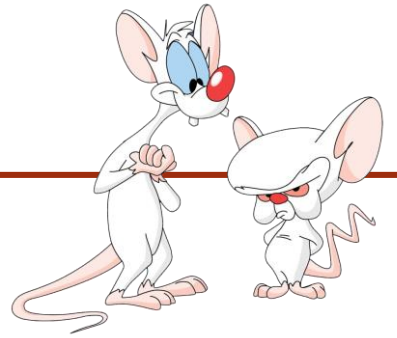
COMP 250
INTRODUCTION TO COMPUTER SCIENCE

Week 10-1: Recursion 3 (Mergesort and Quicksort)

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- Merge sort
- Quick sort

TIME COMPLEXITY

$O(log\, n)$	$O(n)$	$O(n^2)$
▪ convert to binary ▪ binary search ▪ ……	▪ List operations: findMax, remove ▪ grade school addition ▪ …..	▪ insertion/selection/bubble sort ▪ grade school multiplication ▪ ……

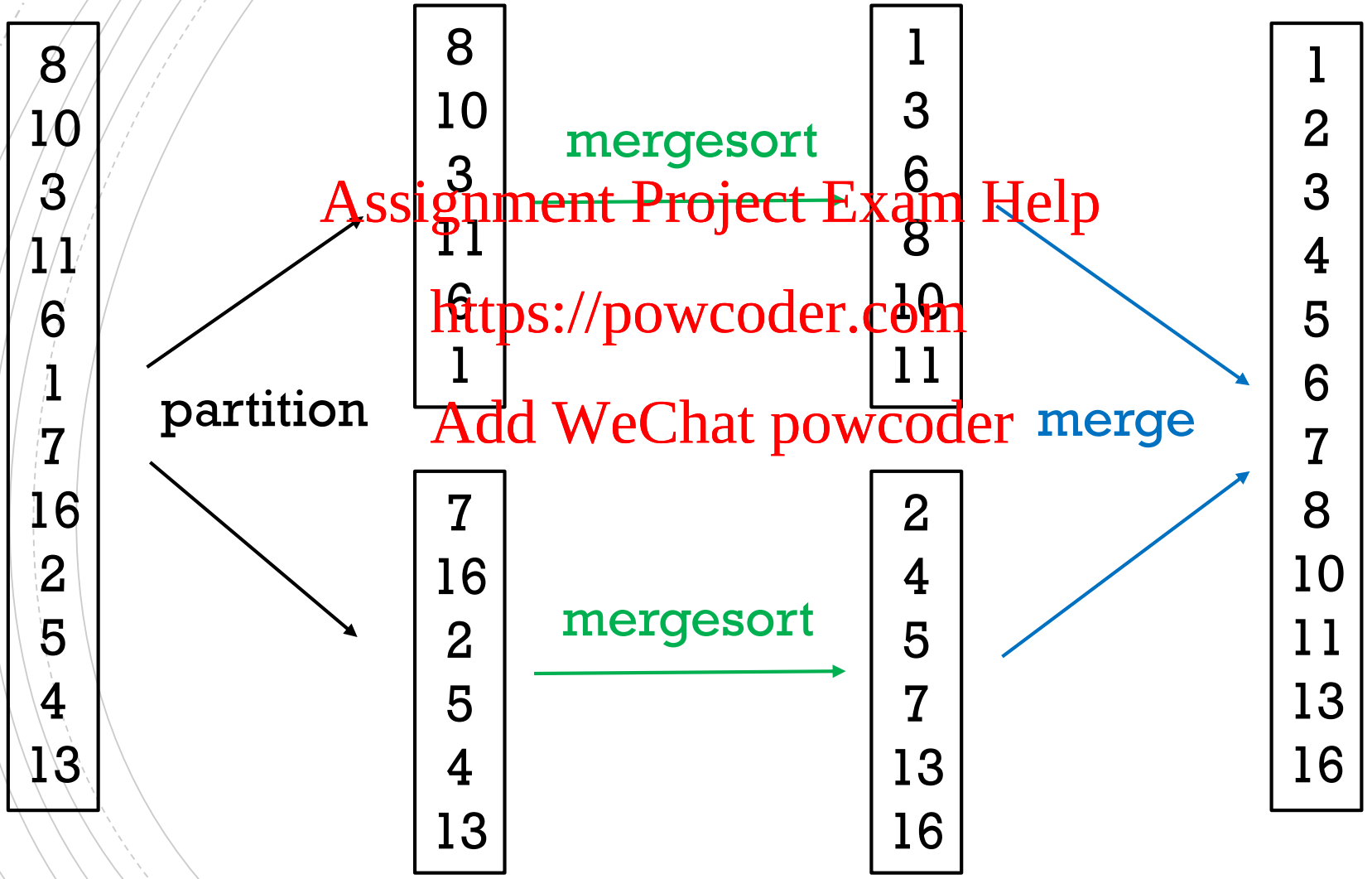
MERGE SORT

MERGE SORT

Merge Sort is a divide and conquer algorithm.

- GOAL: Sort an list.
- IDEA:
  - Partition the list into two halves.
  - Sort each half recursively
  - Merge the sorted half maintaining the order.

IDEA



IMPLEMENTATION

```
mergesort(list){
        if (list.size() == 1)
                return list
        else {
            mid = (list.size()-1) / 2
            list1 = list.getElements(0,mid)
            list2 = list.getElements(mid+1, list.size()-1)
            list1 = mergesort(list1)
            list2 = mergesort(list2)
            return merge(list1, list2)
        }
}
```

IMPLEMENTATION

```
mergesort(list){
        if (list.size() == 1)
              return list
        else {
            mid = (list.size()-1) / 2
            list1 = list.getElements(0,mid)
            list2 = list.getElements(mid+1, list.size()-1)
            list1 = mergesort(list1)
            list2 = mergesort(list2)
            return merge(list1, list2)
        }
}
```

