INFS7901
Database Principles

Sorting

Hassan Khosravi & Ash Rahimi

Learning Objective

Description	Tag
Describe and apply selection sort.	Sorting
Describe and apply insertion sort.	
Describe and apply merge sort.	
Describe and apply quicksort.	
Compare and contrast the trade-offs of different sorting algorithms.	
Explain what is meant by a stability in sorting algorithms and determine which sorting algorithms are stable.	
State differences in performance for large files versus small files on various sorting algorithms.	
Analyse the complexity of different sorting algorithms.	
Manipulate data using various sorting algorithms (irrespective of any implementation).	
Reflect on the importance of sorting in DBMS	

Introduction

Selection Sort

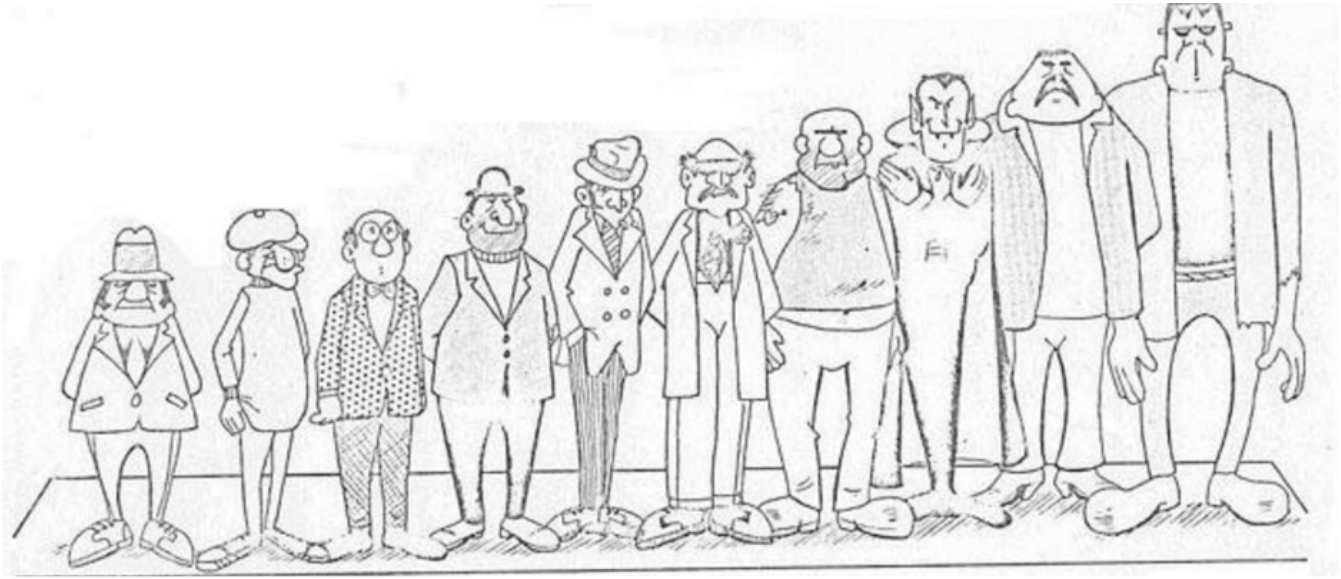
Insertion Sort

MergeSort

QuickSort

Sorting

- Sorting is the process of placing elements from a collection in some kind of order.



(source: www.pleacher.com)

- There are many, many sorting algorithms that have been developed and analyzed, suggesting that sorting is an important area of study in computer and data science.

Applications of Sorting

Web search

Shortest path

Database Management Systems

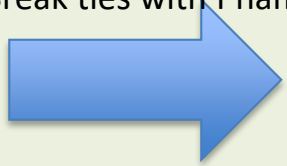
Categorizing Sorting Algorithms

- Computational complexity
  - Average case behaviour: Why do we care?
  - Worst/best case behaviour: Why do we care?
- Memory Usage: How much *extra* memory is used?
- Stability: A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted.

First name	Last name
Tyrion	Lannister
Cersei	Lannister
Daenerys	Targaryen
Jaime	Lannister

