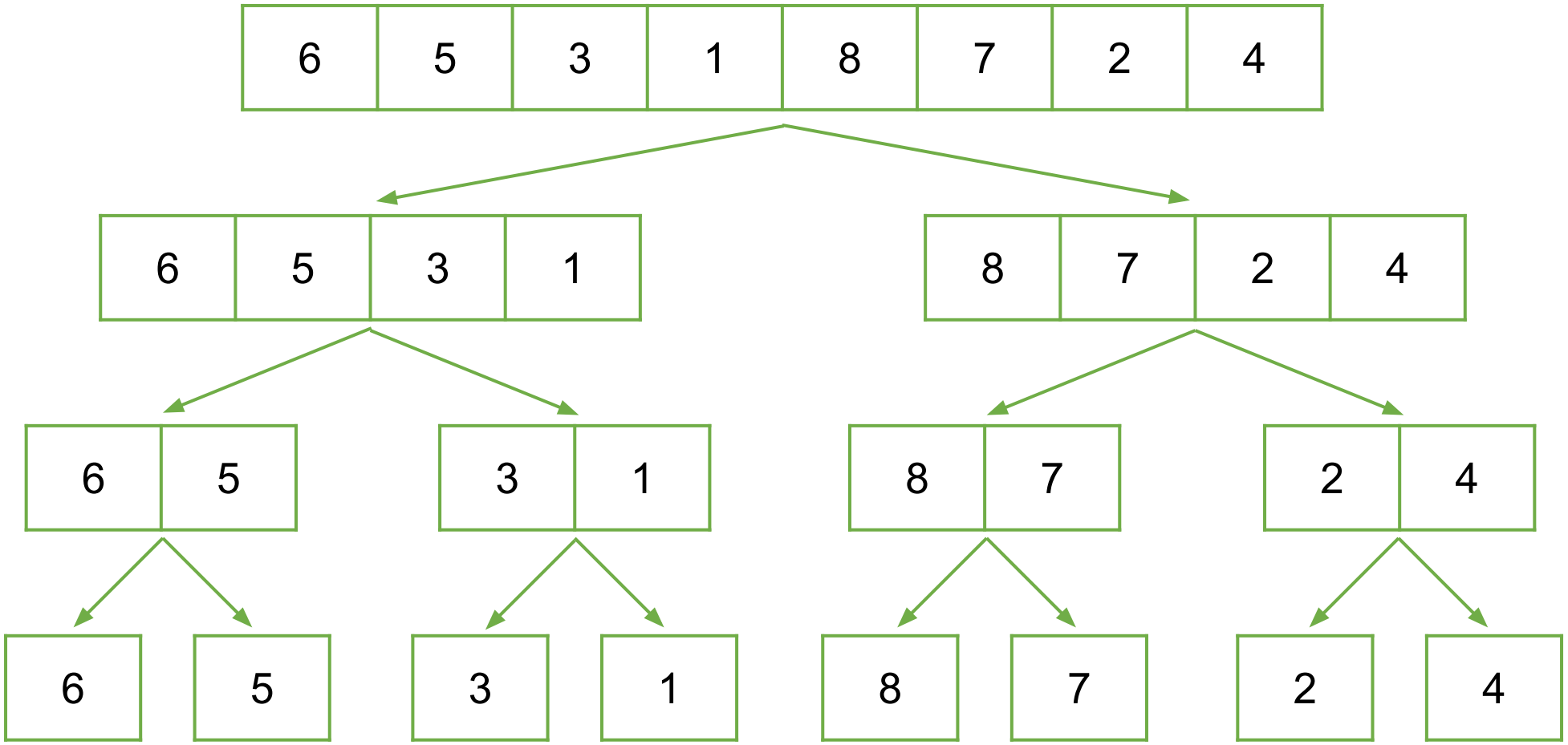
- Dividing the vector into *n* arrays of 1 component each
- Merge adjacent components in sorted order to produce `ceil(n/2)` arrays of length 2
- Merge adjacent vectors of length 2 in sorted order to produce `ceil(n/4)` arrays of length 4

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

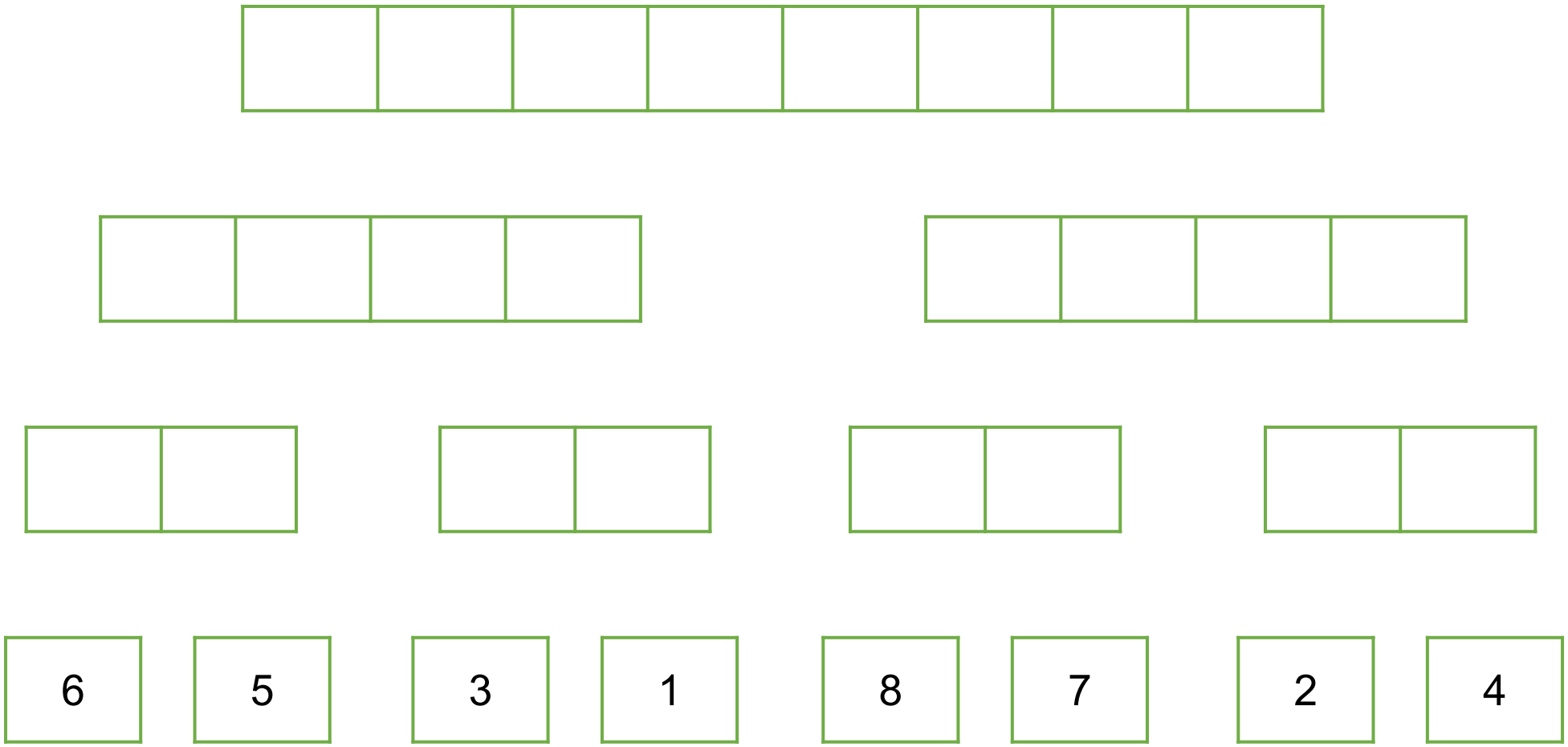
```
function y = mergeSort(x)
% x is a vector. y is a vector
% consisting of the values in x
% sorted from smallest to largest

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
```