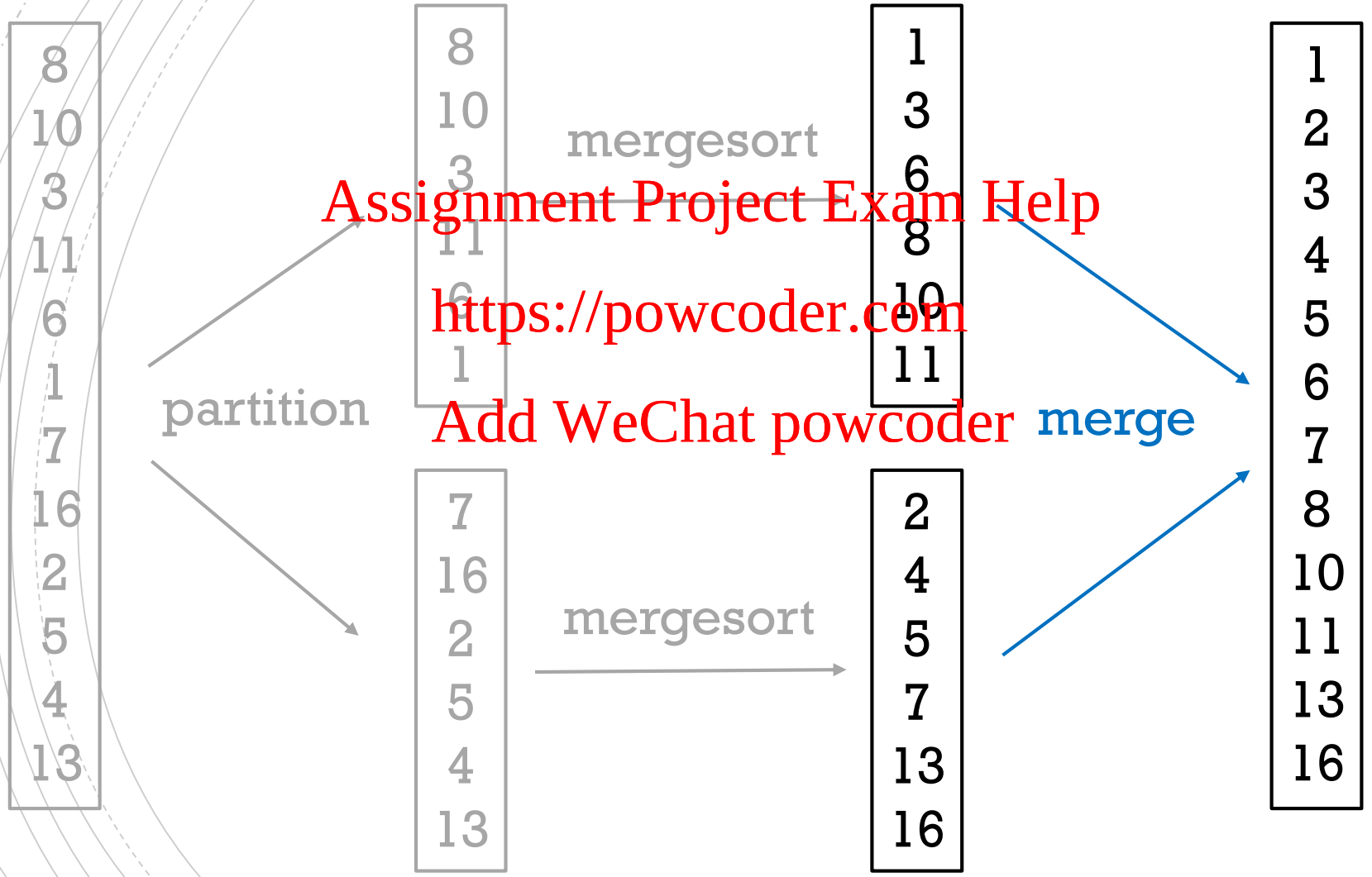
Base case

Partition

Recursive sort

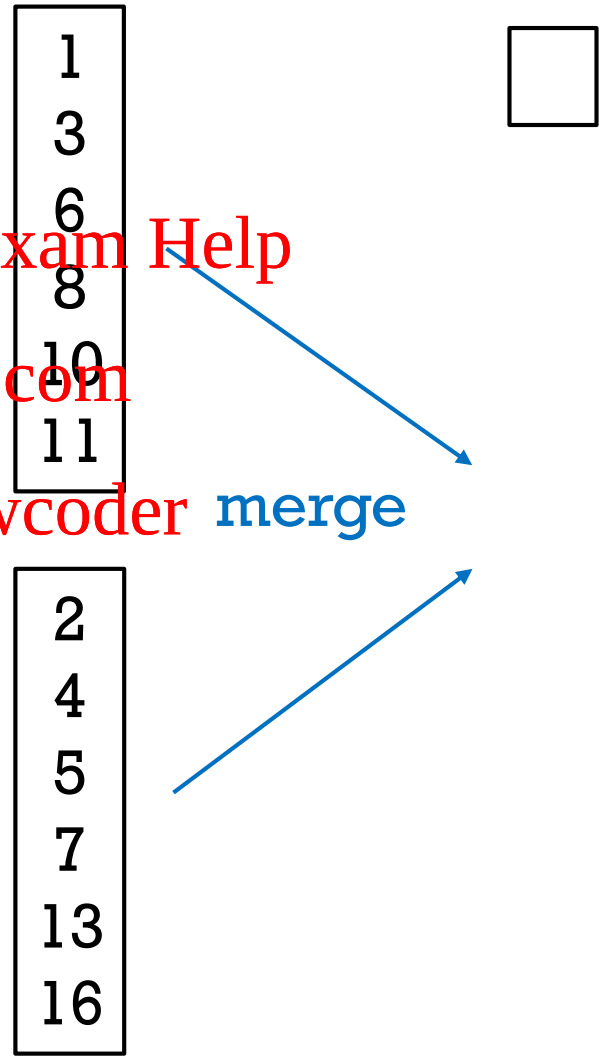
Merge

MERGING PRESERVING ORDER

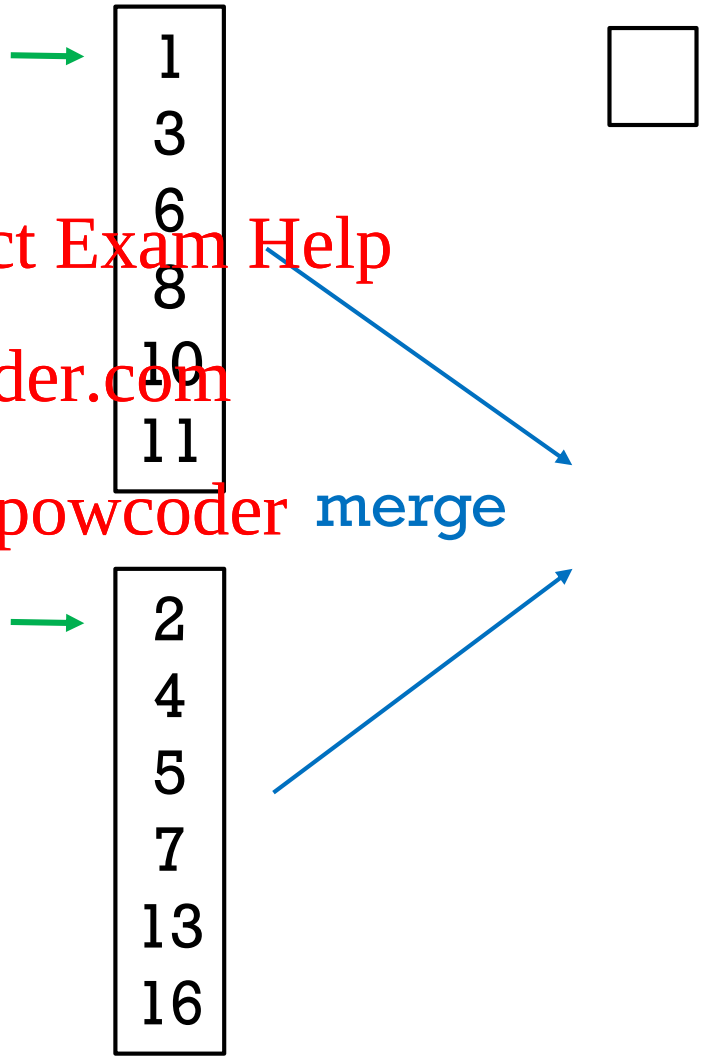


MERGING PRESERVING ORDER

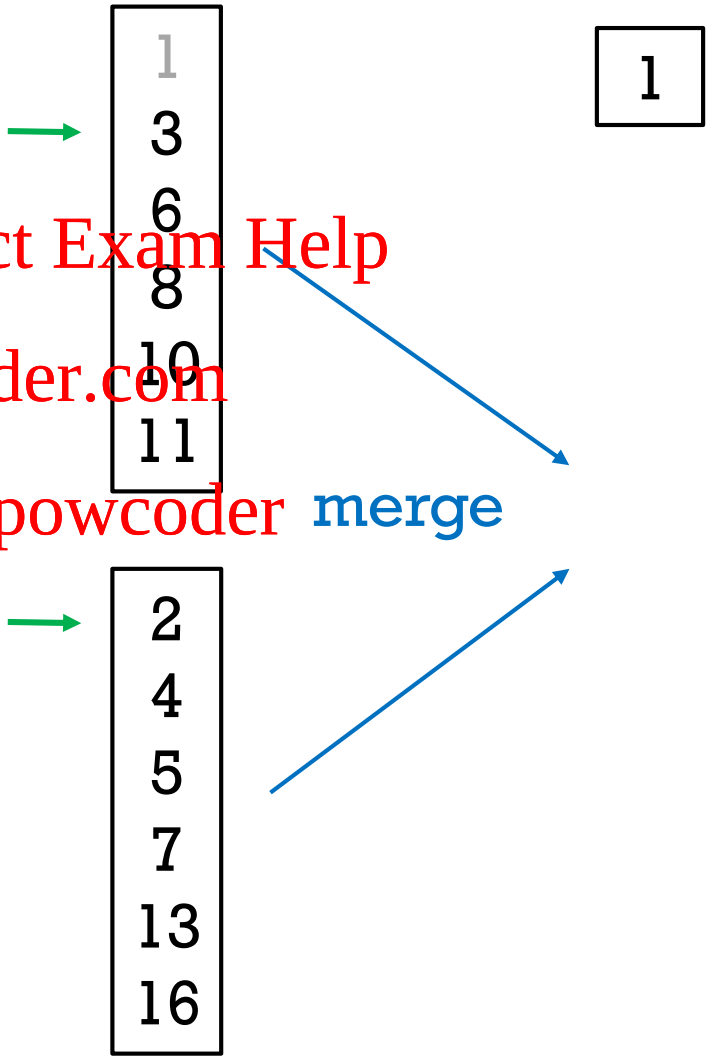
Iterate through the elements of the two sorted list. Depending on how they compare decide which element comes first in the merged list.



