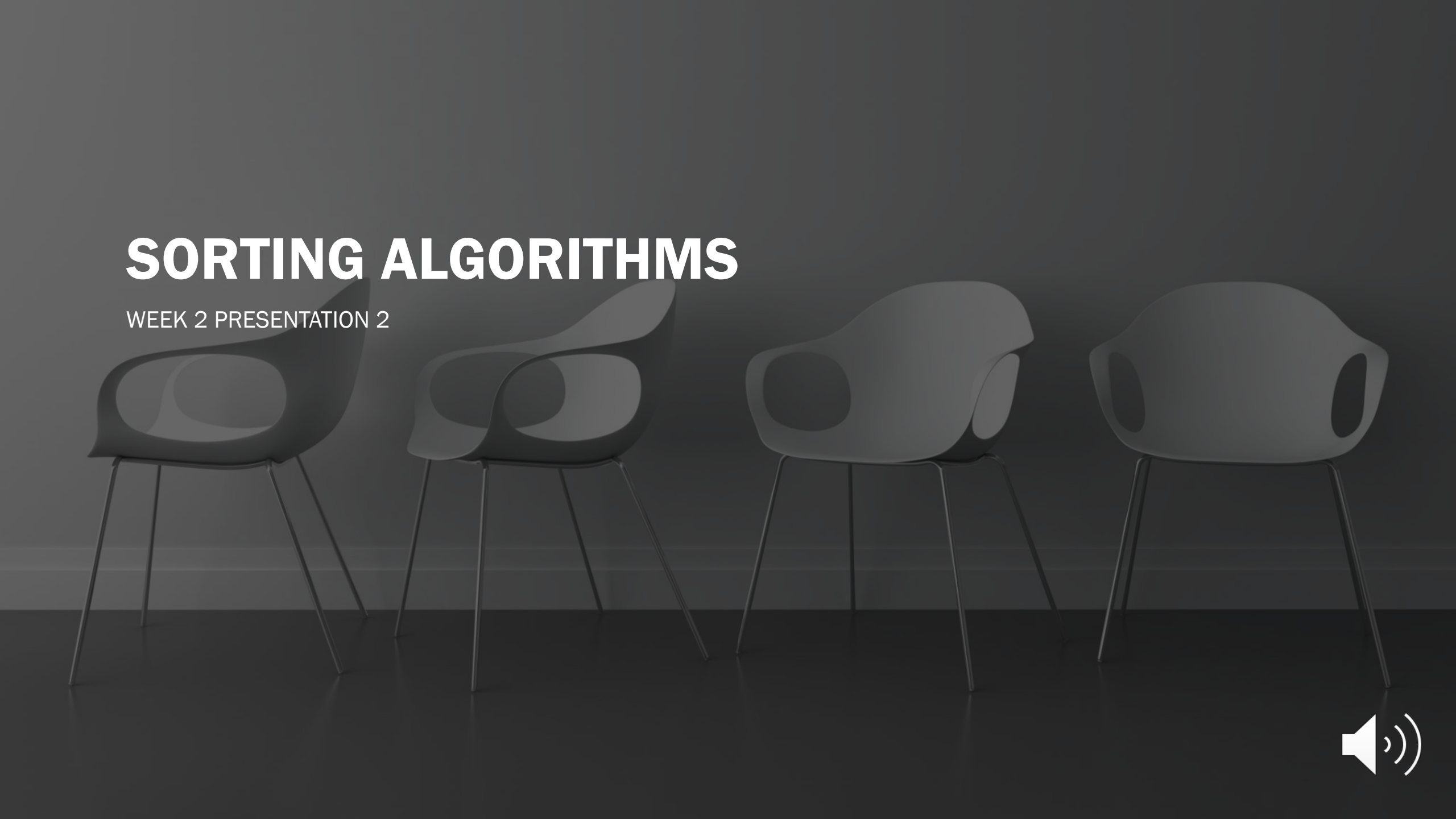
SORTING ALGORITHMS

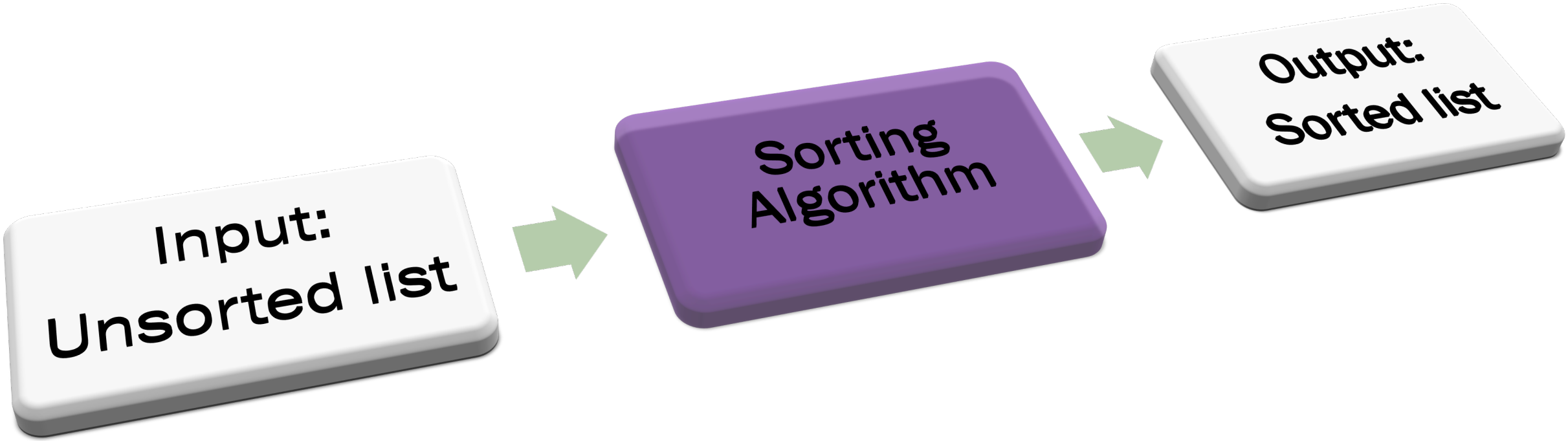
WEEK 2 PRESENTATION 2



SORTING

- Before computing sorting was a major bottleneck, slowing many operations, by requiring huge amounts of time. When the tabulating machine company (that later became IBM) was able to automate sort operations in the 1890s they sped up US Census data collection and reduced the costs dramatically.
- Computers mostly help us manage data; most programs, before they compute anything, have to retrieve data. And for retrieving data easily, data must be sorted, or kept in order. Sorting efficiently is a very important topic.

SORTING

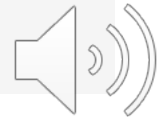


SORTING ALGORITHMS

- Insertion sort
- Bubble sort
- Selection sort
- Quicksort
- Merge sort
- Heap sort

INSERTION SORT

```
function insertion_sort(list)
    for i ← 2 … list.length
        j ← i
        while j and list[j-1] > list[j]
            list.swap_items(j, j-1)
            j ← j - 1
```



BUBBLE SORT

You may have heard about "bubble sort": you start at the left end of the array and move to the right, exchanging values that aren't in the correct order. At the end of the first pass, the biggest value is correctly placed. Then you repeat, to place the 2nd biggest, and so forth.