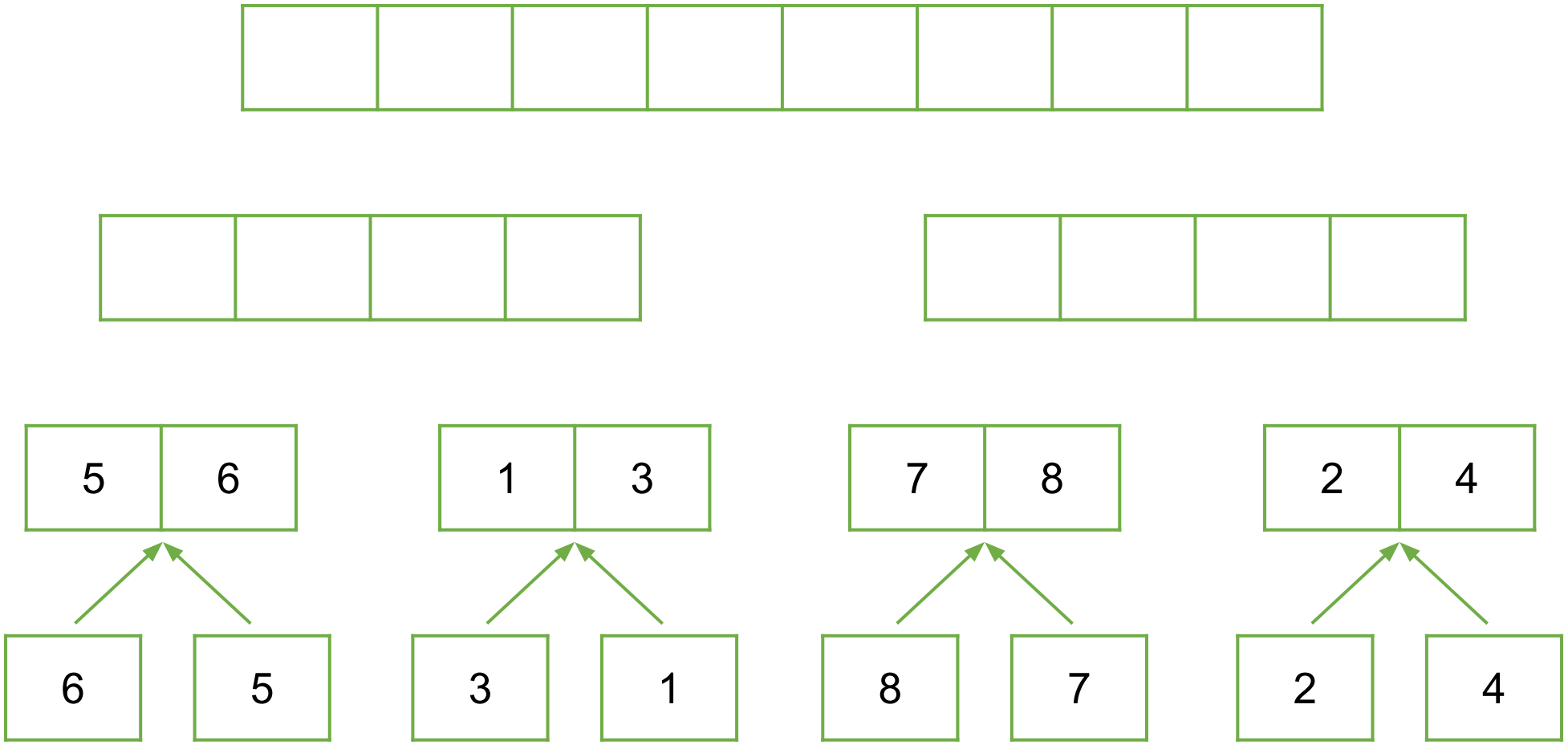
Sorting algorithms: **Merge sort - Example**



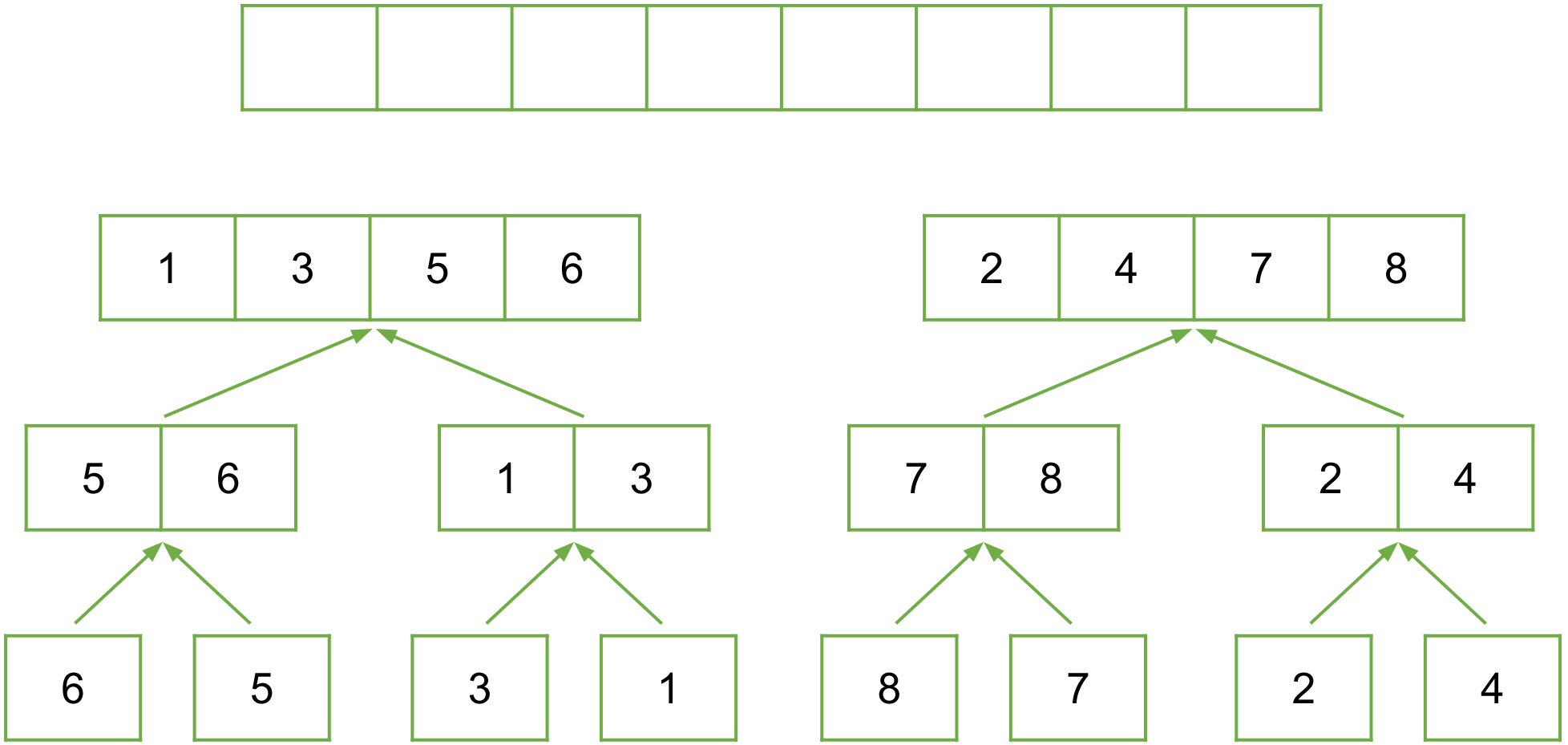
Sorting algorithms: **Merge sort - Example**