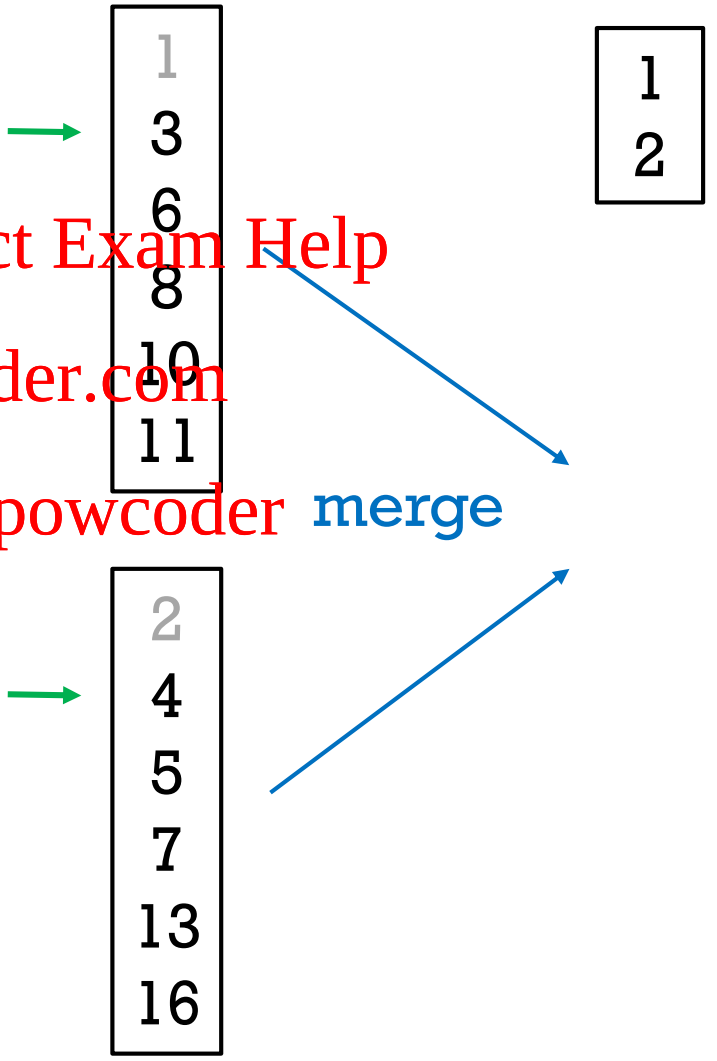
MERGING PRESERVING ORDER



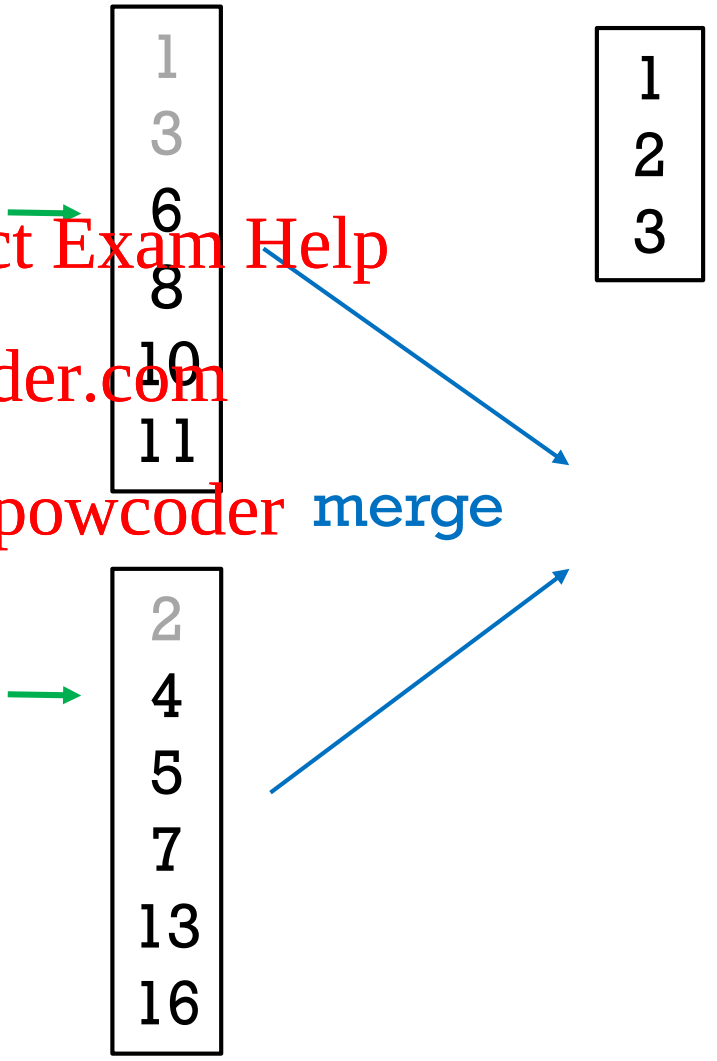
MERGING PRESERVING ORDER



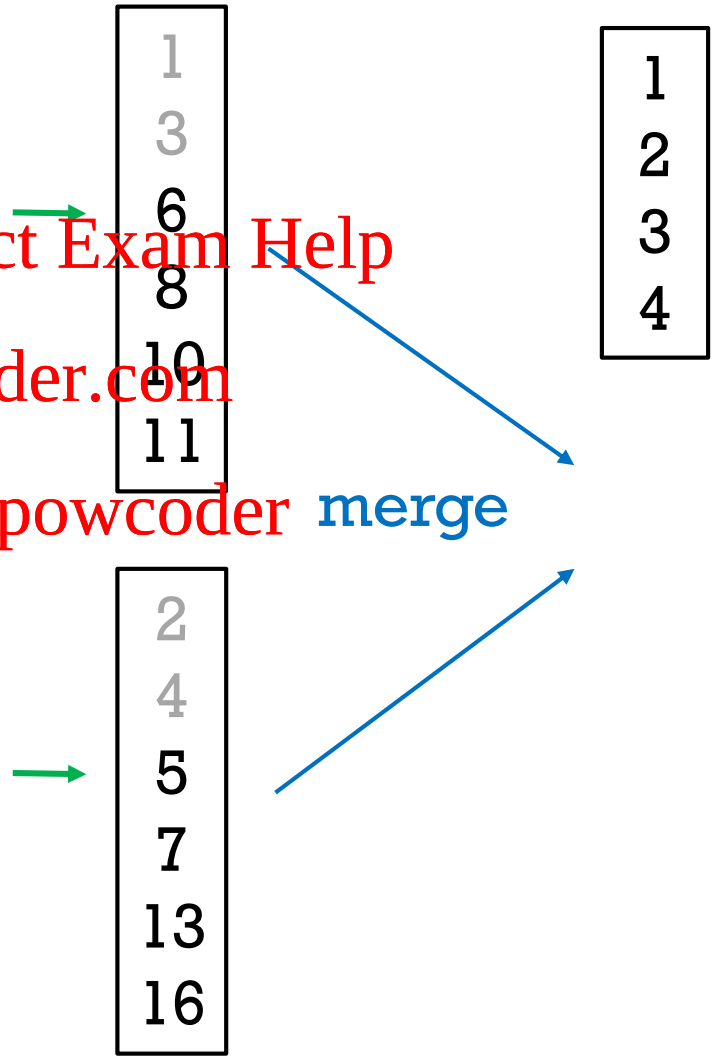
MERGING PRESERVING ORDER



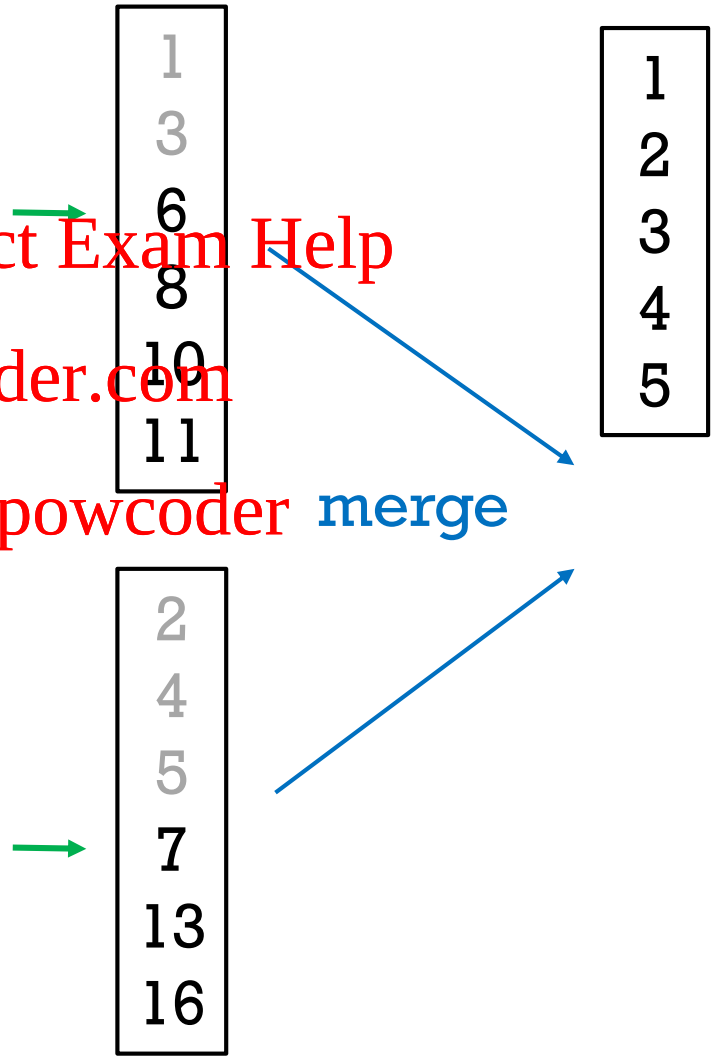
MERGING PRESERVING ORDER



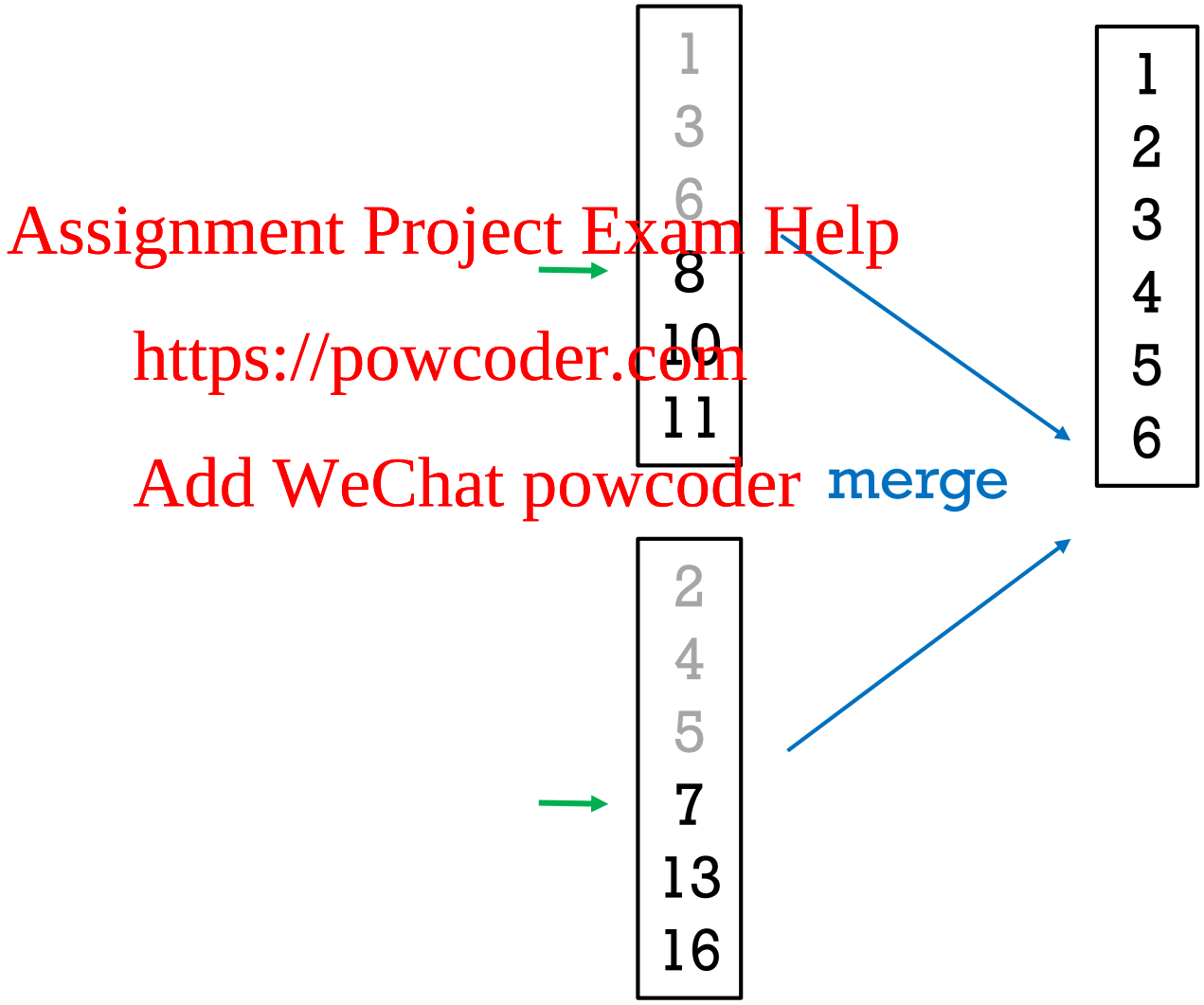
MERGING PRESERVING ORDER



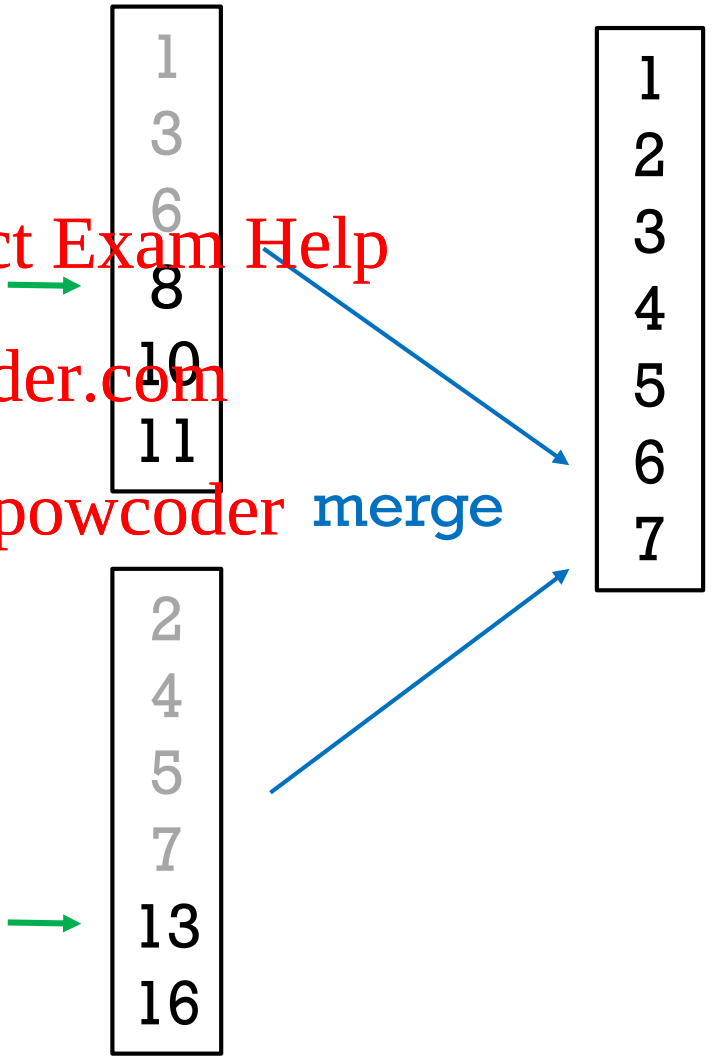
MERGING PRESERVING ORDER



MERGING PRESERVING ORDER

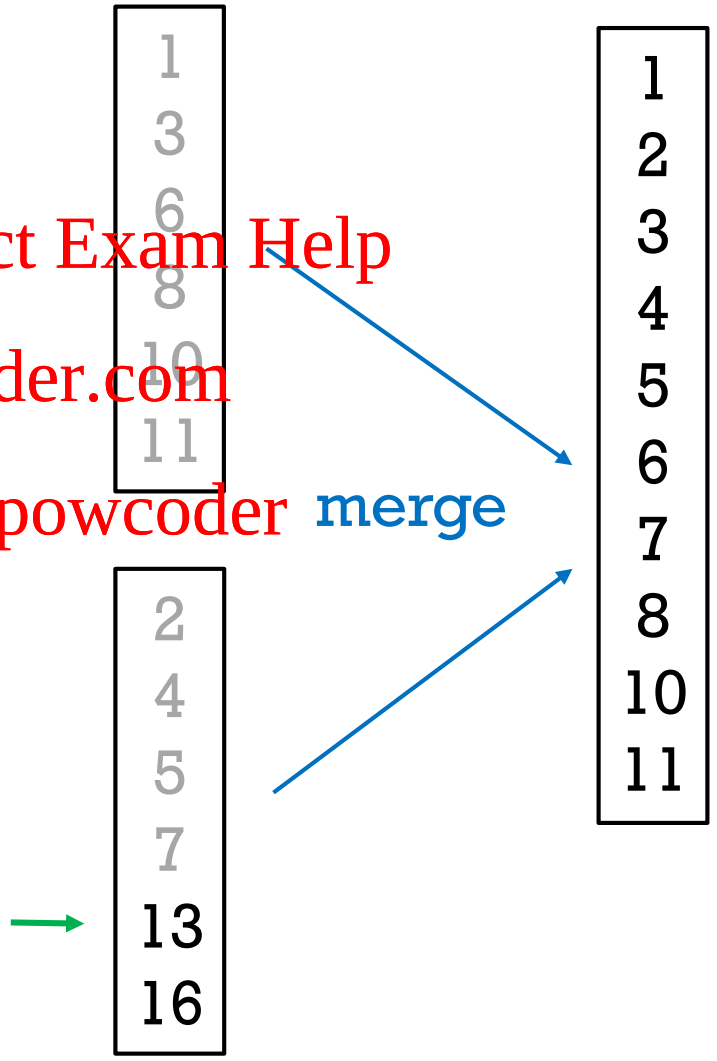


MERGING PRESERVING ORDER



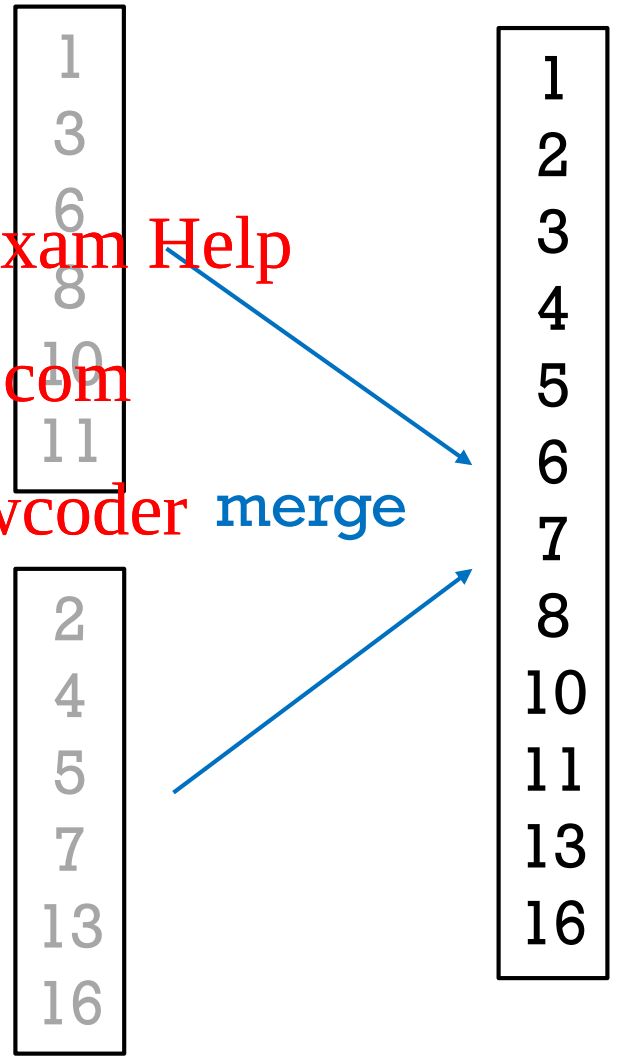
MERGING PRESERVING ORDER

And so on until one list is empty!



MERGING PRESERVING ORDER

Then copy the remaining elements!



IMPLEMENTATION OF MERGE

```
merge(list1, list2){

      list = ...initialize with empty list...

      while (!list1.isEmpty() && !list2.isEmpty()){
            if (list1.get(0) < list2.get(0))
                  list.addlast(list1.removeFirst())
            else
                  list.addlast(list2.removeFirst())
      }

      while (!list1.isEmpty())
            list.addlast(list1.removeFirst())

      while (!list2.isEmpty())
            list.addlast( list2.removeFirst())

      return  list

}
```

IMPLEMENTATION OF MERGE

```
merge(list1, list2){
      list = ...initialize with empty list...
      while (!list1.isEmpty() && !list2.isEmpty()){
            if (list1.get(0) < list2.get(0))
                  list.addlast(list1.removeFirst())
            else
                  list.addlast(list2.removeFirst())
      }
      while (!list1.isEmpty())
            list.addlast(list1.removeFirst())
      while (!list2.isEmpty())
            list.addlast( list2.removeFirst())
      return  list
}
```


Pick elements to add until one of the two lists is empty

Then add the remaining elements

1) Partitions into list and call mergesort until base case is reached.

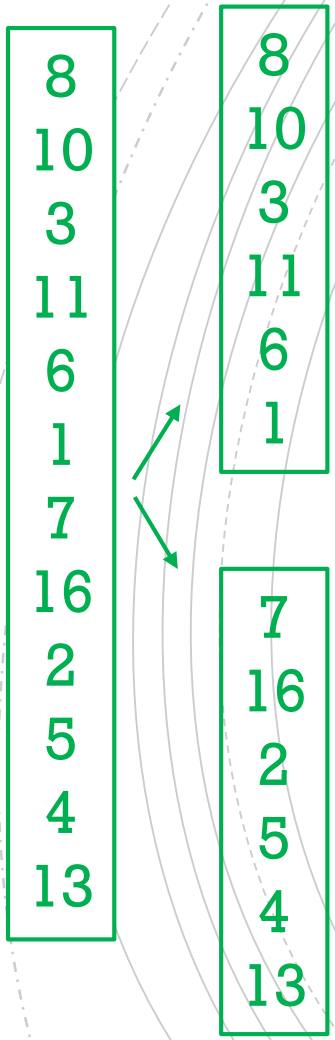
EXAMPLE OF EXECUTION

8
10
3
11
6
1
7
16
2
5
4
13

```
mergesort(list){
        if (list.size() == 1)
            return list
        else {
            mid = (list.size() - 1) / 2
            list1 = list.getElements(0,mid)
            list2 = list.getElements(mid+1, list.size()-1)
            list1 = mergesort(list1)
            list2 = mergesort(list2)
            return merge(list1, list2)
        }
}
```

1) Partitions into list and call mergesort until base case is reached.

EXAMPLE OF EXECUTION

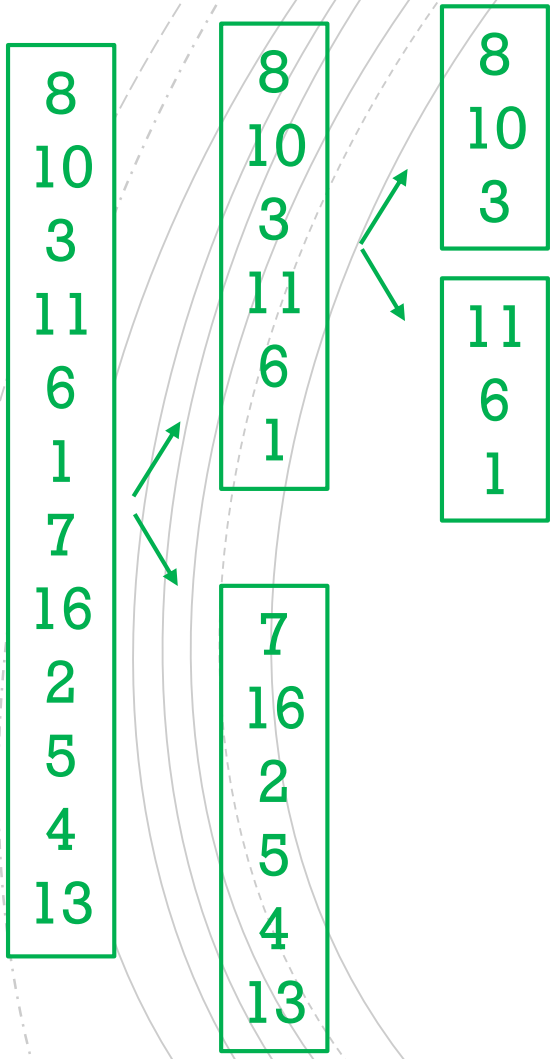


list1

list2

1) Partitions into list and call mergesort until base case is reached.