QUICK SORT AND MERGE SORT

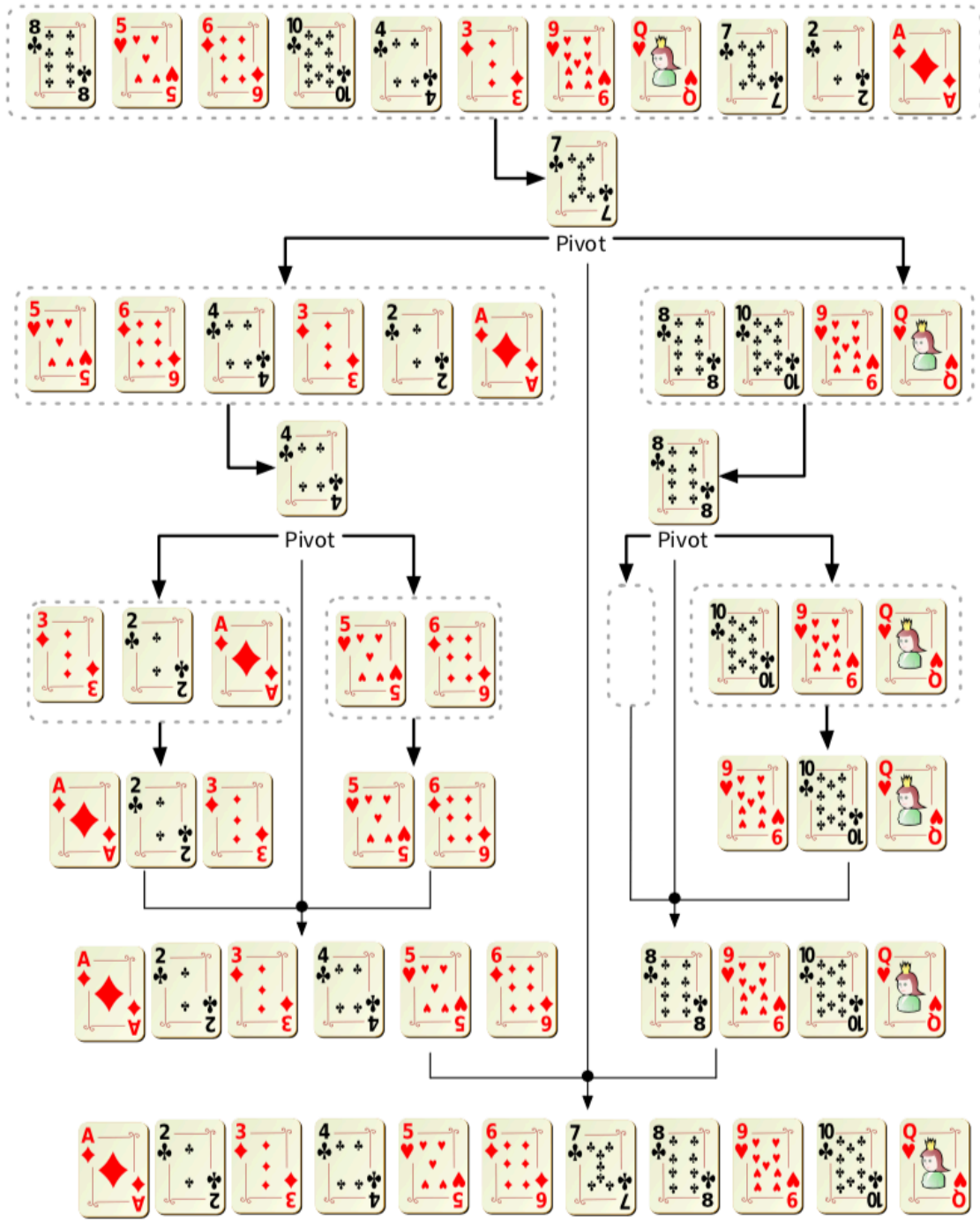
- For large lists that may not be nearly sorted it is necessary to use faster algorithms such as merge sort or quick sort
- These algorithms work by divide and conquer
  - Large problem is successively reduced to smaller sub problems until they are small enough to solve easily
  - Then the solution to the original problem is constructed from the pieces
  - Uses recursion

QUICK SORT

Using quick sort to sort a a pile of cards:

1. If the list has fewer than 4 cards, put them in the right order and finish, else go to step 2
2. Choose at random any card from the pile to be the pivot
3. Cards larger than the pivot go into a new pile to the right; cards smaller than the pivot go to the left
4. Start this procedure for each of the two piles to get a sorted pile
5. Join the left pile, pivot and right pile to get a sorted pile

Exercise: get a deck of cards (or other sortable objects) and apply these steps to strengthen your understanding of quick sort.