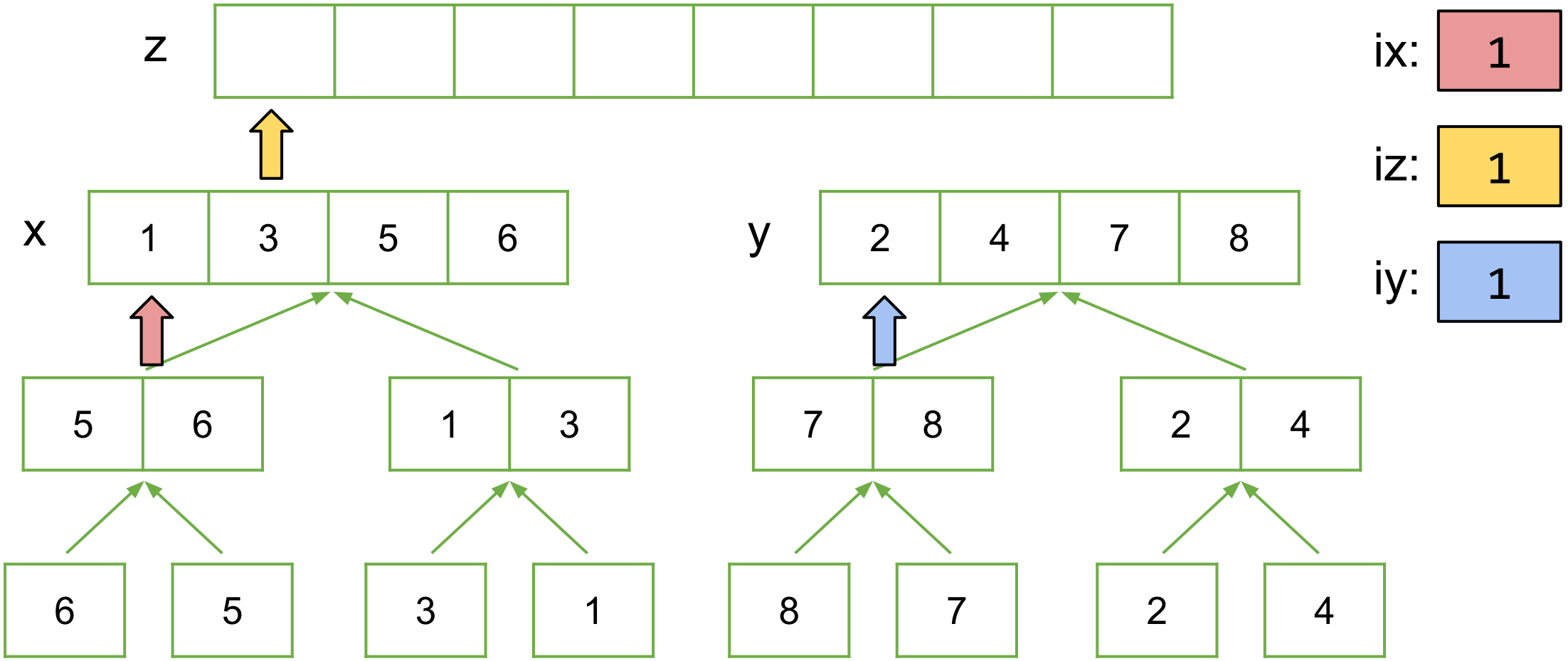
Sorting algorithms: **Merge sort - Example**



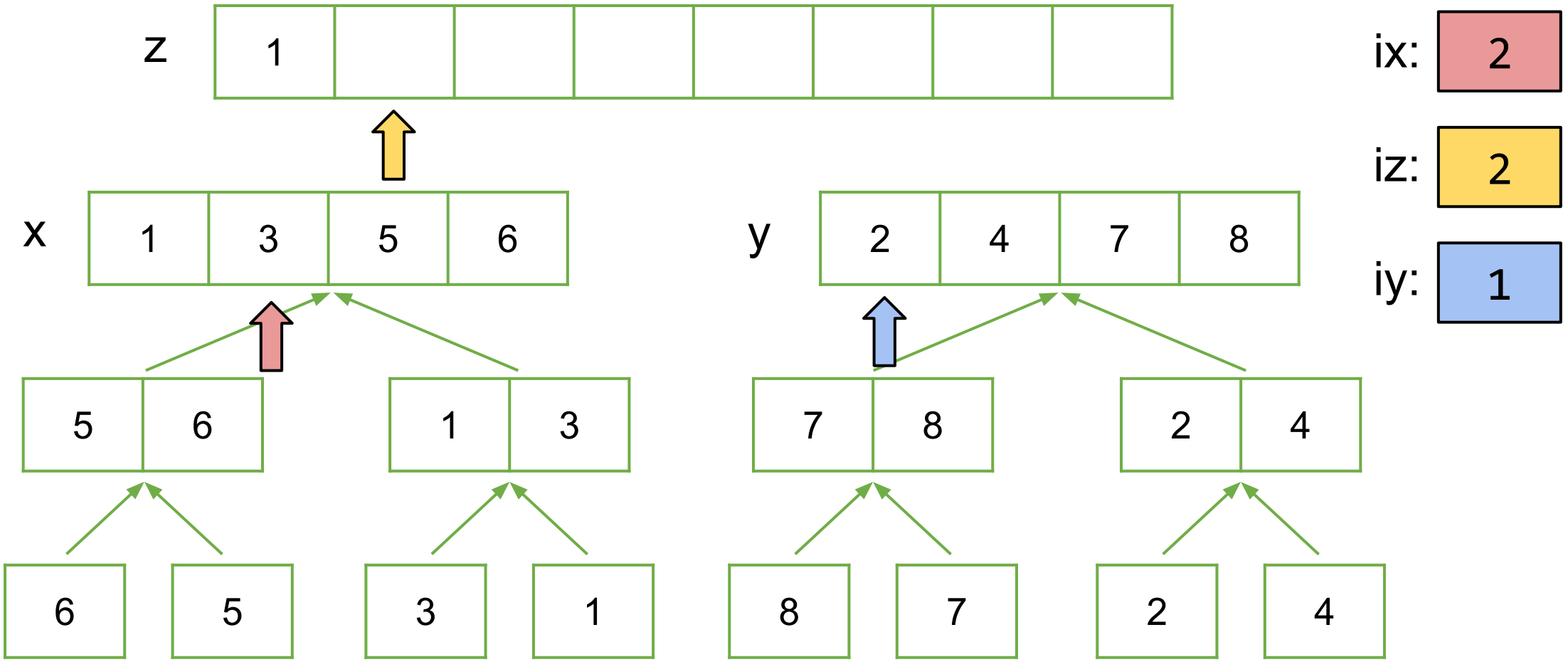
Sorting algorithms: **Merge sort - Example**