EXAMPLE OF EXECUTION

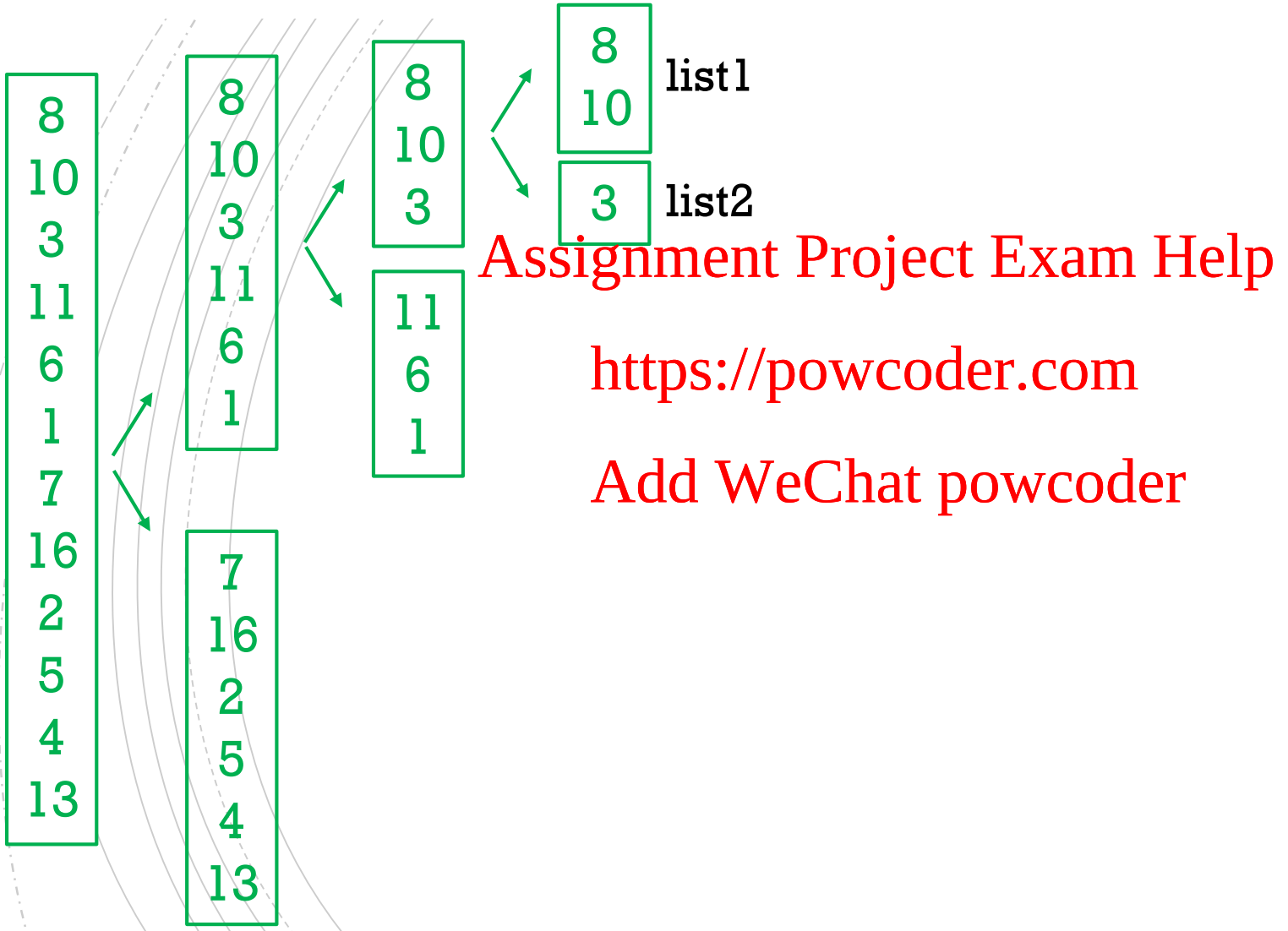


list1

list2

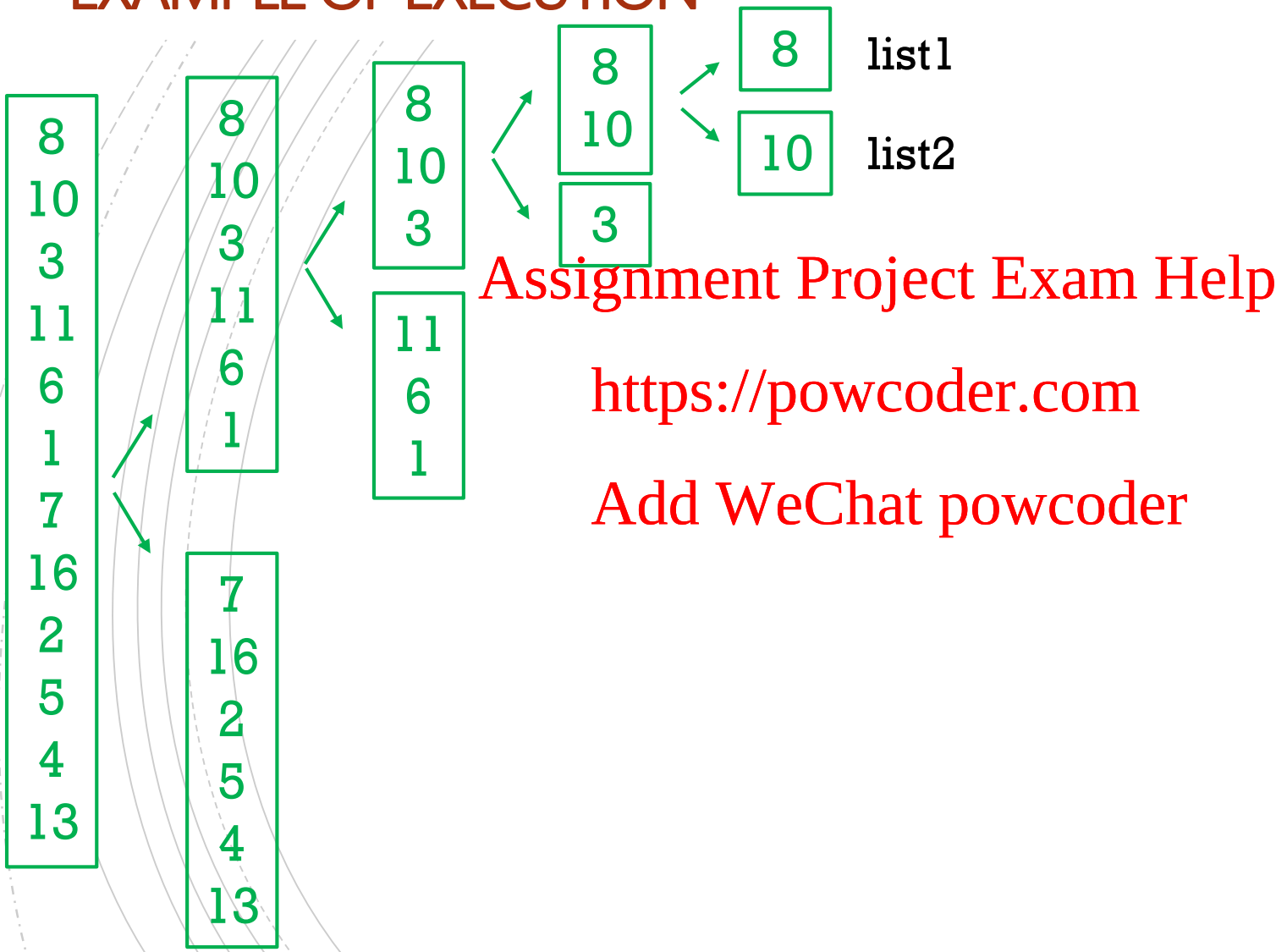
1) Partitions into list and call mergesort until base case is reached.

EXAMPLE OF EXECUTION



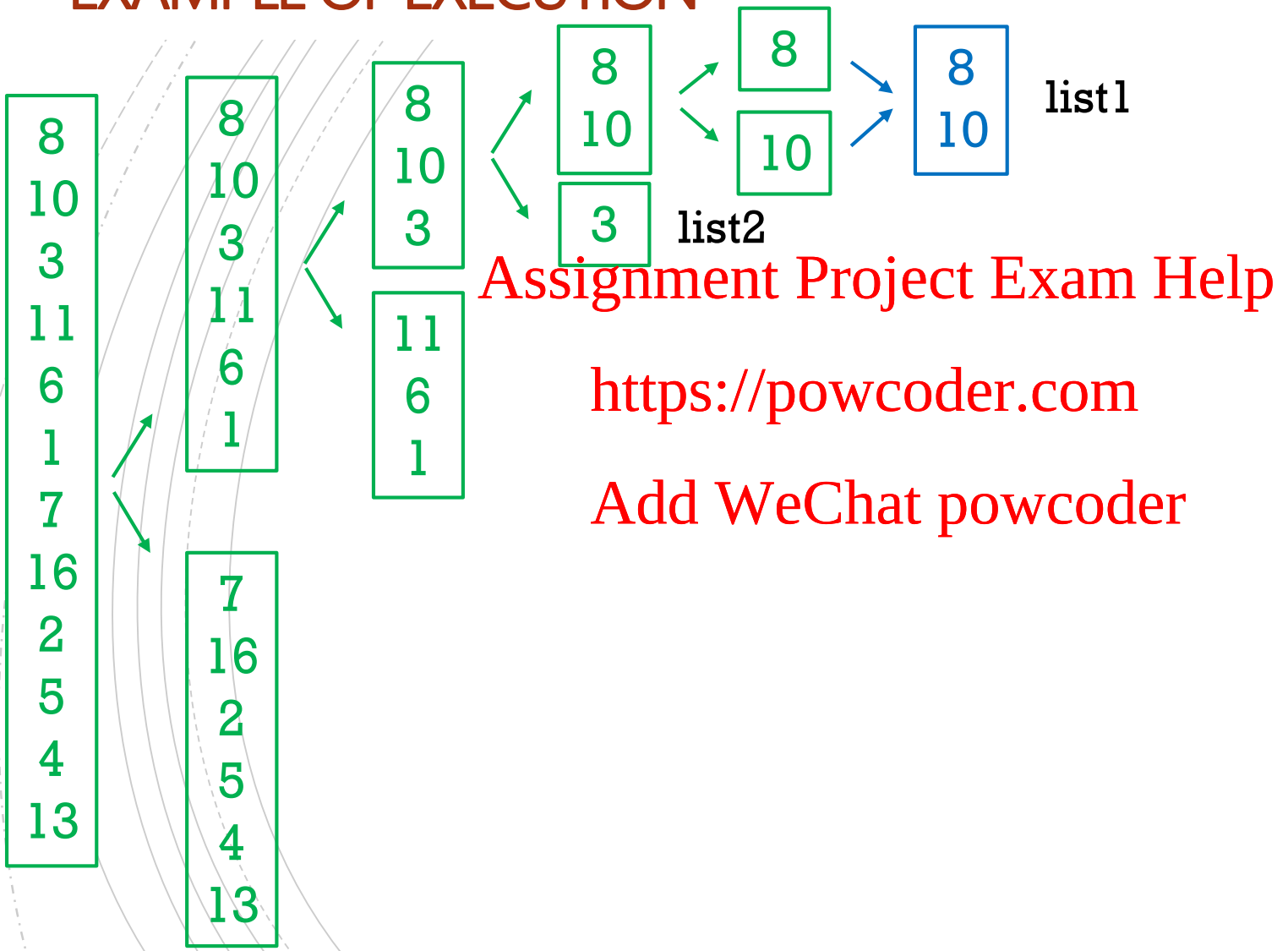
1) Partitions into list and call mergesort until base case is reached.

EXAMPLE OF EXECUTION



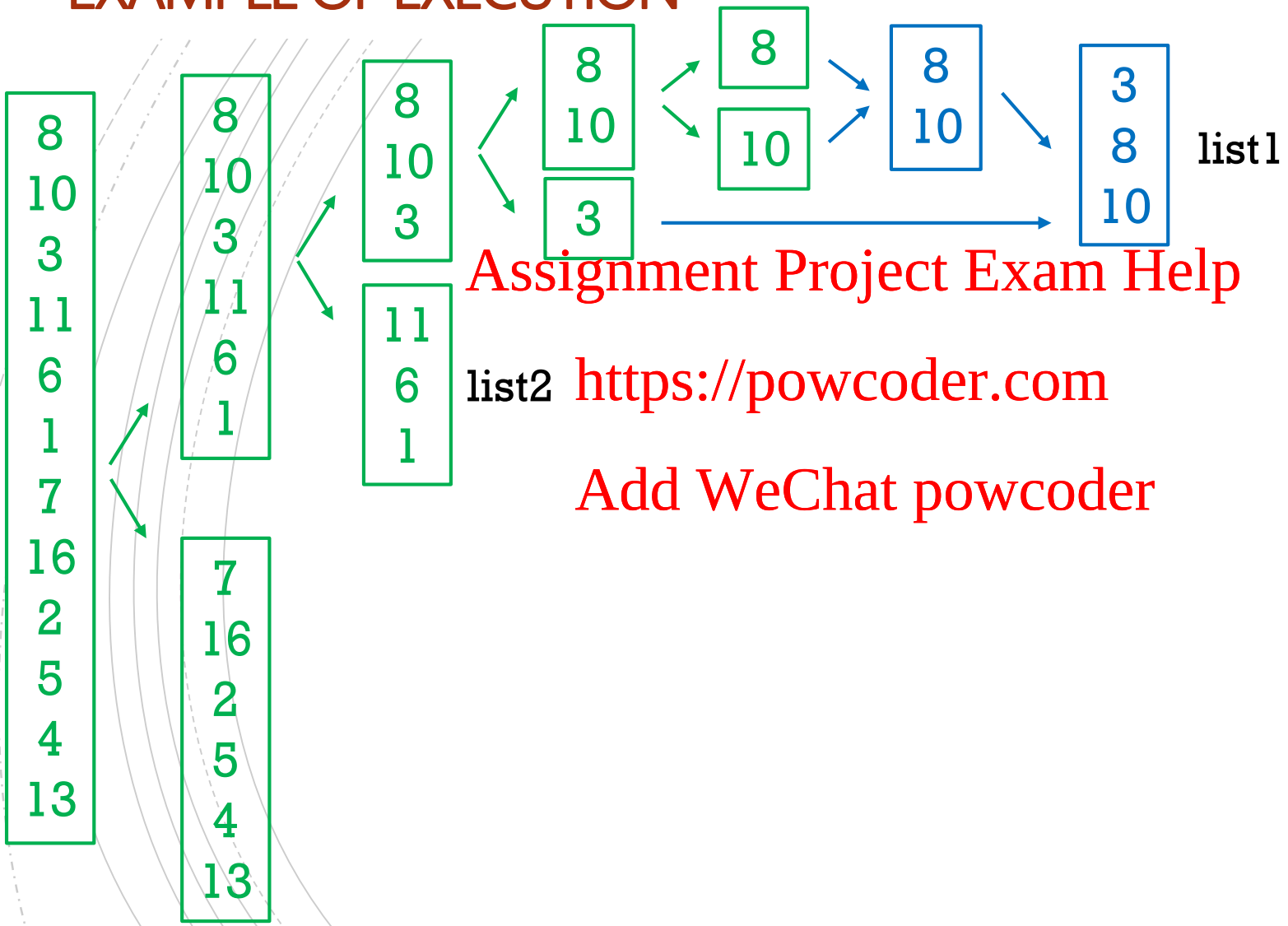
Then start to merge!