Pivot
Pivot
Pivot

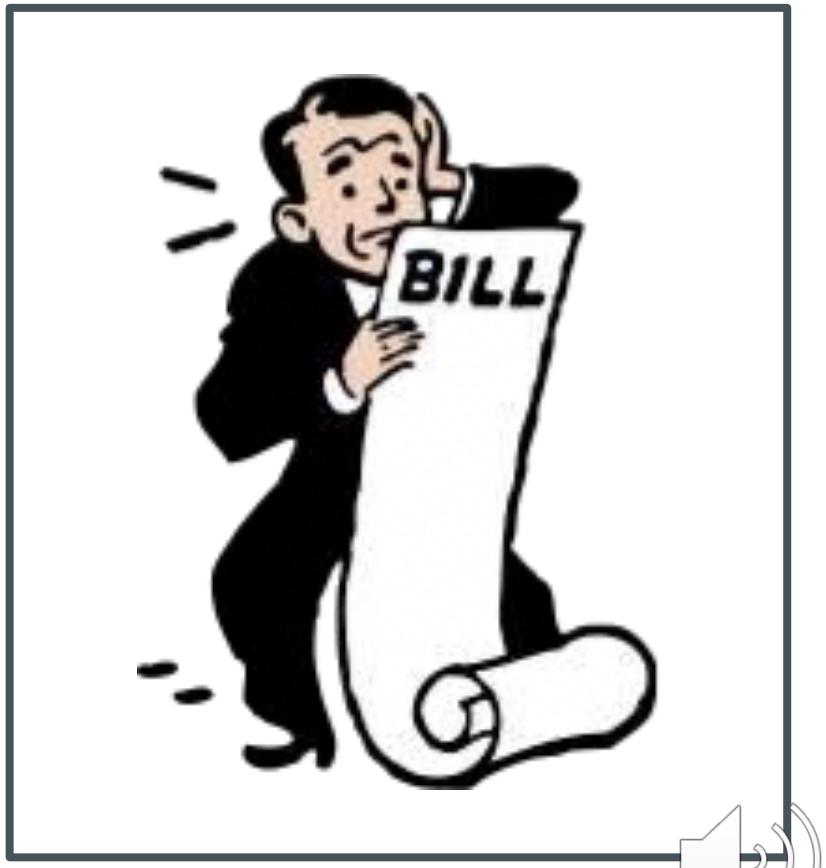
REMEMBER

- There are many sorting algorithms
- Each is suitable for a specific situation although in the average case quick sort and merge sort are faster
- We will discuss what we mean by faster in the next presentation

ALGORITHM COMPLEXITY

- Time complexity
- Space complexity

- Loops



```
42
43  void bubbleSort(int arr[])
44  {
45      int n = array.length;
46      for (int i = 0; i < n-1; i++)
47          for (int j = 0; j < n-i-1; j++)
48              if (array[j] > array[j+1])
49              {
50                  // swap
51                  int tmp = array[j];
52                  array[j] = array[j+1];
53                  array[j+1] = tmp;
54              }
55  }
56
```

First pass:
n – 1 comparisons between elements
n/2 swaps (assume on average 50% out of order)

Second pass:
n-2 comparisons between elements
(n-1)/2 swaps (on average)

...

Time complexity in terms of operations performed:

Number of comparisons: $n(n-1)/2$

Number of swaps: $1/4\ (n + 2)(n - 1)$

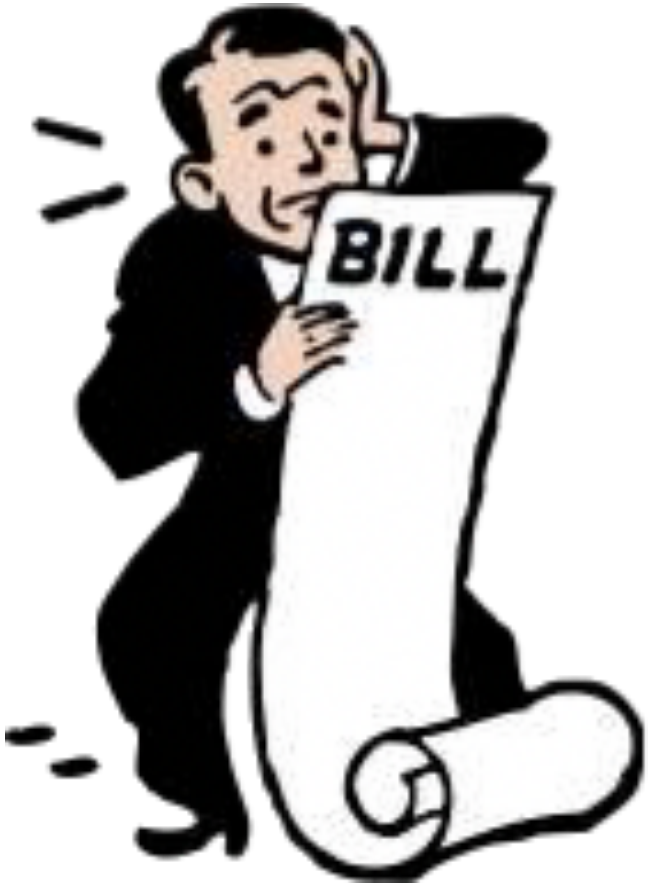
$O(n^2)$

ALGORITHM COMPLEXITY

When people study algorithms, they try to measure how algorithms will perform when applied to larger volumes of data.
This is extremely important, because some operations require a truly astronomical number of computations, taking hours, days, weeks or sometimes years …

Picking the right algorithm makes the difference. Algorithm complexity is specified using the O (big o) notation, that says how the time increases when the number n of objects to process increases.

Time complexity refers to runtime, space complexity refers to memory requirements.



O(1)

Means that the time is constant and doesn't depend on the number of objects processed. This is what happens when in a program you access the nth element in an array – whether you access the first or last element, time is the same, and doesn't depend either on the size of the array.