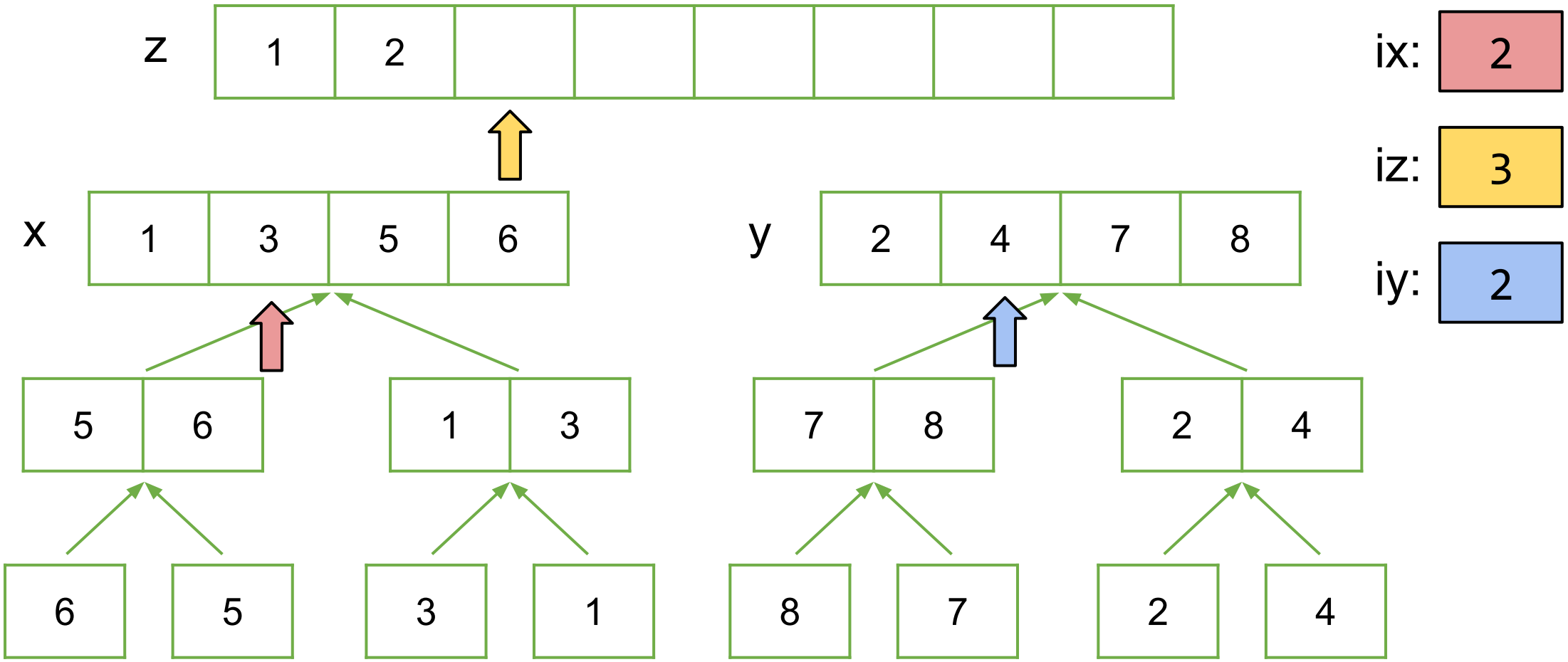
Sorting algorithms: **Merge sort - Example**



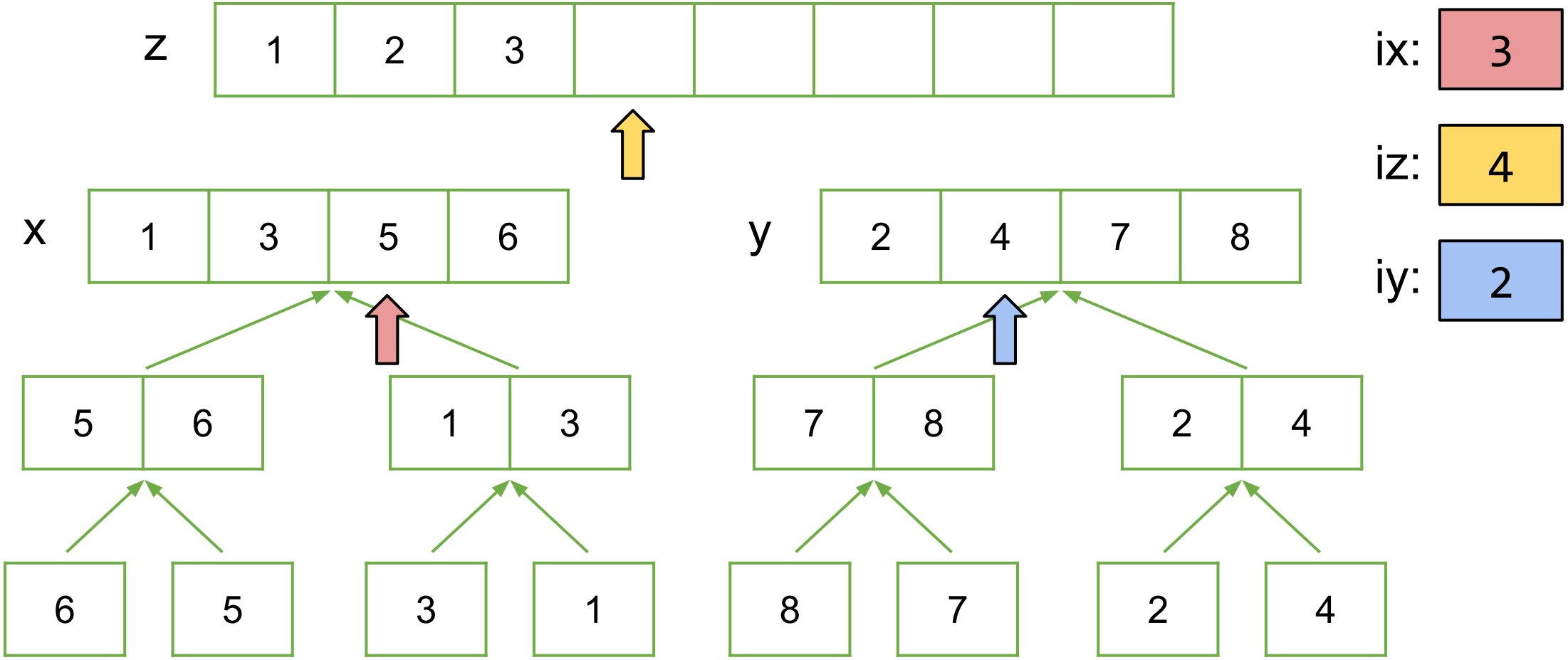
Sorting algorithms: **Merge sort - Example**