EXAMPLE OF EXECUTION



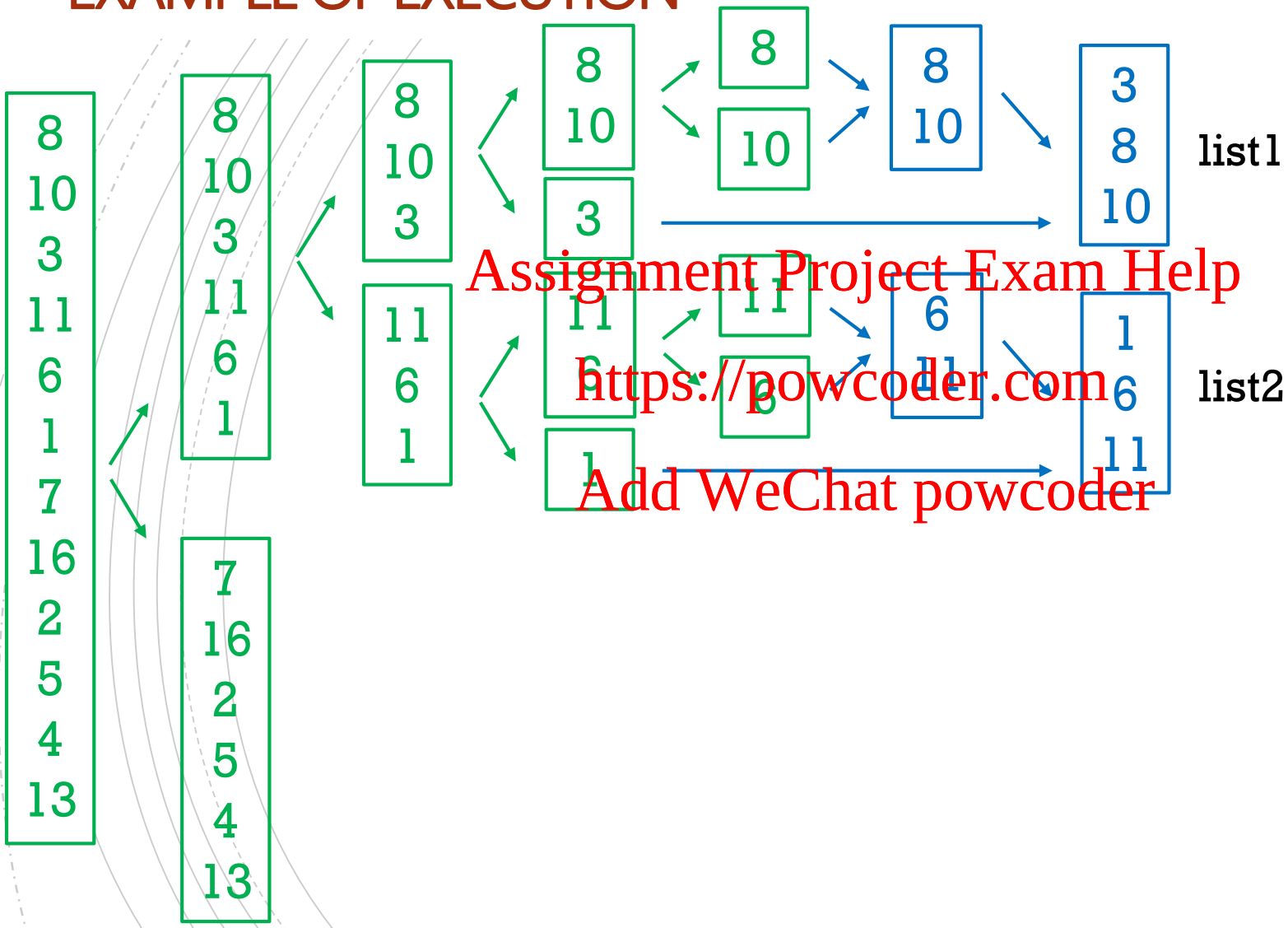
Then start to merge!

EXAMPLE OF EXECUTION



Then start to merge!

EXAMPLE OF EXECUTION

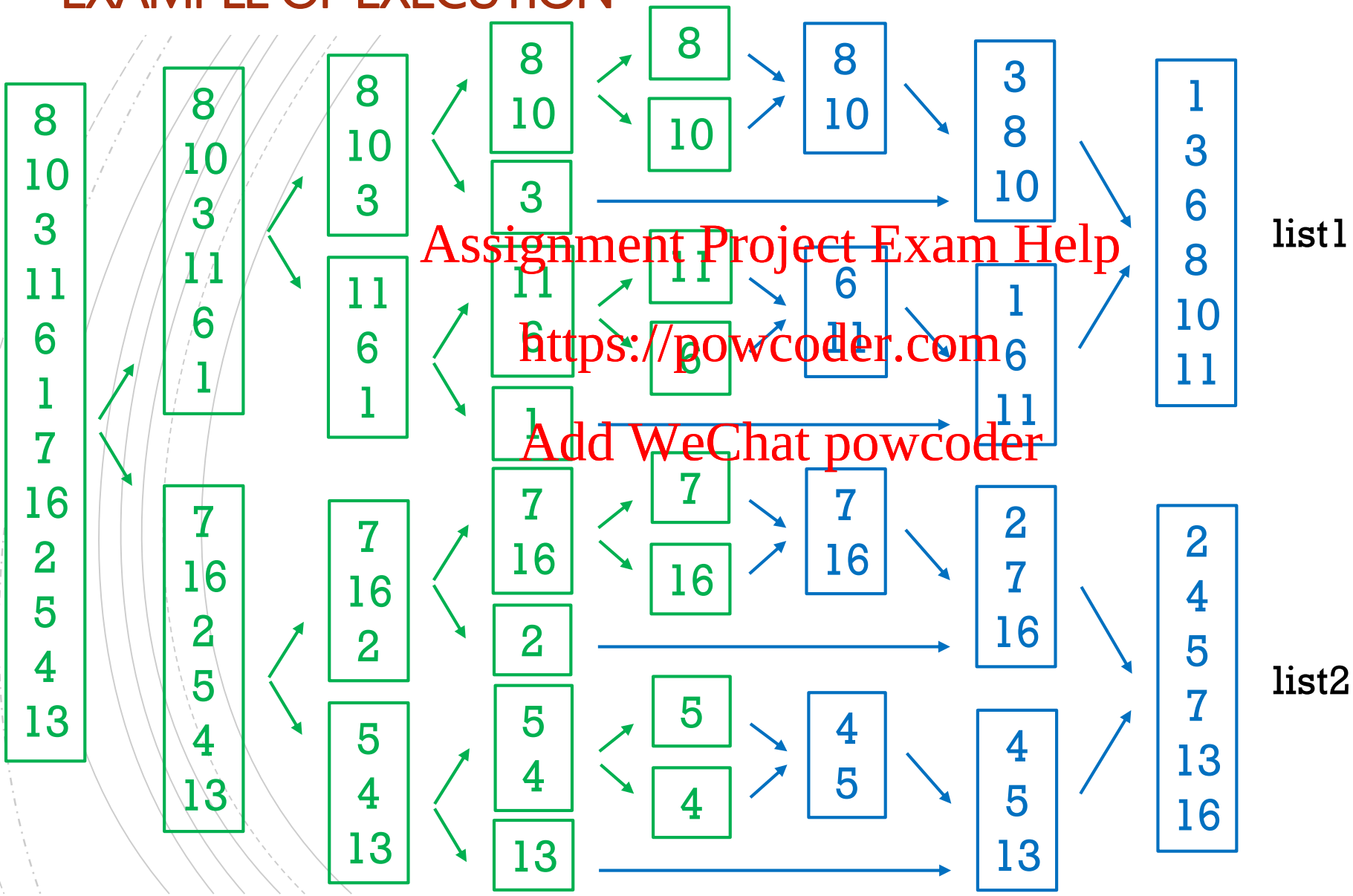


Then start to merge!

EXAMPLE OF EXECUTION

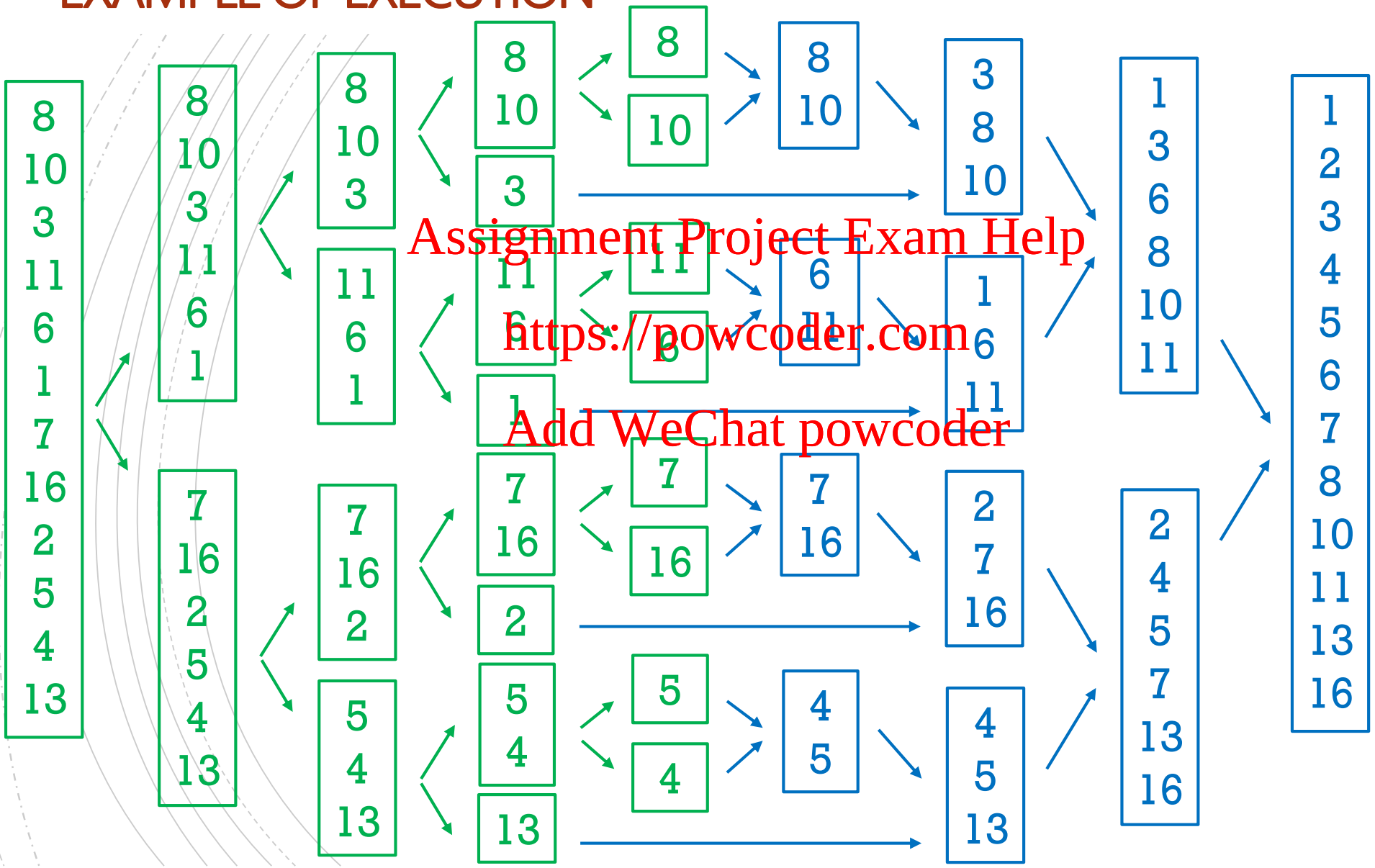
Then start to merge!

EXAMPLE OF EXECUTION



Then start to merge!