O(LOG N)

Means that the time evolves as the log. So, operating on 1000 more objects will multiply the time by 3 only (the log isn't necessarily decimal, it's the order of magnitude that counts), which is pretty good.

O(N)

Means that the time is directly proportional to the number of items processed. This is what you get when you scan an unordered array, looking for a value: if the size of the array doubles, it will take twice the time on average.

O(N LOG N)

Means that processing 1,000 times the number of objects will take about 3,000 more time. That's what the best sort algorithms can do.

$O(N^2)$

means that multiplying the number of values by 1,000 will multiply the time by 1,000,000. That's what insertion sort and bubble sort typically do. You definitely don't want to use that on large numbers of values.

$O(N^2)$

Insertion sort

Bubble sort

O(N LOGN)

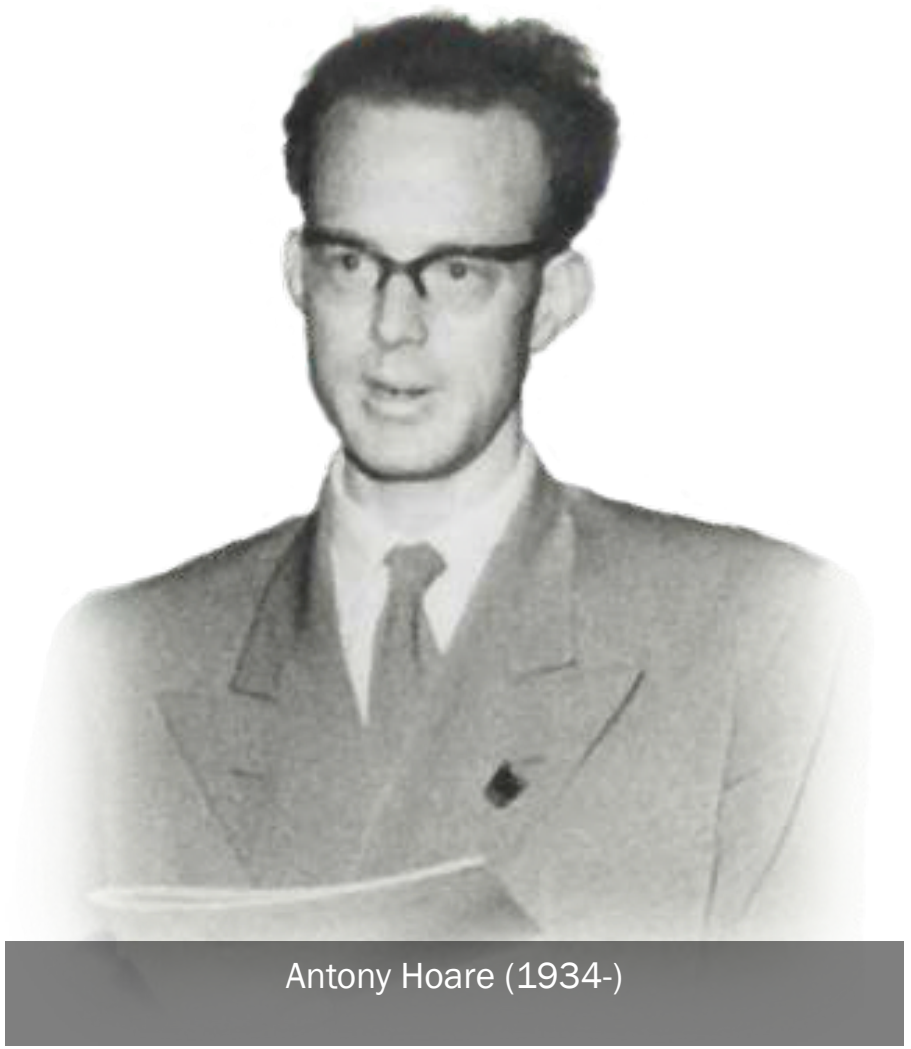
I have said to you, the best sorting algorithms are O(nlogn) algorithms. What does it mean compared to a O(n2) algorithm?