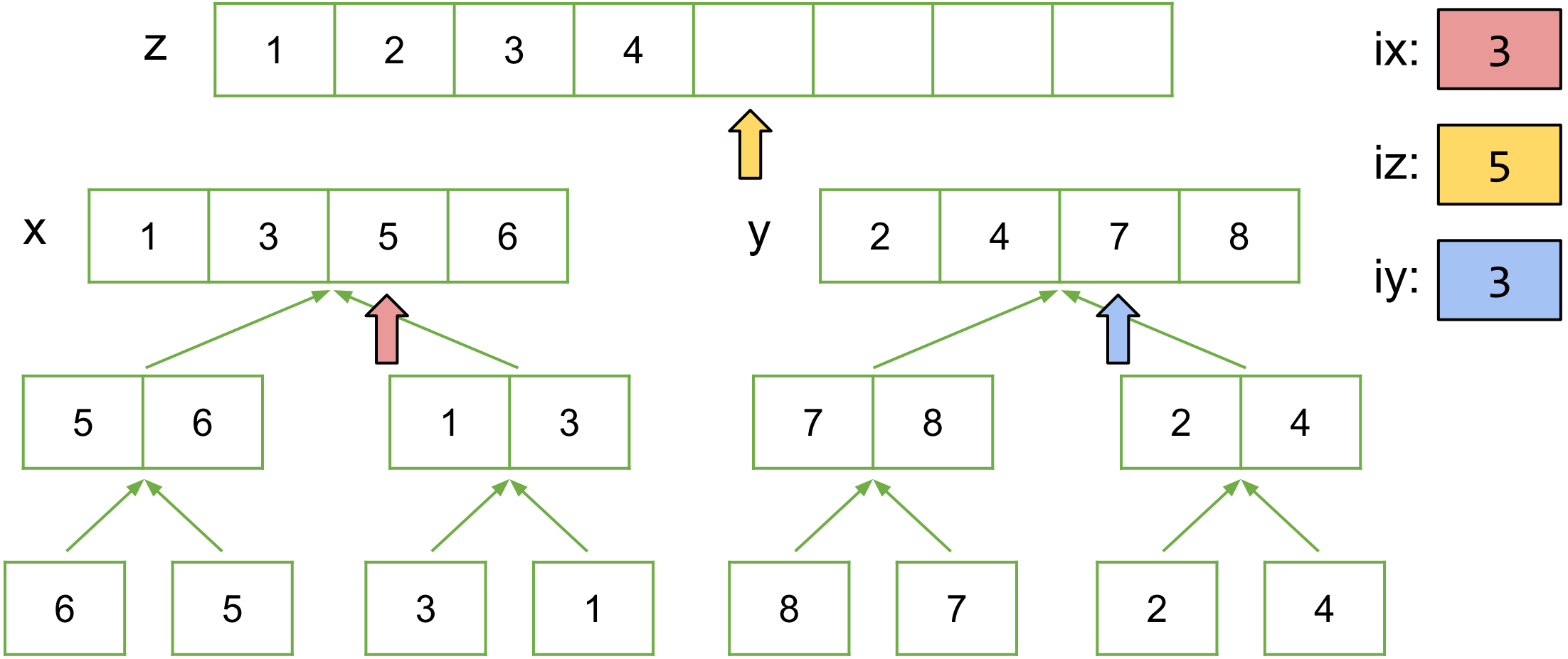
Sorting algorithms: **Merge sort - Example**



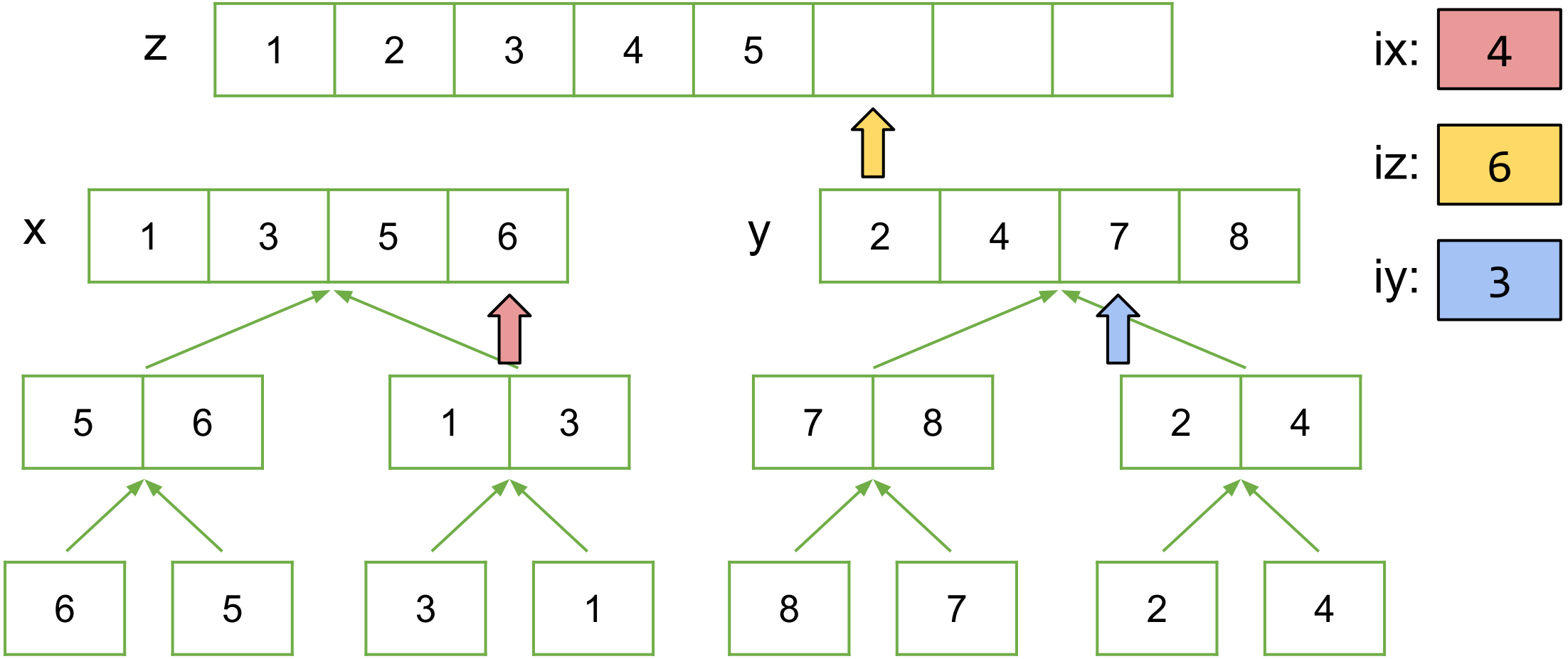
Sorting algorithms: **Merge sort - Example**