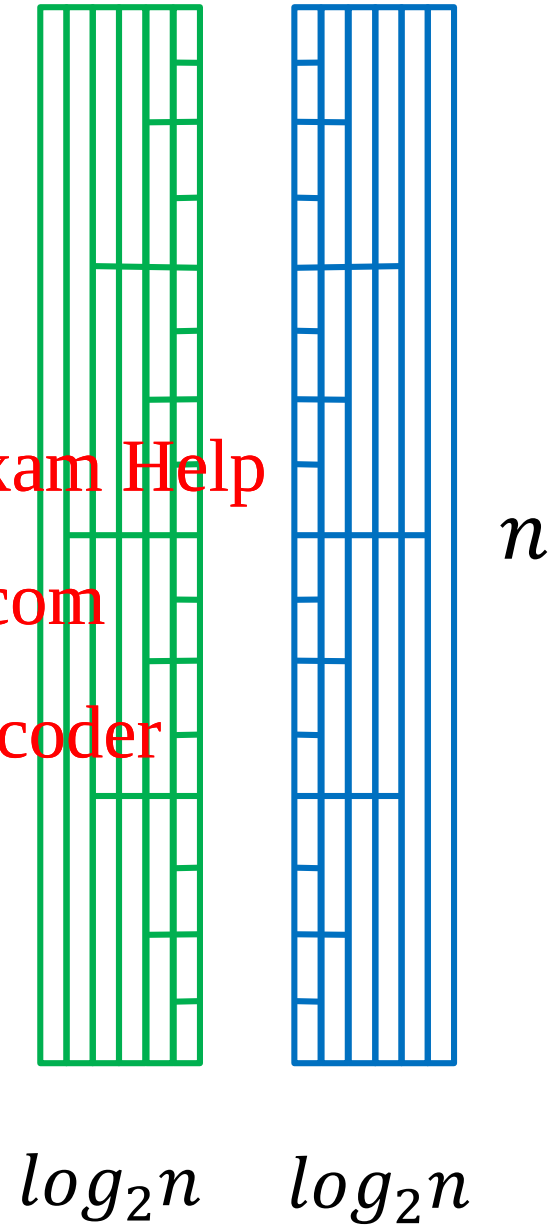
EXAMPLE OF EXECUTION



Q: How many operations are required to mergesort a list of size n ?

A: $O(\ n\ \ log_2 n\ )$

This will become more clear in the next video when we discuss recurrences.



COMPLEXITY

$n\ log_2\ n$ is much closer to n than to n^2

$log_2\ n$	n	$n\ log_2\ n$	n^2
10	$2^{10} \approx 10^3$	10^4	10^6
20	$2^{20} \approx 10^6$	$\sim 10^7$	10^{12}
30	$2^{30} \approx 10^9$	$\sim 10^{10}$	10^{18}

COMPLEXITY

$n\ log_2\ n$ is much closer to n than to n^2

$log_2\ n$	n	$n\ log_2\ n$	n^2
10	$2^{10} \approx 10^3$	10^4	10^6
20	$2^{20} \approx 10^6$	$\sim 10^7$	10^{12}
30	$2^{30} \approx 10^9$	$\sim 10^{10}$	10^{18}

milliseconds seconds minutes/hours centuries

QUICK SORT

QUICK SORT

- Quick Sort is a divide and conquer algorithm.
- GOAL: Sort a list.
- IDEA:
  - Pick an element of the array (the pivot).
  - Partition the list moving the pivot to its correct position making sure that all the lower elements are on its left and all the larger elements are on its right.
  - Sort the left part AND the right part of the list recursively.
  - Keep doing it until there's nothing left to sort.

IDEA

IDEA:

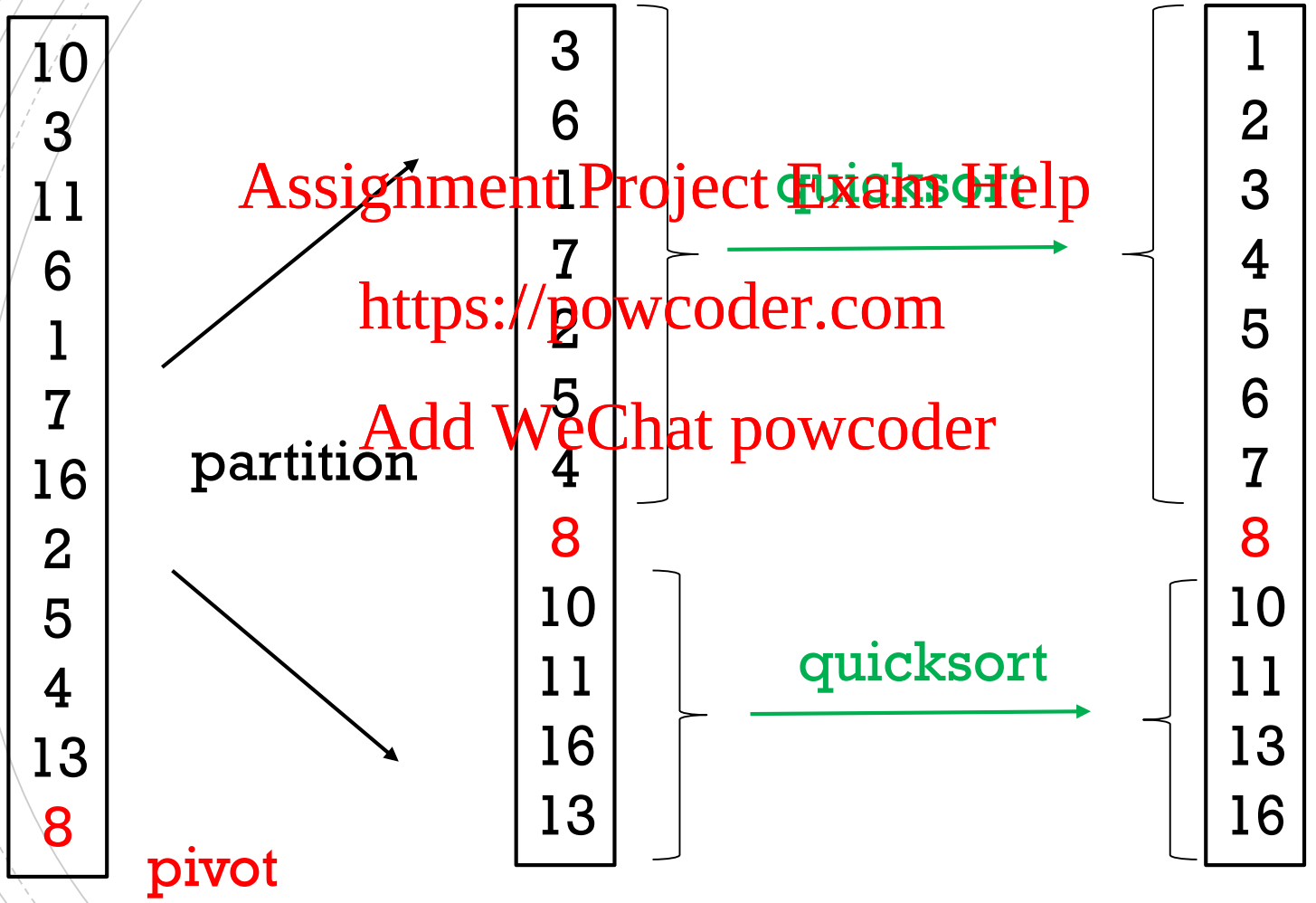
- Pick an element of the array (the pivot).
- Move the pivot to its correct position making sure that all the smaller elements are on its left and all the larger elements are on its right.
- Sort the left part AND the right part
- Keep doing it until there's nothing left to sort.

This is the crucial process of the algorithm!

Recursive Step

Base case

IDEA



THE PIVOT

Different versions of Quick Sort pick the pivot in different ways:

- Always pick the first element as the pivot
- Always pick the last element as the pivot

- Pick a random element

- Pick the median as pivot

THE PIVOT

Different versions of Quick Sort pick the pivot in different ways:

- Always pick the first element as the pivot
- Always pick the last element as the pivot

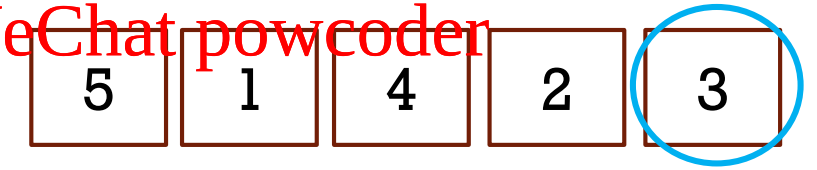
- Pick a random element

- Pick the median as pivot

QUICK SORT EXAMPLE

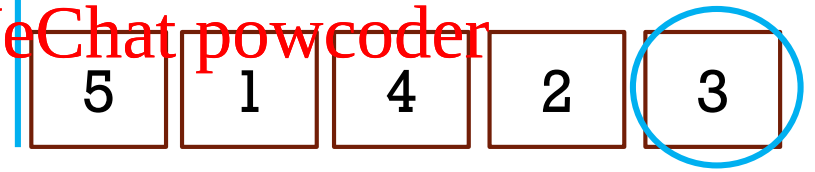
QUICK SORT EXAMPLE

1. Pick the pivot.



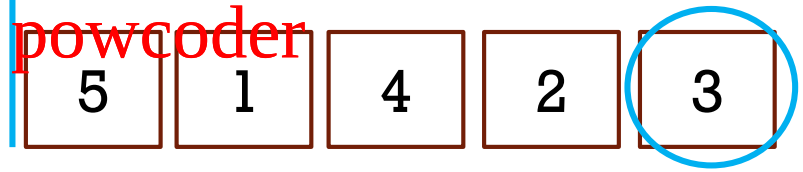
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left



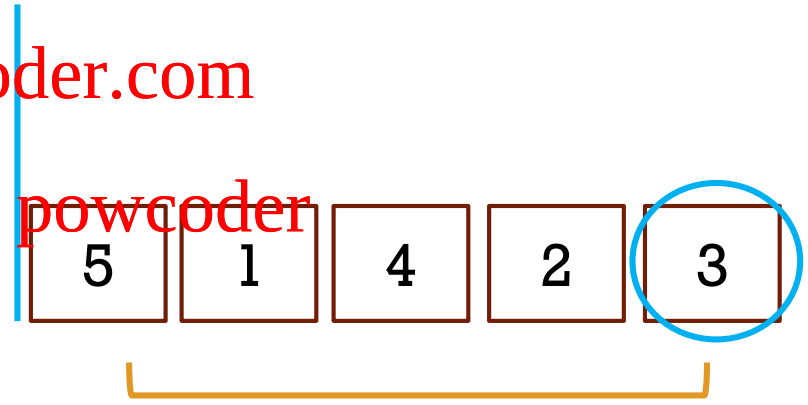
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



QUICK SORT EXAMPLE

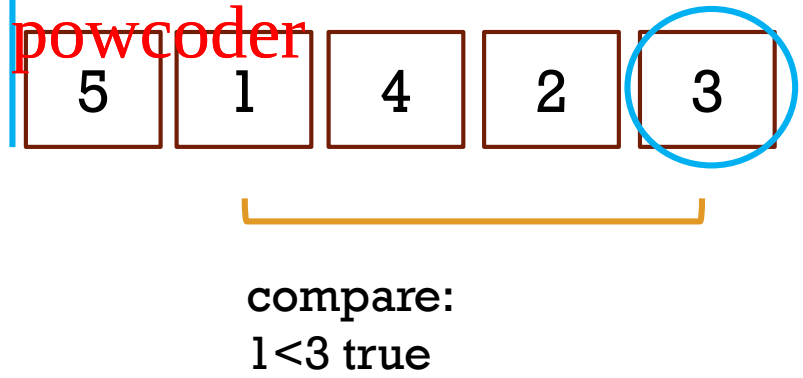
1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



compare:
5<3 false → do nothing!

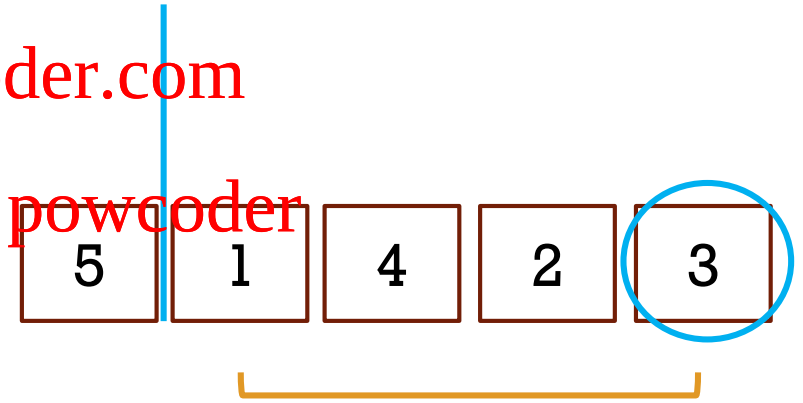
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



QUICK SORT EXAMPLE

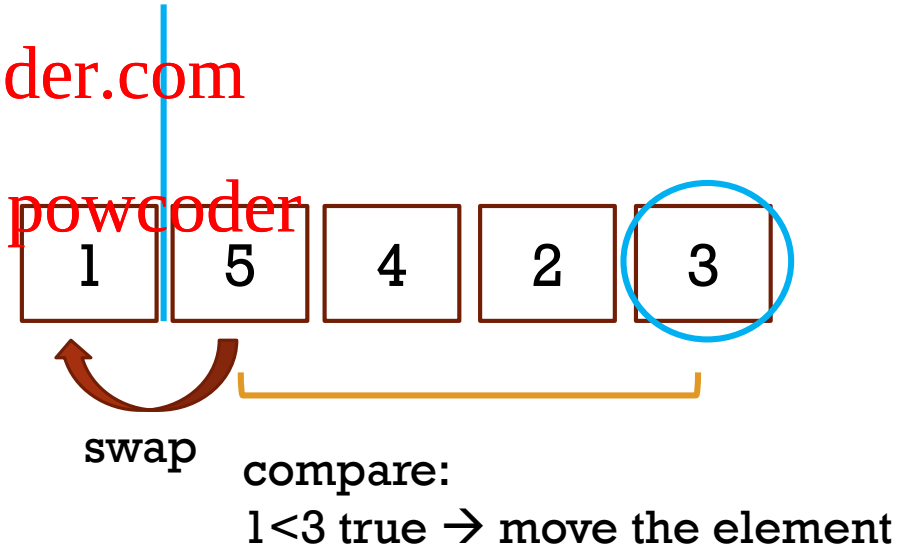
1. Pick the pivot.
2. Set the wall on the left.
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



compare:
1<3 true → move the wall

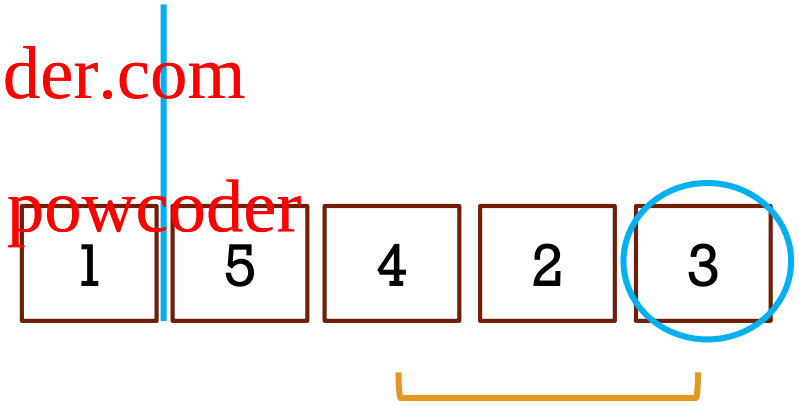
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