“N LOG N” VS “N^2”:
INCREASES FASTER THAN N BUT NOT MADLY FASTER

N	N^2	N Log N
10	100	10
100	10000	200
1000	1000000	3000

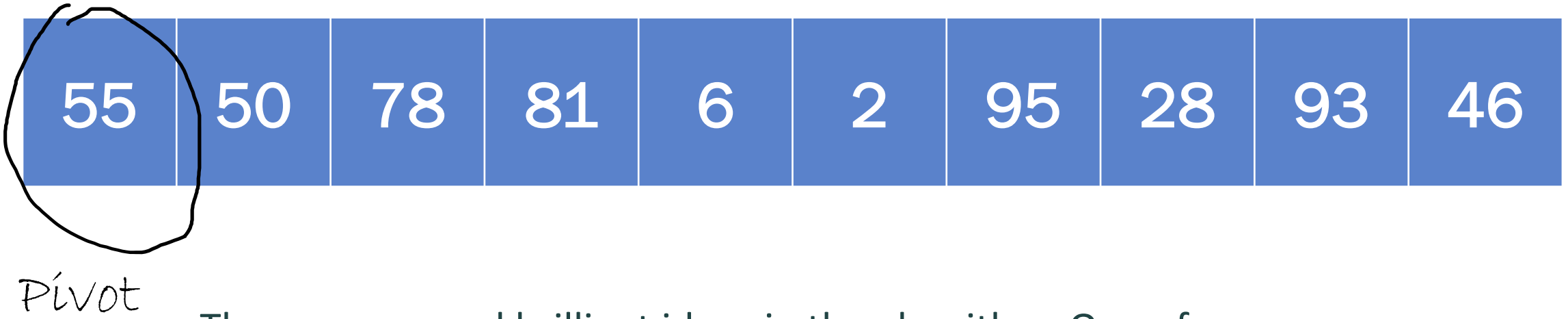
QUICKSORT

The first really fast sorting algorithm was invented by Antony Hoare in the early 1960s when he was in Moscow.



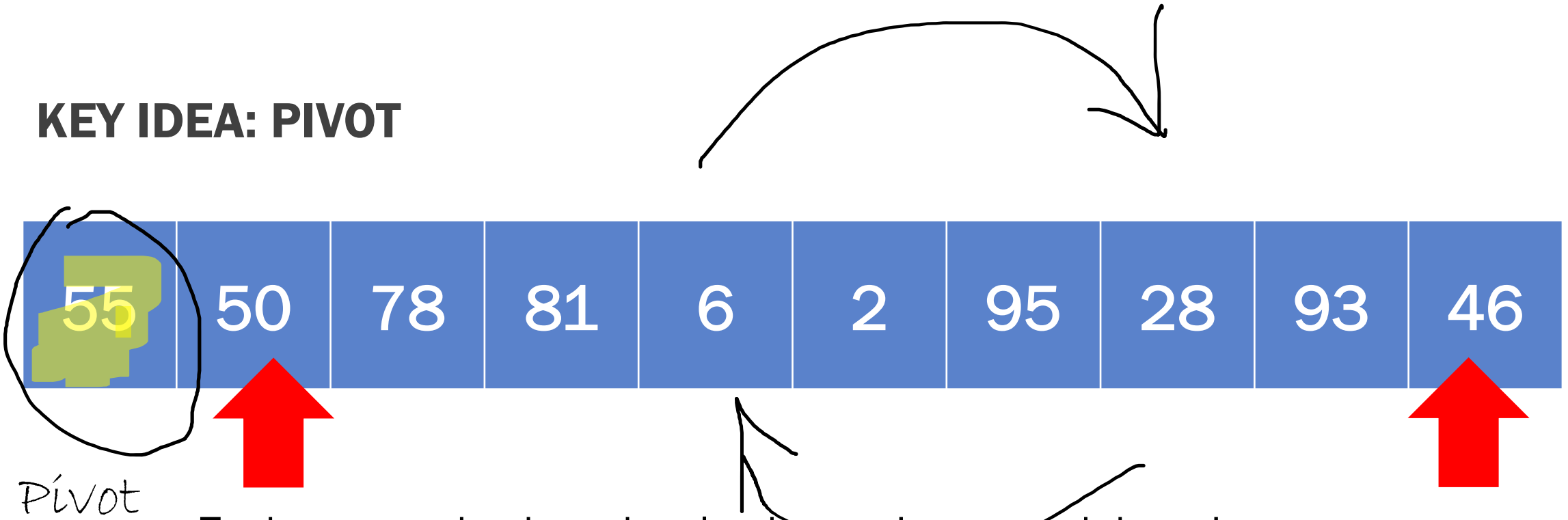
Antony Hoare (1934-)