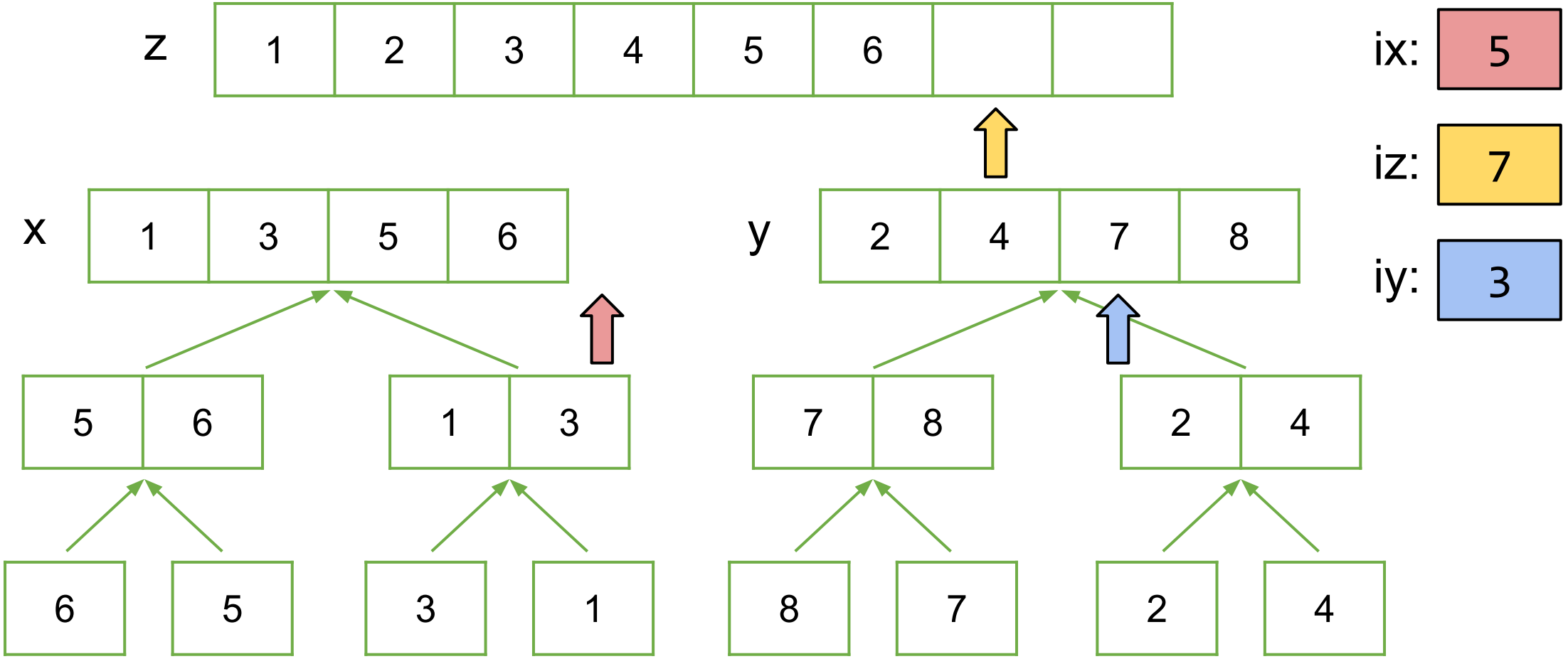
Sorting algorithms: **Merge sort - Example**



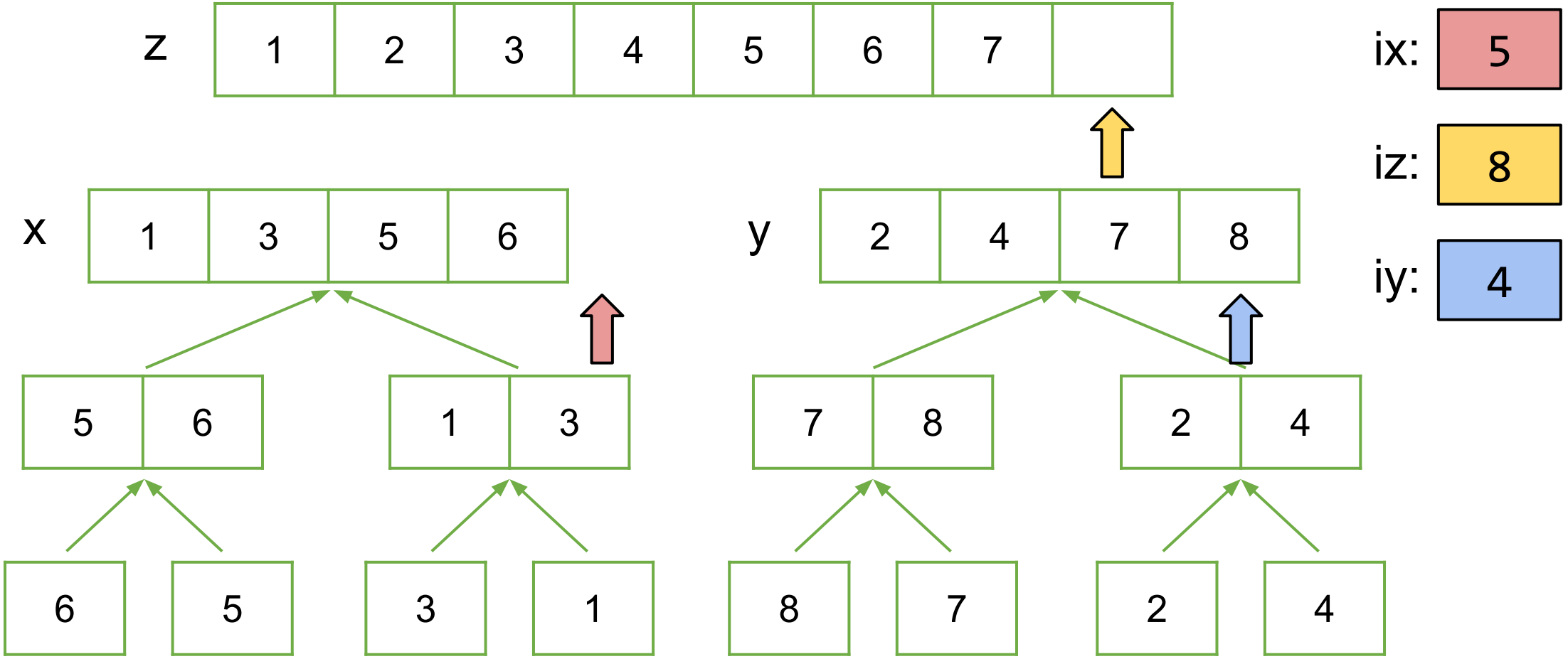
Sorting algorithms: **Merge sort - Example**



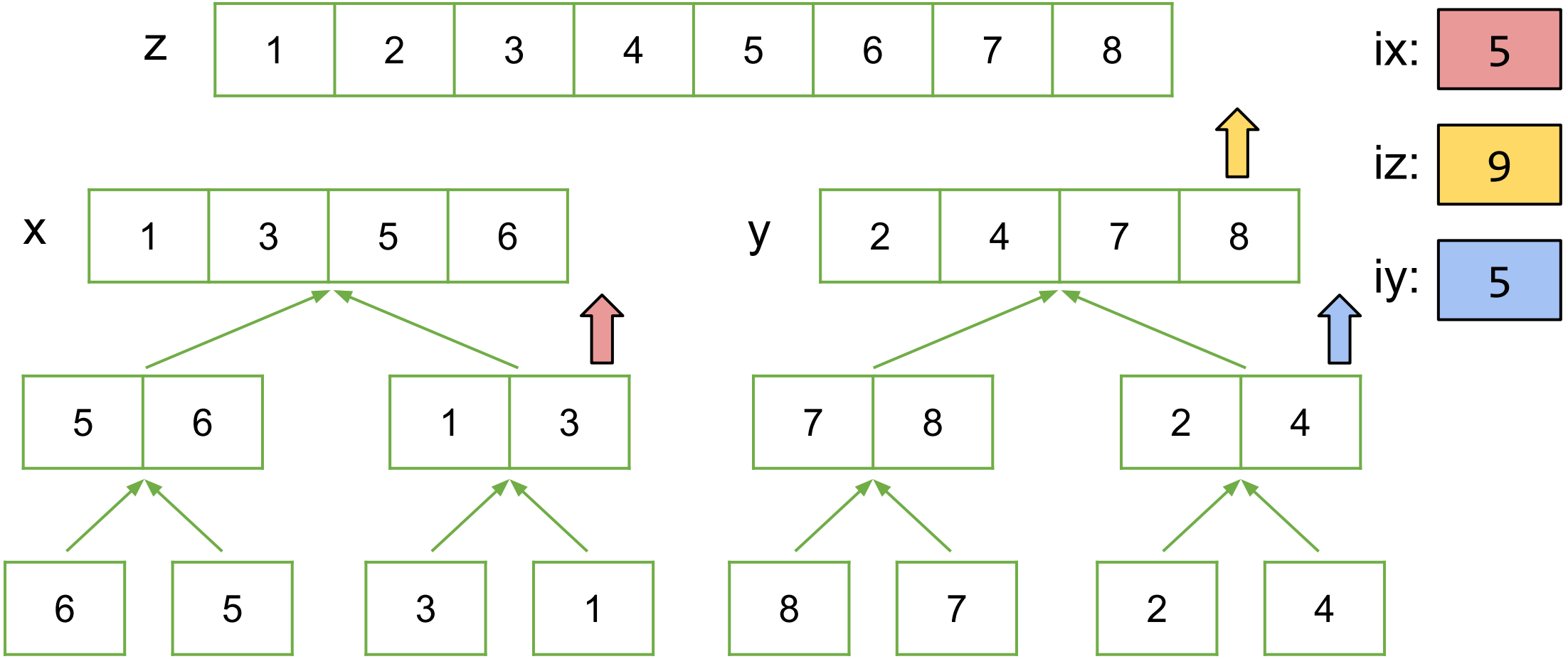
Sorting algorithms: **Merge sort - Example**