QUICK SORT EXAMPLE

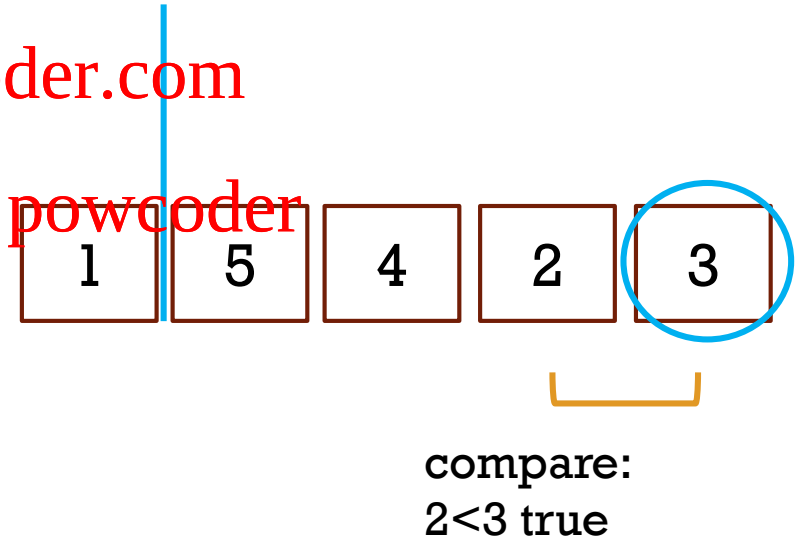
1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



compare:
4<3 false → do nothing

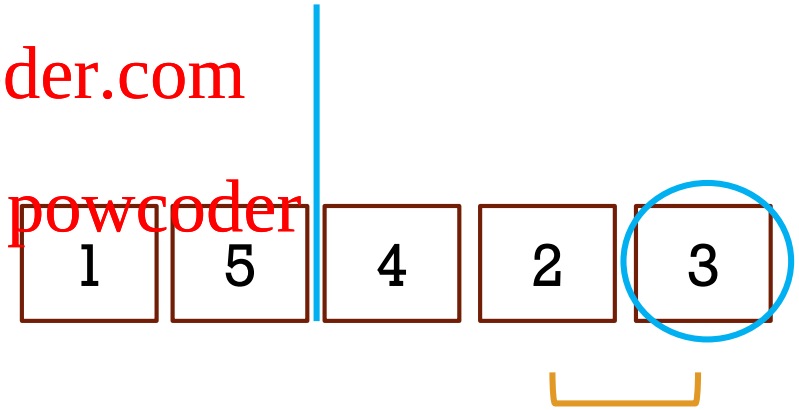
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



QUICK SORT EXAMPLE

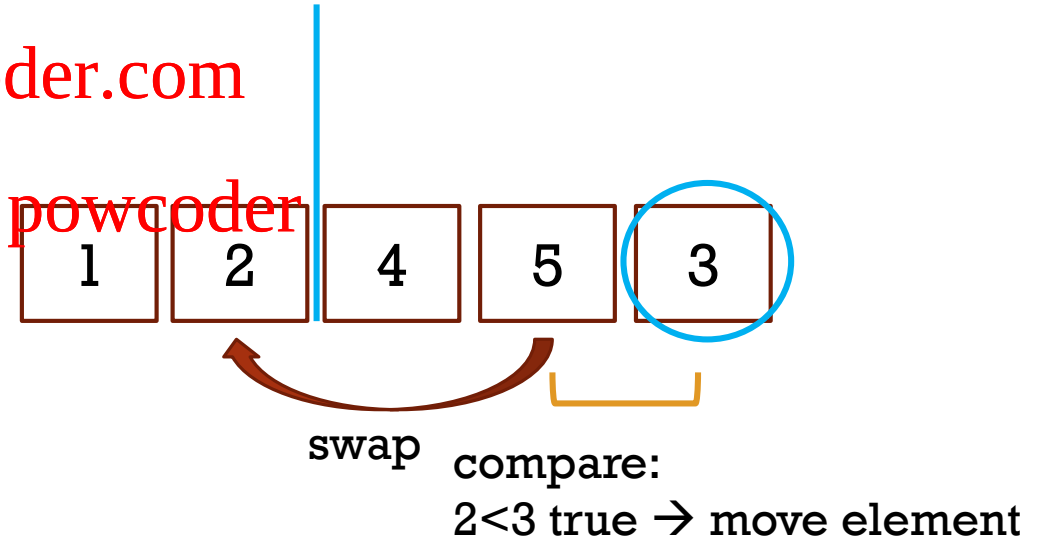
1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



compare:
2<3 true → move wall

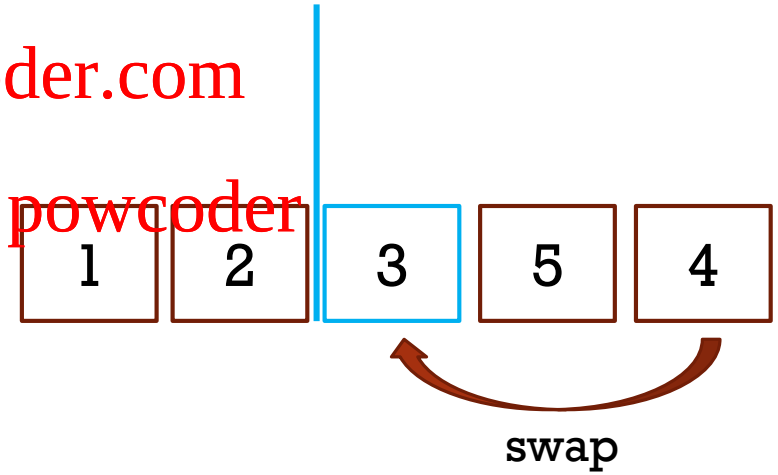
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



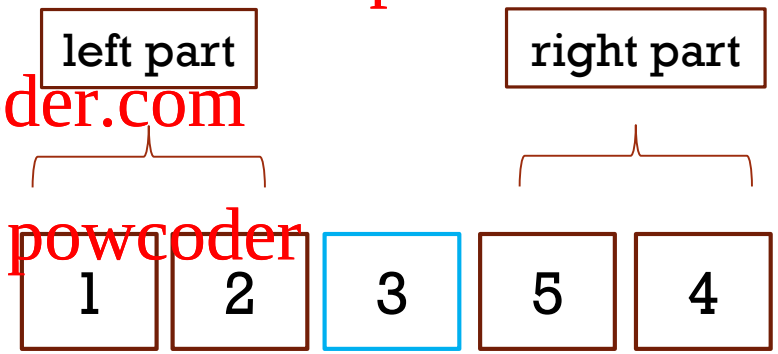
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left.
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.
4. Move the pivot next to the wall.



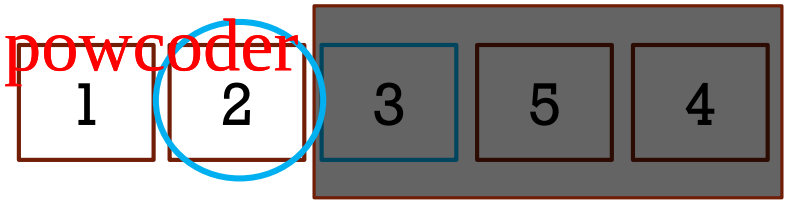
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.
4. Move the pivot next to the wall.
5. Use Quick sort on left part and then on the right part



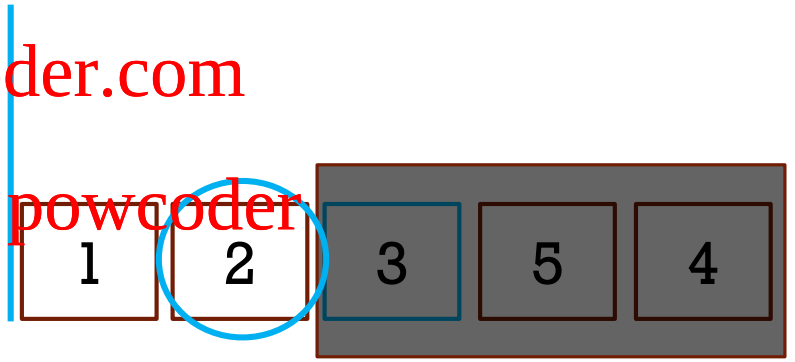
QUICK SORT EXAMPLE

1. Pick the pivot.



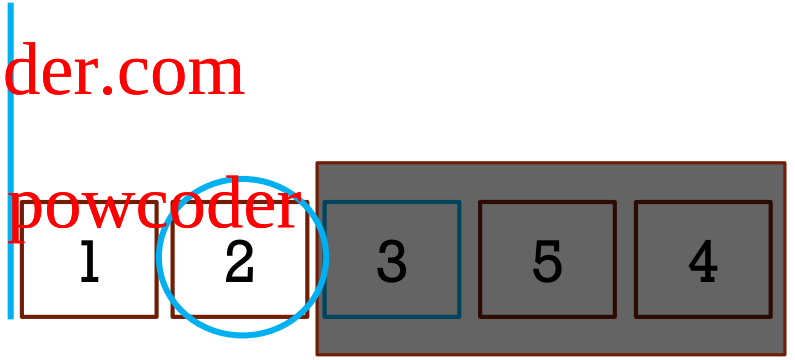
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left



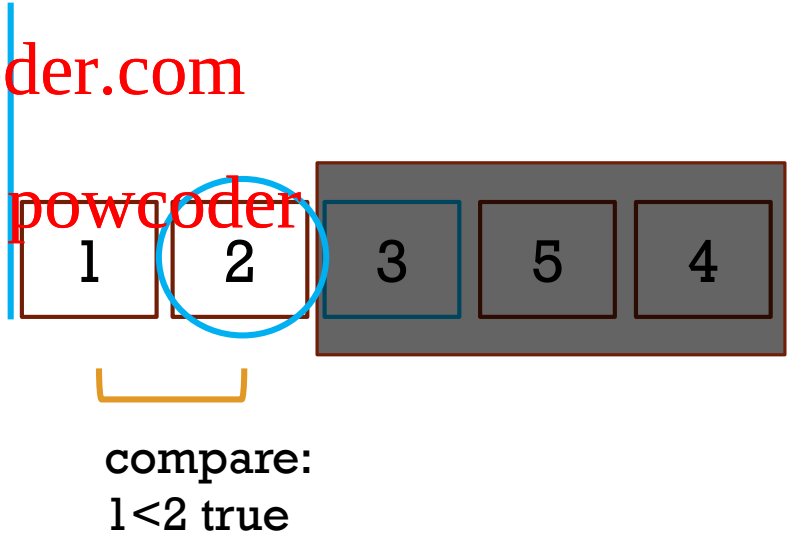
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



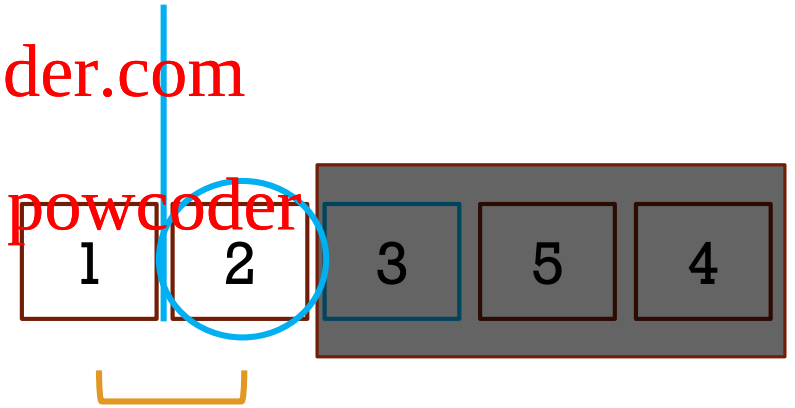
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



QUICK SORT EXAMPLE

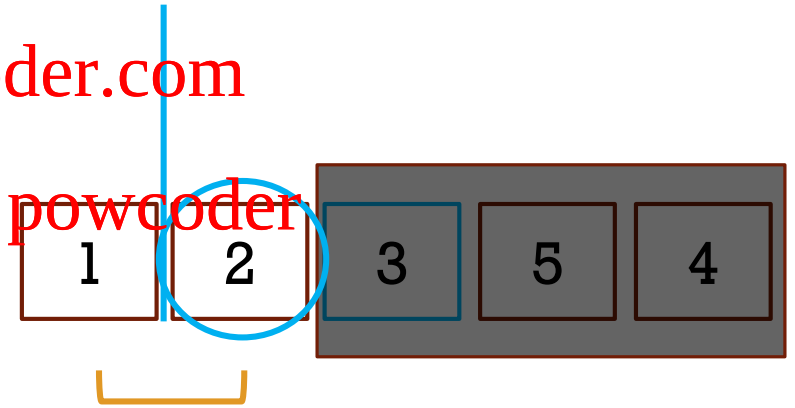
1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



compare:
1<2 true → move wall

QUICK SORT EXAMPLE

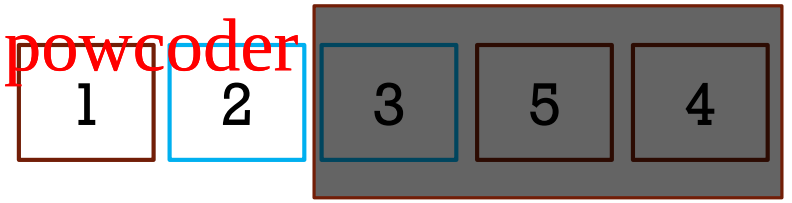
1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



compare:
1<2 true → element already in position.