KEY IDEA: PIVOT



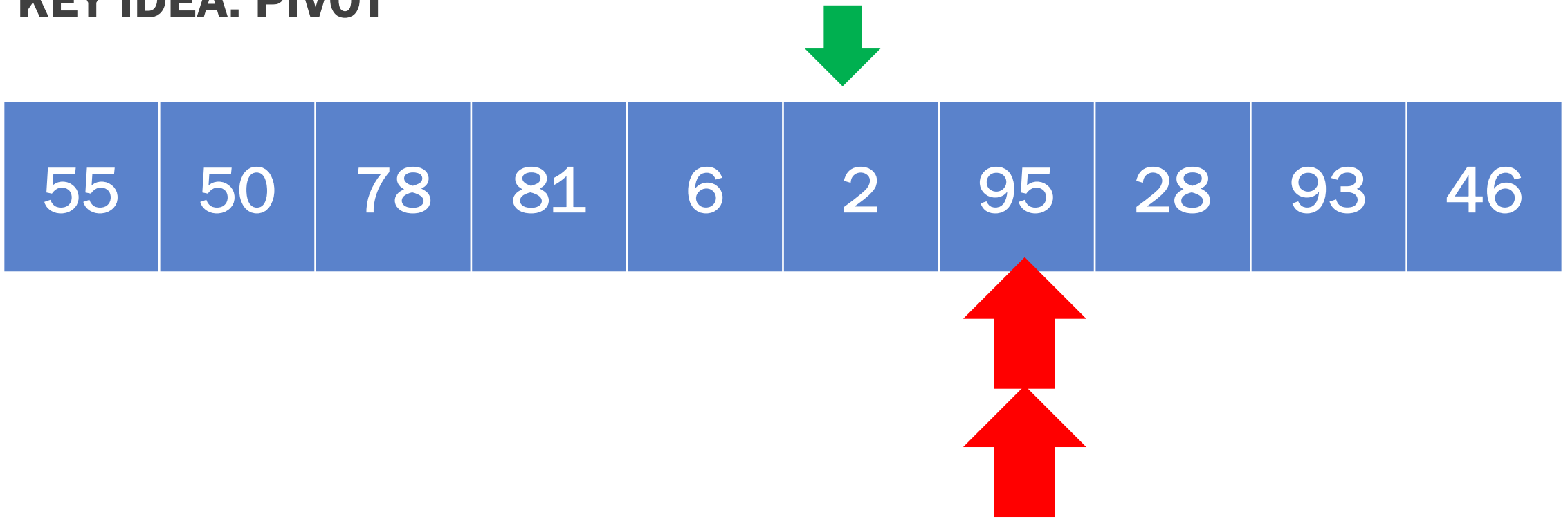
There are several brilliant ideas in the algorithm. One of them is, instead of successively looking for smallest (or greatest) values, to take arbitrarily one value called pivot and find its final location.

KEY IDEA: PIVOT

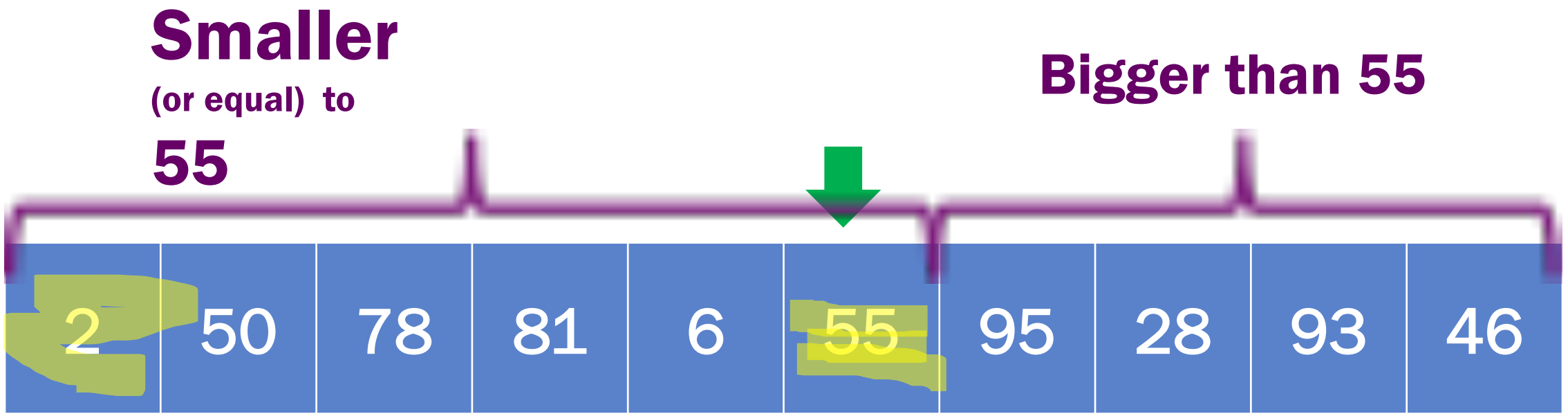


To do so, we check each value by moving up and down in the array at once, and stopping when the "up" pointer encounters a greater value than the pivot, and the "down" pointer a smaller one. Those values are swapped.