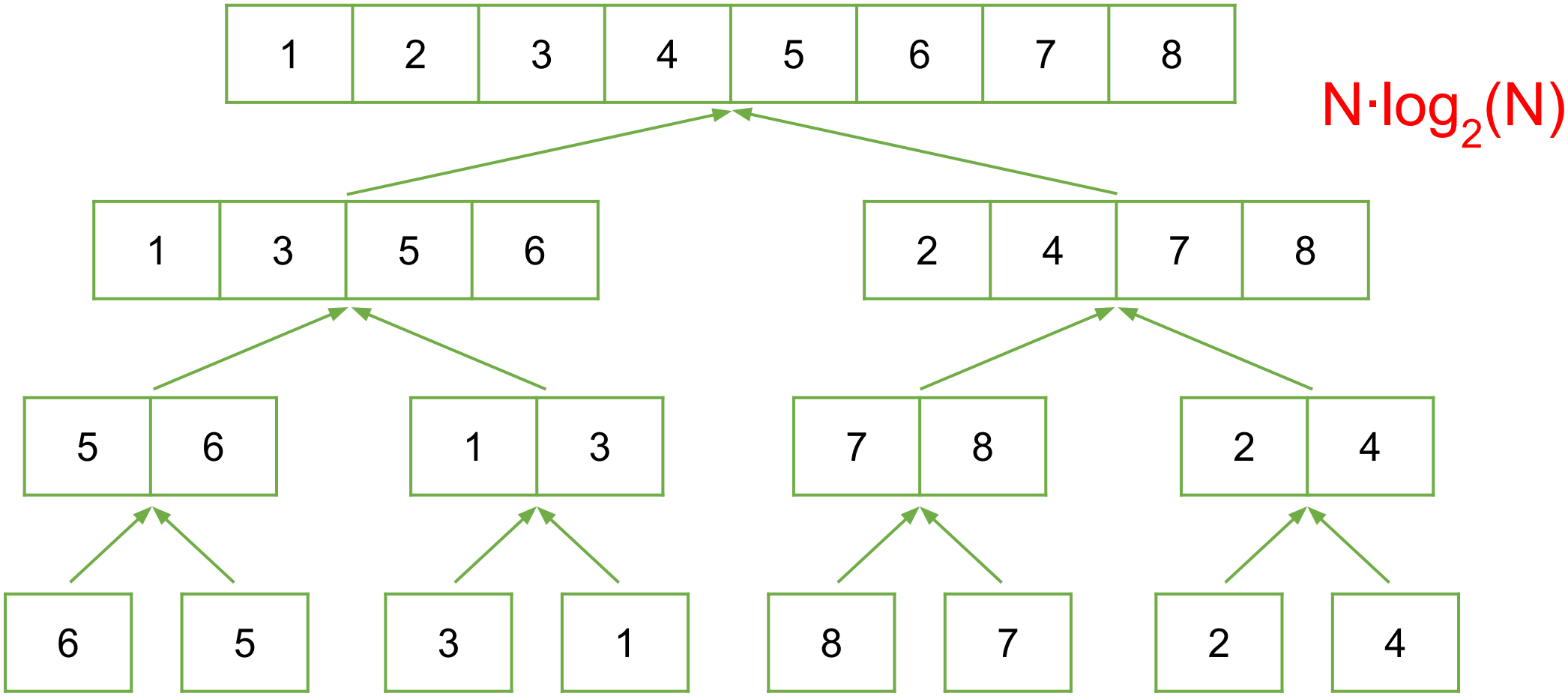
Sorting algorithms: **Merge sort - Example**



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**

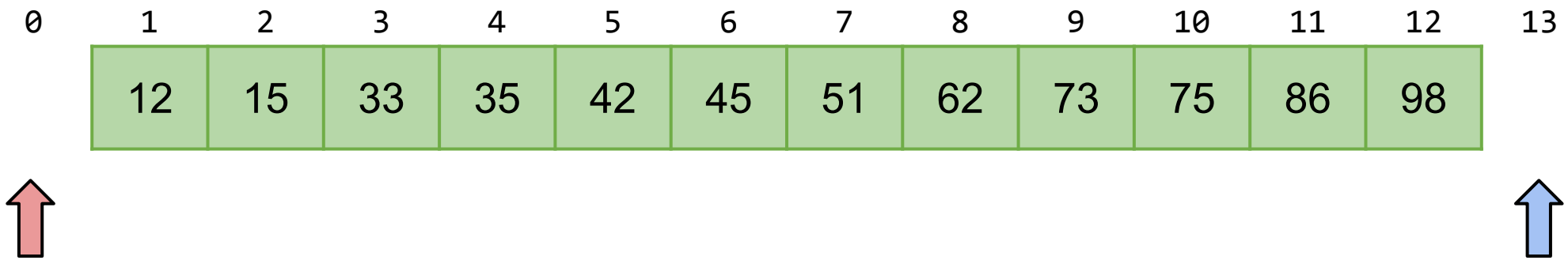
only works on sorted arrays!