Sort by l name

Break ties with f name



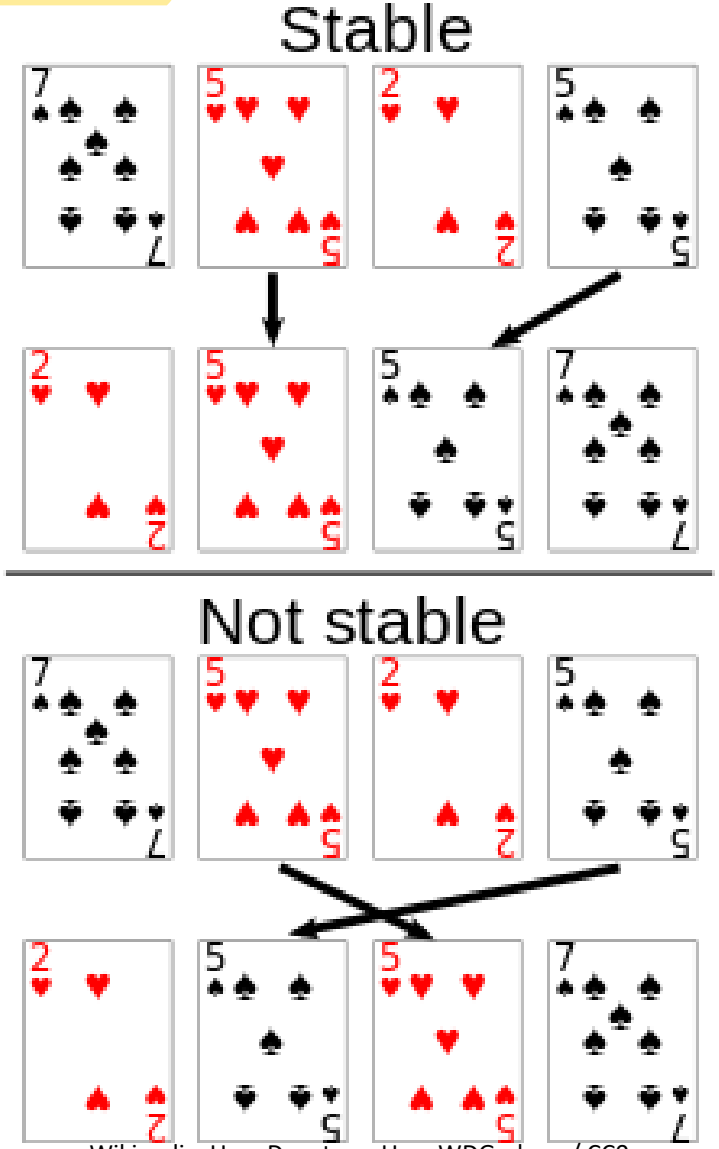
Step 1

First name	Last name
Cersei	Lannister
Daenerys	Targaryen
Jaime	Lannister
Tyrion	Lannister

Step 2

First name	Last name
Cersei	Lannister
Jaime	Lannister
Tyrion	Lannister
Daenerys	Targaryen