KEY IDEA: PIVOT



When both pointers meet, everything on the right is bigger than the pivot, everything on the left is smaller (or equal) and we know that the final pivot location will be where the small green arrow points.



By swapping the pivot and the value occupying "its" place, we partition the original set into a subset containing smaller values, and a subset containing greater values. We "just" need to sort these two subsets.

KEY IDEA: “DIVIDE AND CONQUER”

$$O(N^2)$$

Sorting twice the number of values is 4 times as costly

As we have seen, most simple sorting algorithms have a cost in number of operations (and time) that increases as the square of the number of values sorted.

So it's better to perform two sorts applied to N values each than one sort applied to 2N values.

JAVA CODE TO PLACE PIVOT

Here is the Java code to find where the pivot goes: first we move two indices 'up' and 'down' until they find a value respectively greater and smaller than the pivot ...

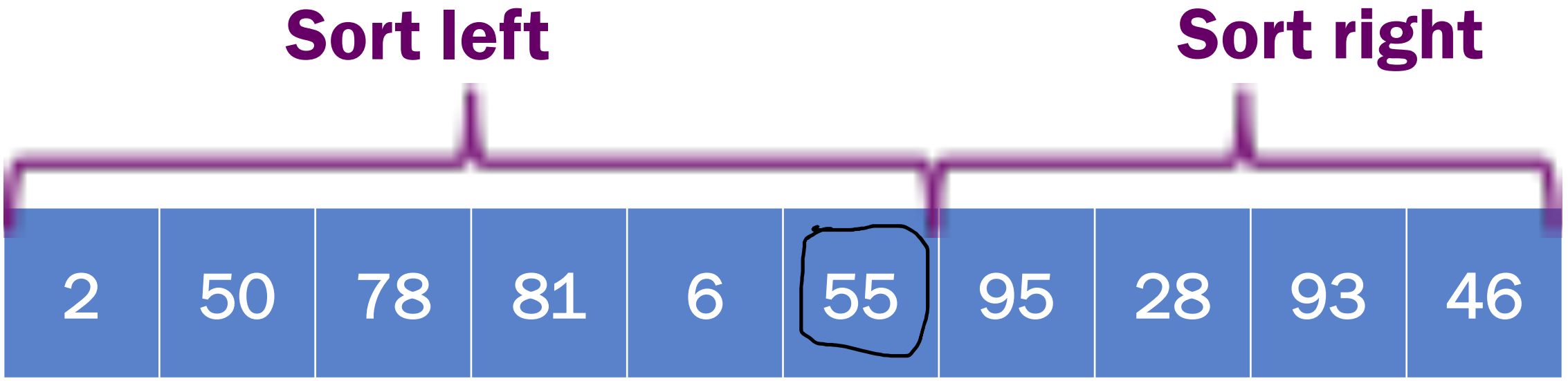
```
static int placePivot(int[] arr, int first, int last) {
    int pivot;
    int tmp;
    int up = first + 1; int down = last;

    pivot = arr[first]; while (down > up) {
        while ((arr[up] <= pivot) && (up < down)) {
            up++;
            }
            while ((arr[down] > pivot) && (up < down)) {
            down--;
            }
    }
}
```

JAVA CODE TO PLACE PIVOT

... then values are swapped. In the end, we return the pivot should go, which is the limit between the smaller subset and the bigger subset.

```
            if (up < down) {
                // Exchange values
                tmp = arr[up];
                arr[up] = arr[down];
                arr[down] = tmp;
            }
    }
    // up has stopped at a value > pivot
    // or when it has met the down pointer
    if  (pivot < arr[up]) {
        // Place pivot at up - 1
        up --;
    }
    arr[first] = arr[up];
    arr[up] = pivot;
    return up;
}
```

The problem is that we must defer operations, sorting one subset, then the other, and if we apply each time the same "recipe" it can become very complicated.



"Remember" that we must sort 0 to 4 Sort 6 to 9

"Remember" that we must sort x to y

Sort w to Z

"Remember" ...

MATHEMATICAL INDUCTION

- Thankfully, maths come to the rescue to solve these parenthesis; not maths itself, but a mathematical method which is very closely related to what we'll do.
- This closely related mathematical method is induction, a very clever way of proving theorems.
- More in discrete maths