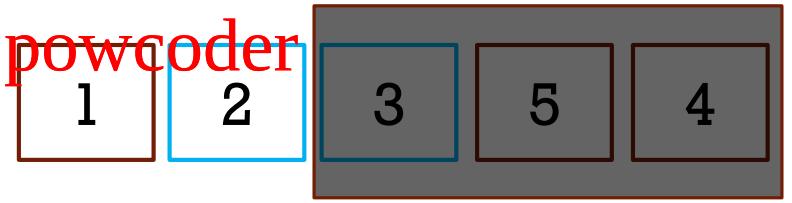
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.
4. Move the pivot next to the wall.
5. Use Quick Sort on left part and then on the right part.



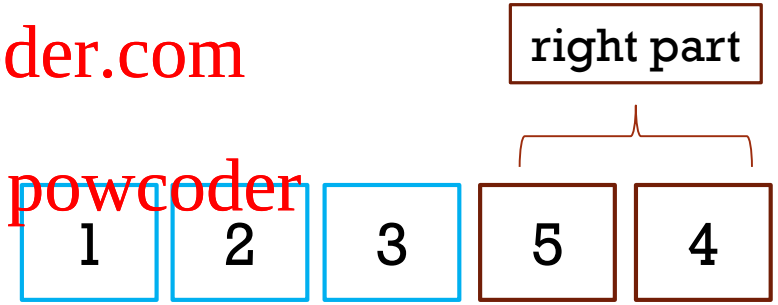
QUICK SORT EXAMPLE

- In this case the left part and the right part are base cases.
- The left part has 1 element → already sorted!
- The right part is empty → sorted!



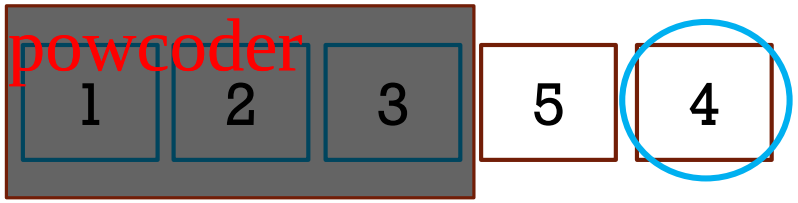
QUICK SORT EXAMPLE

- It is left to sort the part of the list to the right of the first pivot.



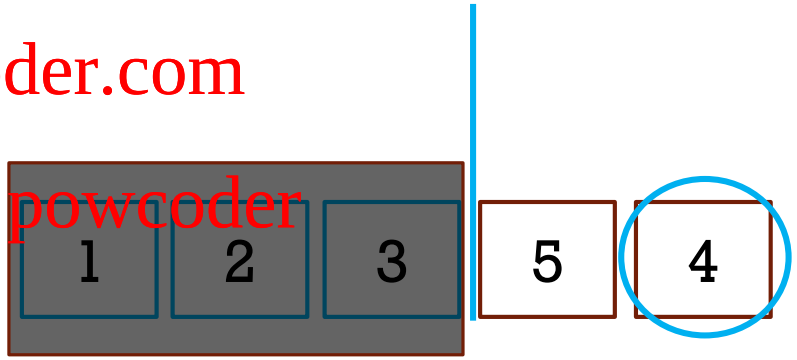
QUICK SORT EXAMPLE

1. Pick the pivot.



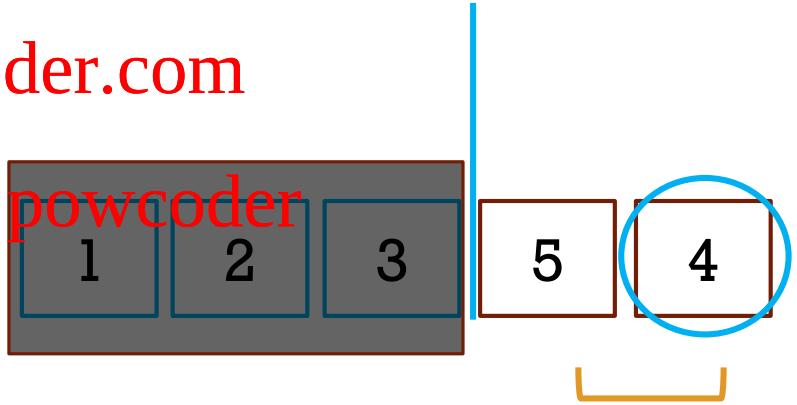
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left



QUICK SORT EXAMPLE

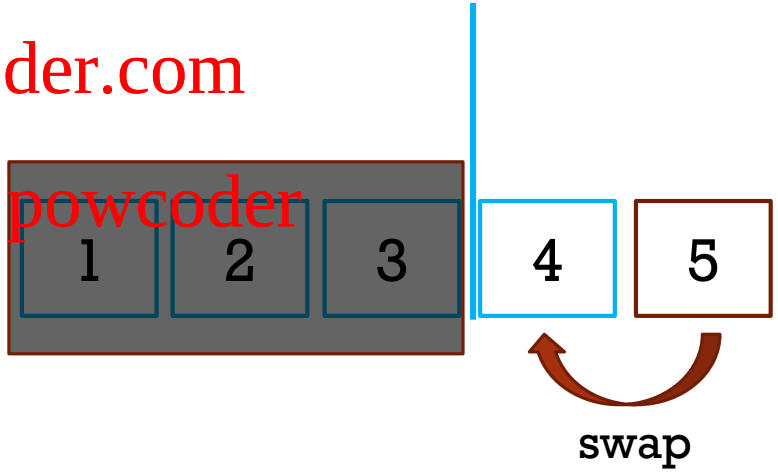
1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.



compare:
5<4 false → do nothing!

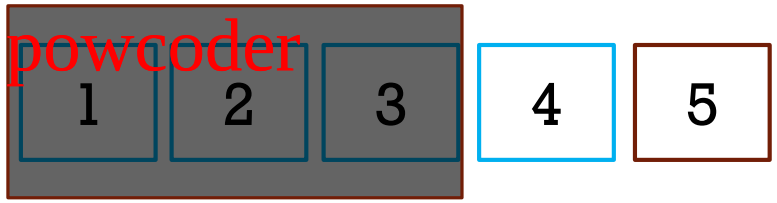
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.
4. Move the pivot next to the wall.



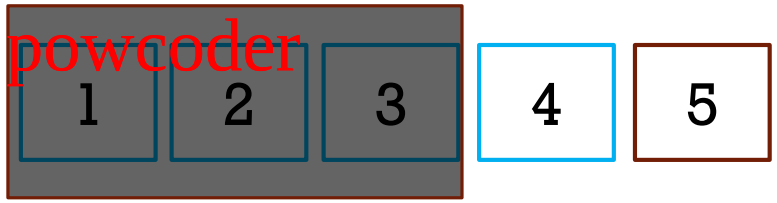
QUICK SORT EXAMPLE

1. Pick the pivot.
2. Set the wall on the left
3. Go through all the elements of the list that are not the pivot.
   - If the element is smaller than the pivot, move the wall right by 1, and place the element just behind the wall.
   - Otherwise, do nothing.
4. Move the pivot next to the wall.
5. Use Quick Sort on left part and then on the right part.



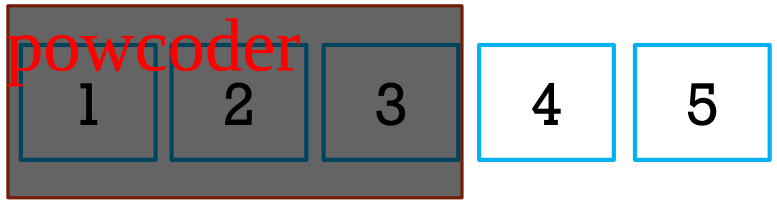
QUICK SORT EXAMPLE

- Once again, both the left part and the right part are base cases.



QUICK SORT EXAMPLE

- Once again, both the left part and the right part are base cases.
- The array is sorted



QUICK SORT EXAMPLE

- Once again, both the left part and the right part are base cases.
- The array is sorted
- The original array is sorted!

QUICK SORT – IMPLEMENTATION

What do we need to implement this algorithm?

- A method that swaps two elements
- A way to refer to parts of the list
- A method that places the pivot in its correct position and moves the elements around so that all the lower elements are on the left, and all the larger elements are on the right. Call it `placeAndDivide`
- A method that implements the Quick Sort, that is:
  - Pick a pivot
  - placeAndDivide
  - quickSort left part
  - quickSort right part

PARTS OF THE LIST

What can we use to denote a part of the list?

- We can use the same idea used from binary search -> keep track of the left and right index denoting where the part begins and ends.
- Consider for example the list {5,3,6,1,2}. Then:
  - The indices 0 and 4 denote the entire list.
  - The indices 0 and 2 denote the part of the list with the 3 left most elements.
  - The indices 1 and 3 denote the part of the list with the 3 middle elements.

quickSort – PSEUDO CODE

```
quickSort(list, leftIndex, rightIndex) {
    // Base case:
    if(leftIndex >= rightIndex) {
        return; // done!
    }









}
```

quickSort – PSEUDO CODE

```
quickSort(list, leftIndex, rightIndex) {
    // Base case:
    if(leftIndex >= rightIndex) {
        return; // done!
    } else { // recursive step:
        i ← placeAndDivide(list, leftIndex, rightIndex)
        // i = index where the pivot is placed
        quickSort(list, leftIndex, i-1)
        quickSort(list, i+1, rightIndex)
    }
}
```

placeAndDivide – PSEUDO CODE

```
placeAndDivide(list, leftIndex, rightIndex) {

        // pick the right most element
        pivot ← list.get(rigthIndex)

}
```

placeAndDivide – PSEUDO CODE

```
placeAndDivide(list, leftIndex, rightIndex) {

        // pick the right most element
        pivot ← list.get(rigthIndex)

        // place the wall to the left
        wall ← leftIndex -1




}
```

placeAndDivide – PSEUDO CODE

```
placeAndDivide(list, leftIndex, rightIndex) {

        // pick the right most element
        pivot ← list.get(rigthIndex)

        // place the wall to the left
        wall ← leftIndex -1

        // go through all elements and compare them to the pivot
        for(int i=leftIndex; i< rigthIndex; i++) {




        }



}
```

placeAndDivide – PSEUDO CODE

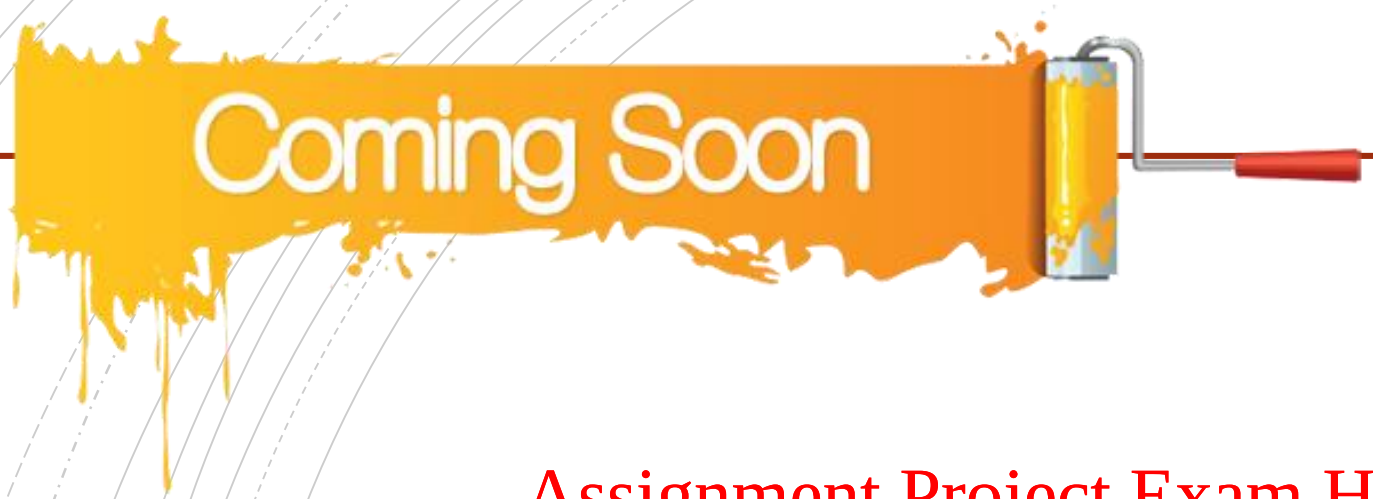
```
placeAndDivide(list, leftIndex, rightIndex) {

        // pick the right most element
        pivot ← list.get(rigthIndex)

        // place the wall to the left
        wall ← leftIndex -1

        // go through all elements and compare them to the pivot
        for(int i=leftIndex; i< rigthIndex; i++) {

                if(list.get(i)<pivot) {
                        wall++; // move wall
                        swap list.get(i) list.get(wall)// move element behind wall
                }
        }



}
```

placeAndDivide – PSEUDO CODE

```
placeAndDivide(list, leftIndex, rightIndex) {

        // pick the right most element
        pivot ← list.get(rigthIndex)

        // place the wall to the left
        wall ← leftIndex -1

        // go through all elements and compare them to the pivot
        for(int i=leftIndex; i< rigthIndex; i++) {

                if(list.get(i)<pivot) {
                        wall++; // move wall
                        swap list.get(i) list.get(wall)// move element behind wall
                }
        }

        swap list.get(rigthIndex) list(wall+1) // move pivot next to wall

        return wall+1;
}
```

MERGESORT VS. QUICKSORT

- Mergesort typically uses an extra list. More space can hurt performance for big lists.
- We will discuss worst case performance of quicksort later in the course.
- See stackoverflow if you want opinions on which is better. The answer is, it depends …



In the next video:

- Recurrences