Stable vs. Not Stable



Wikipedia: User:Dcoetzee, User:WDGraham / CC0

Introduction

Selection Sort

Insertion Sort

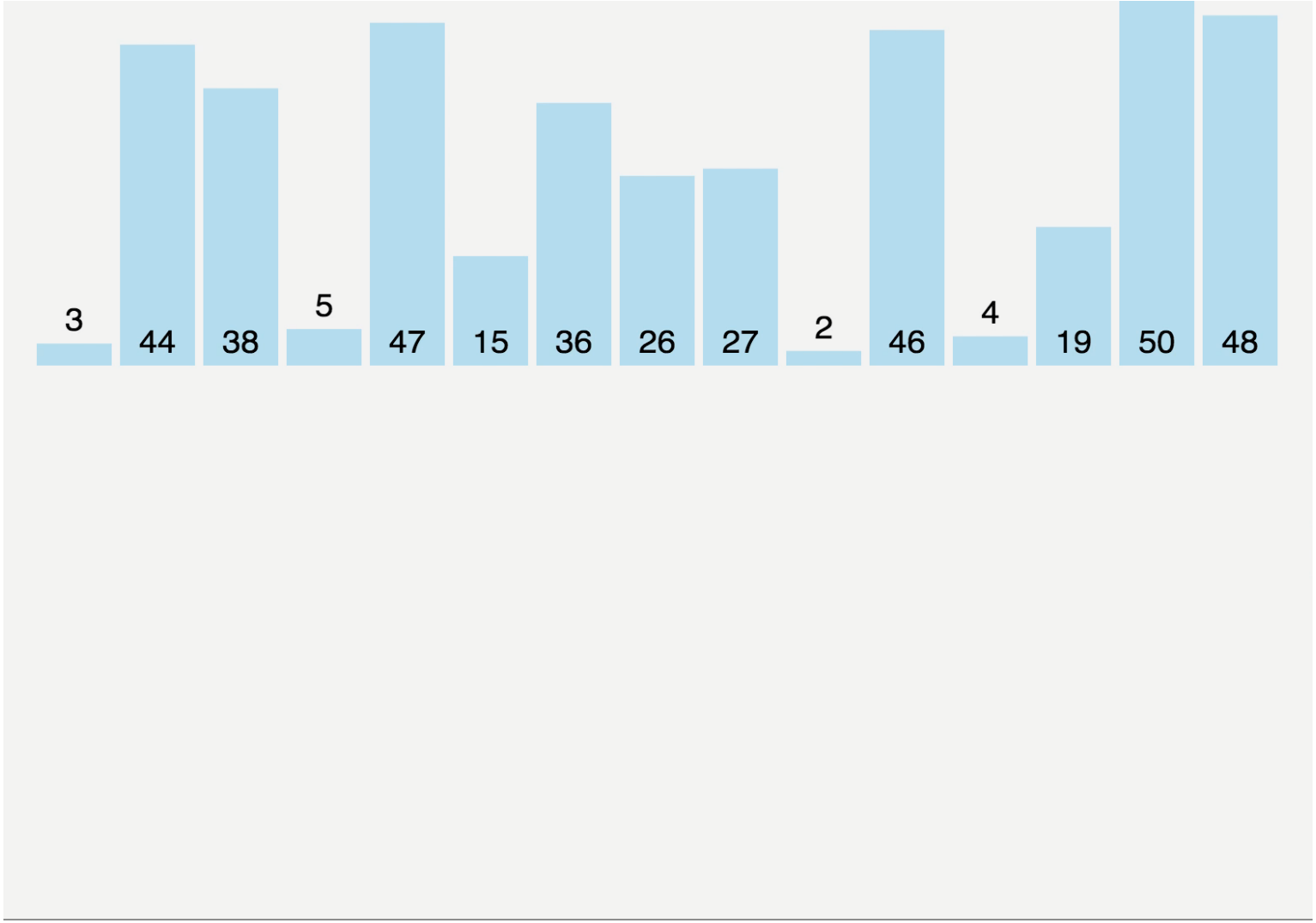
MergeSort

QuickSort

Selection Sort

- Sorts an array by repeatedly finding the smallest element of the unsorted tail region and moving it to the front.

Algorithm Visualisation



Choose Selection sort on https://visualgo.net/en/sorting

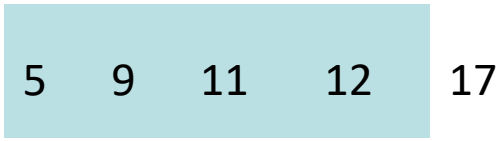
Selection Sort

- Find the smallest and swap it with the first element

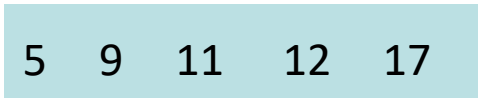
- Find the next smallest. It is already in the correct place

- Find the next smallest and swap it with first element of unsorted portion

- Repeat



- When the unsorted portion is of length 1, we are done



In-Class Exercise

- Write out all the steps that selection sort takes to sort the following sequence:

91	5	11	90	6	16	31	88

In-Class Exercise

- Write out all of the steps that selection sort takes to sort the following sequence:

91	5	11	90	6	16	31	88
5	91	11	90	6	16	31	88
5	6	11	90	91	16	31	88
5	6	11	90	91	16	31	88
5	6	11	16	91	90	31	88
5	6	11	16	31	90	91	88
5	6	11	16	31	88	91	90
5	6	11	16	31	88	90	91

Clicker Question

- What is the time complexity of selection sort in the best and worst case.
- *A: $O(n^2)$, $O(n^2)$*
- *B: $O(n)$, $O(n^2)$*
- *C: $O(n \lg n)$, $O(n \lg n)$*
- *D: $O(n \lg n)$, $O(n^2)$*
- *E: $O(n)$, $O(n \lg n)$*

Clicker Question (answer)

- What is the time complexity of selection sort in the best and worst case.
- *A: $O(n^2)$, $O(n^2)$*
- *B: $O(n)$, $O(n^2)$*
- *C: $O(n \lg n)$, $O(n \lg n)$*
- *D: $O(n \lg n)$, $O(n^2)$*
- *E: $O(n)$, $O(n \lg n)$*

Selection Sort

```
# Given a disordered list of integers (or any other items),
# rearrange the integers in natural order.
#
# Time Complexity of Solution:
# Best O(n^2); Average O(n^2); Worst O(n^2).
#
# Approach:
# The algorithm proceeds by finding the smallest element in the
# unsorted sublist, swapping it with the leftmost unsorted element
#============================================================
def selectionsort( aList ):
        for i in range( len( aList ) ):
                least = i
                for k in range( i + 1 , len( aList ) ):
                        if aList[k] < aList[least]:
                                least = k
                swap( aList, least, i )

def swap( A, x, y ):
        tmp = A[x]
        A[x] = A[y]
        A[y] = tmp
```

Clicker Question

- Is selection sort stable?
- *A: Yes*
- *B: No*
- *C: I don't know*

Clicker Question

- Is selection sort stable?
- *A: Yes*
- *B: No*
- *C: I don't know*

90	5	11	90	6	16	2	88

2	5	11	90	6	16	90	88

Selection sort can be made Stable while still running in $O(n^2)$, but it may run significantly slower

When is the Selection Sort algorithm used?

