Big-O and sorting

Insert sort

```
List<Integer> result = new ArrayList<Integer>();

//Insert all numbers into the result collection at the
for(int num : numbers) {                                          n

    //Loop until end of the result loop, or the pos whe
    for (int i = 0; i <= result.size(); i++) {                    n*n
        if(i== result.size() || result.get(i) > num) {            n*n
            //Insert num at this position
            result.add(i, num);                                   n..n*n
            break;
        }
    }
}                                                                 Therefore O(n^2)
```

Bubble sort: O(n^2)

```
List<Integer> result = new ArrayList<Integer>(numbers);

//Move high values toward the end
for(int i = 0; i<result.size()-1; i++) {

    //Swap incorrectly positioned values
    if(result.get(i) > result.get(i+1)) {
        Collections.swap(result,i,i+1);

        //Restart at beginning of the collection
        i = -1; //i++ happens after this so will become
    }
}
```

n times

n

1..n

1..n

O(n)? WRONG!

Bubble sort: O(n^2)

```
List<Integer> result = new ArrayList<Integer>(numbers);

//Move high values toward the end
for(int i = 0; i<result.size()-1; i++) {

    //Swap incorrectly positioned values
    if(result.get(i) > result.get(i+1)) {
        Collections.swap(result,i,i+1);

        //Restart at beginning of the collection
        i = -1; //i++ happens after this so will become
    }
}
```

n*(1..n) times

n*(1..n)

1..n

1..n

O(n^2)

Bogo sort

```
List<Integer> result = new ArrayList<Integer>(numbers);

//Shuffle collection and check if it is sorted
while(!isSorted(result)) {
    Collections.shuffle(result);
}
```

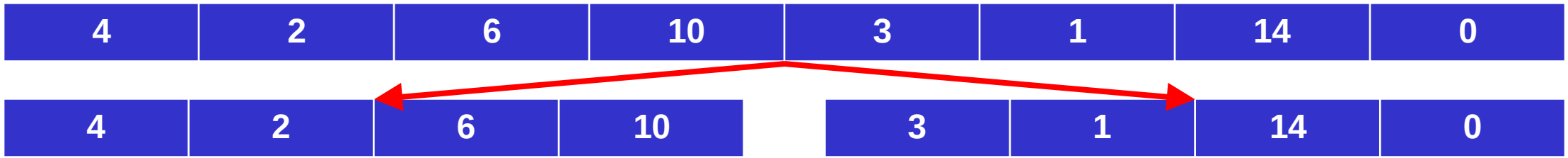
How

Prob

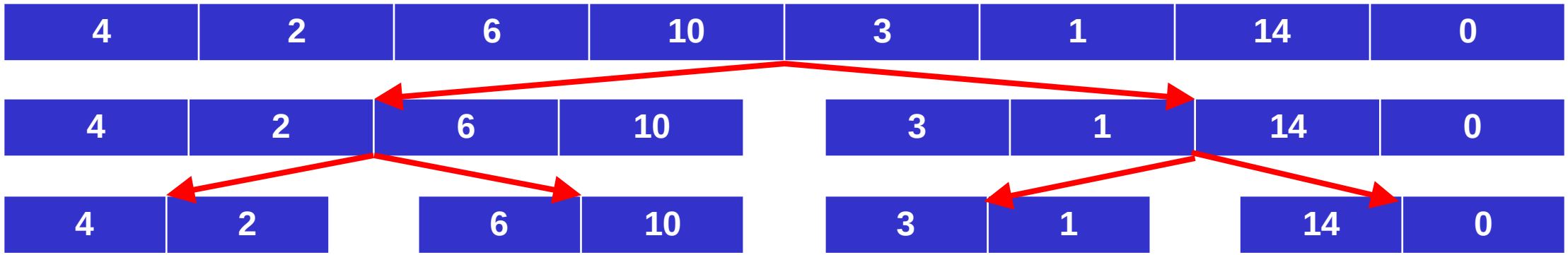
so average case is O((n+1)!), worst case O(infinite), best case O(n)

Merge sort:

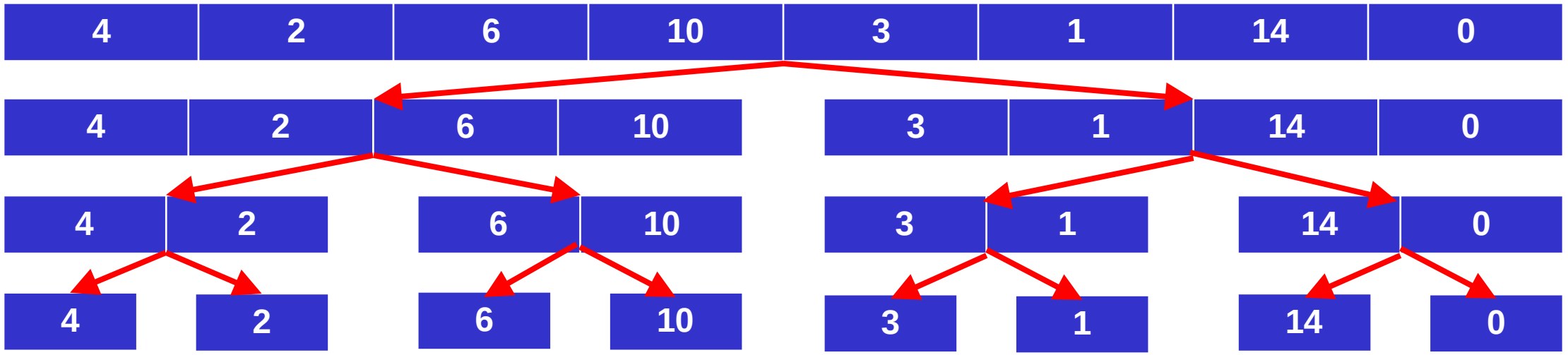
Merge sort: Split recursively



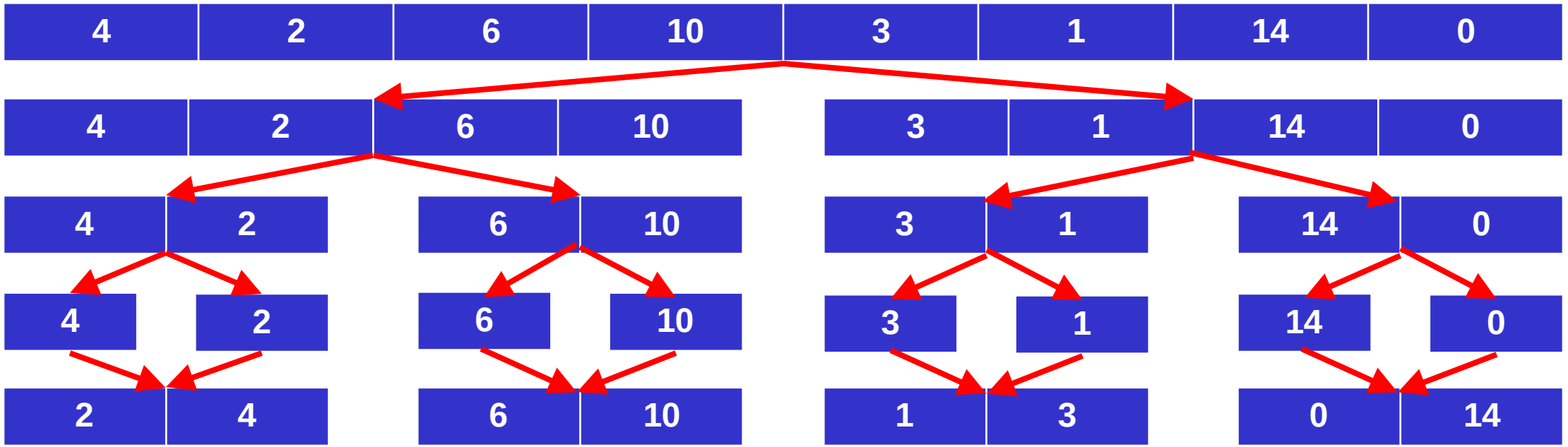
Merge sort: Split recursively



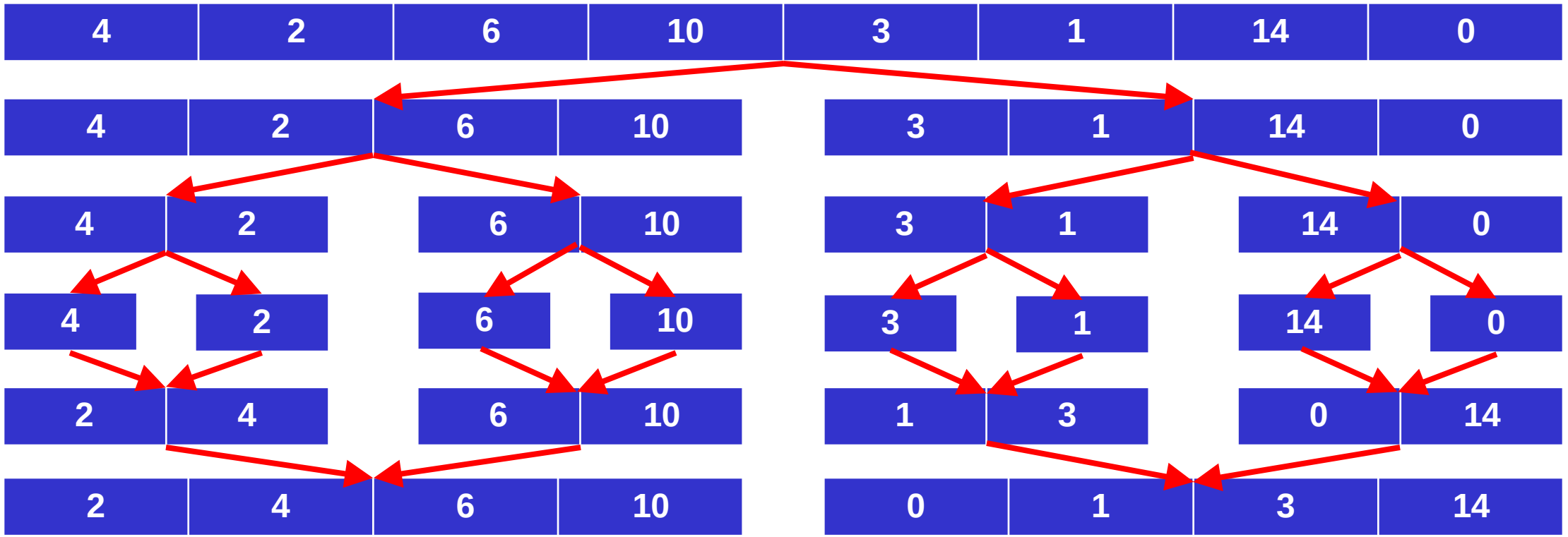
Merge sort: Split recursively



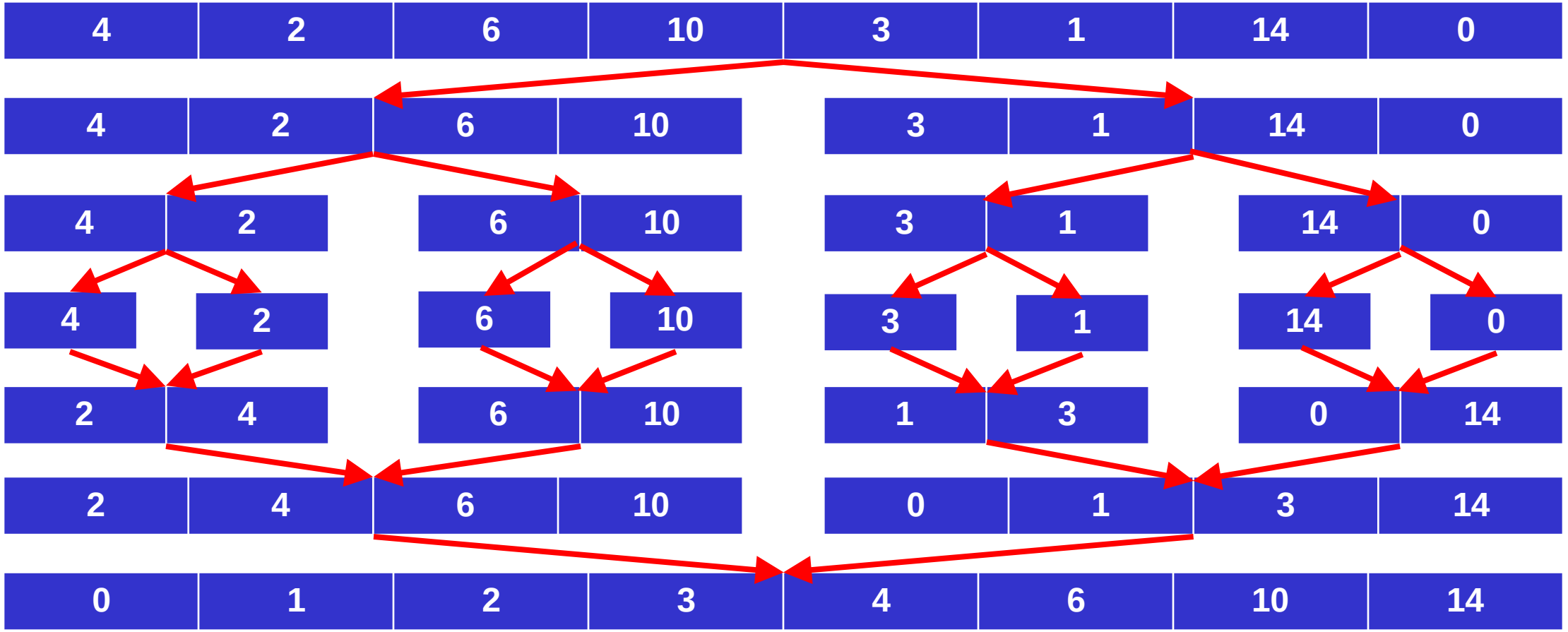
Merge sort: Split recursively



Merge sort: Merge recursively



Merge sort: Merge recursively



Merge sort: O(n *log n)

List of length n

Recursively divide into two halves until length 0 or 1:

log (n) layers

Recursively split:

Java 7+: O(1) at each layer

Java 1-6 : O(n) at each layer

Recursively merge the lists together in each layer:

n comparisons at each “level”

Java 7+: 0.5 * log(n) * n

Java 1-6: n * log(n)