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**



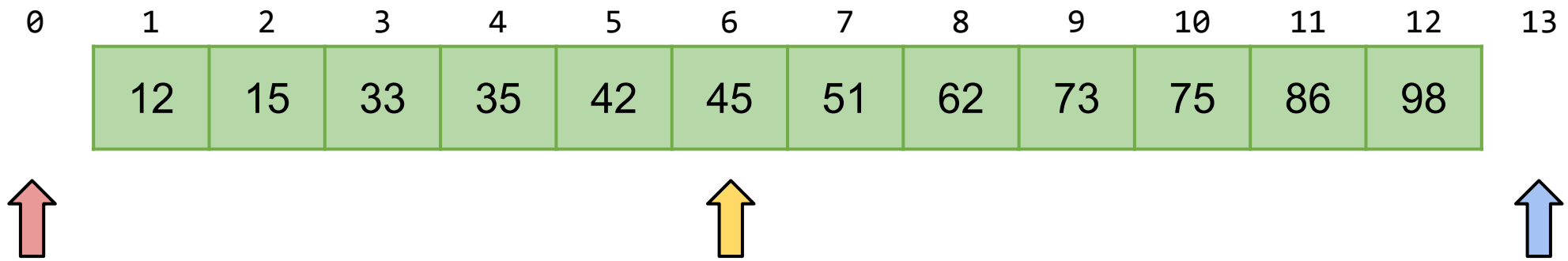
L: 0

m:

R: 13

Target x = 70

Searching algorithms: **Binary Search - Example**



L: 0

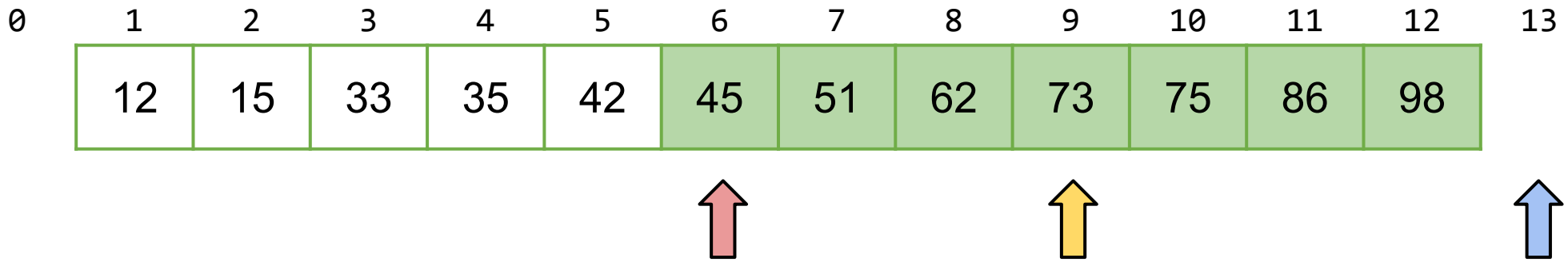
m: 6

R: 13

Target `x = 70`

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

Searching algorithms: **Binary Search - Example**



L: 6

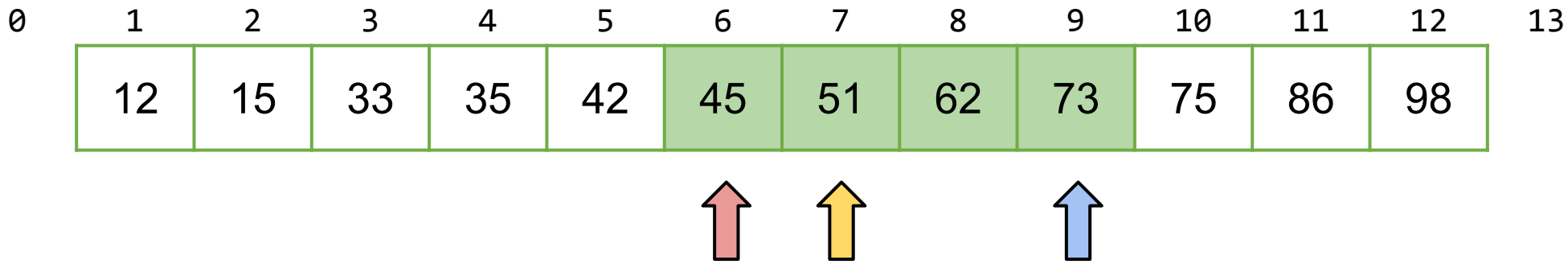
m: 9

R: 13

Target `x = 70`

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

Searching algorithms: **Binary Search - Example**



L: 6

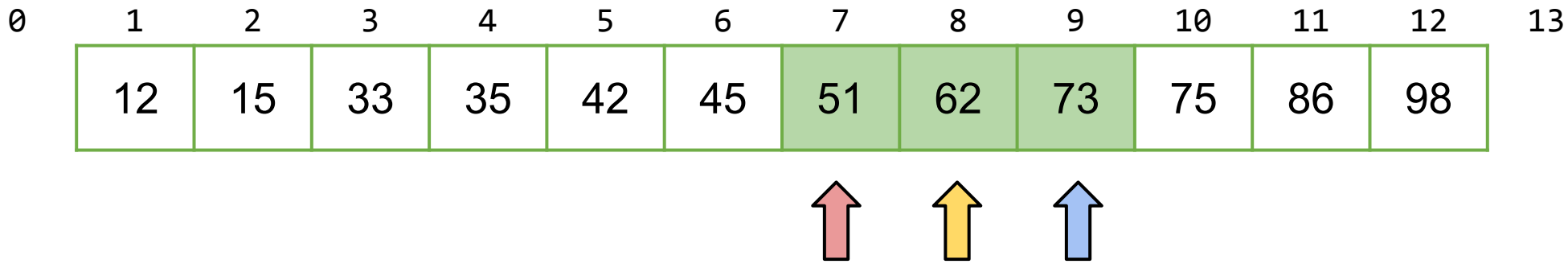
m: 7

R: 9

Target `x = 70`

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

Searching algorithms: **Binary Search - Example**



L: 7

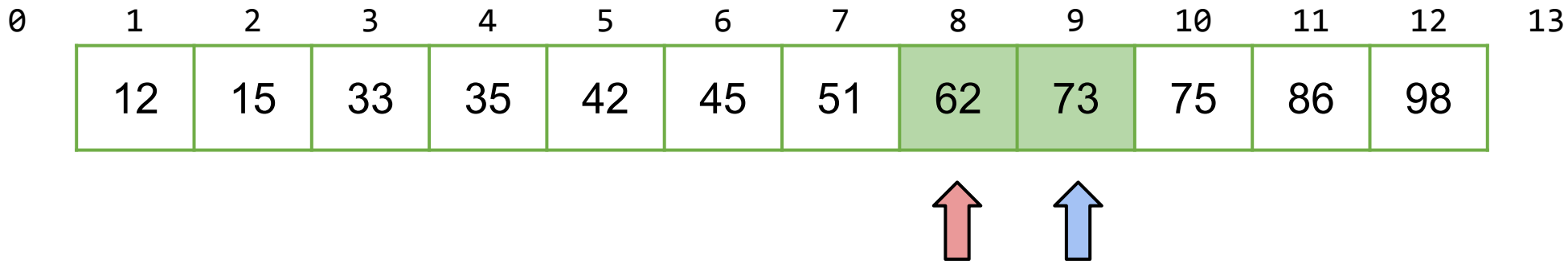
m: 8

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!

Write Efficient Code

- Instead of looping through the whole array, can we stop earlier? (Lab 6 Problem 3)

  Write a function `vectorQuery(v,n,r)` to determine whether the number `r` appears in the first `n` components of vector `v`. The function returns 1 if `r` is in the first `n` components of `v` and 0 otherwise. Your function assumes that `v` is a vector of numbers, `n` is a positive integer, and `r` is a number. Use a loop to do the search. (Do not use find or vectorized code.) Make sure that the loop index doesn't go "out of bounds" (if `n` is greater than the length of vector `v`).

- For a nested for loop problem (Lab 14 problem 2)
  - Can we modify the loop header to save iterations?
  - Can we precompute/move computation to outer for-loop to save operations?