- One advantage of selection sort is that it requires only *O(n)* write operations. If we have a system where write operations are extremely expensive and read operations are not, then selection sort could be ideal. One such scenario would be if we are sorting a file in-place on flash memory or an external hard drive.

Name	Best	Average	Worst	Stable	Memory
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	challenging	O(1)

Introduction

Selection Sort

Insertion Sort

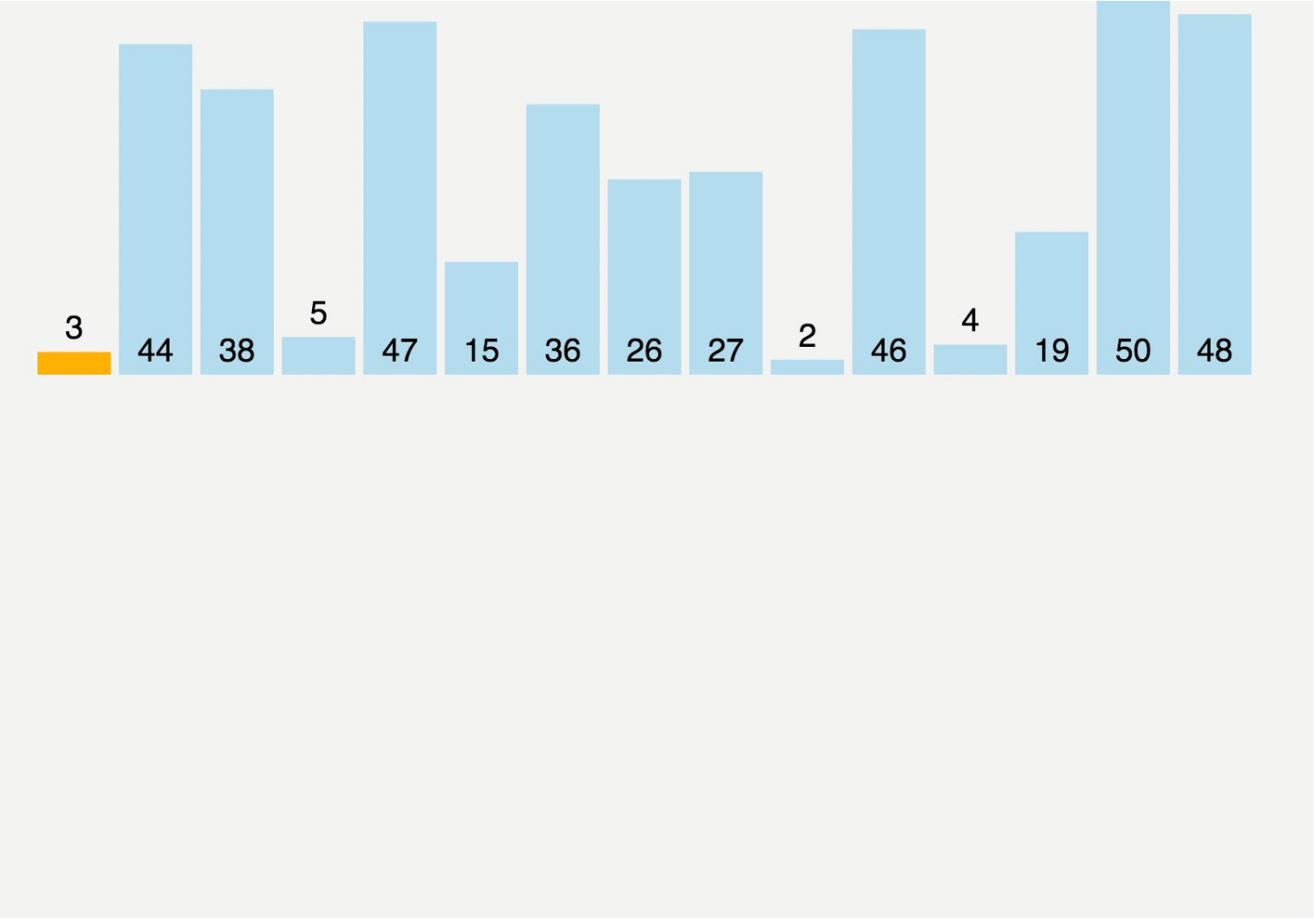
MergeSort

QuickSort

Insertion Sort

- Given a list, take the current element and insert it at the appropriate position of the list, adjusting the list every time you insert

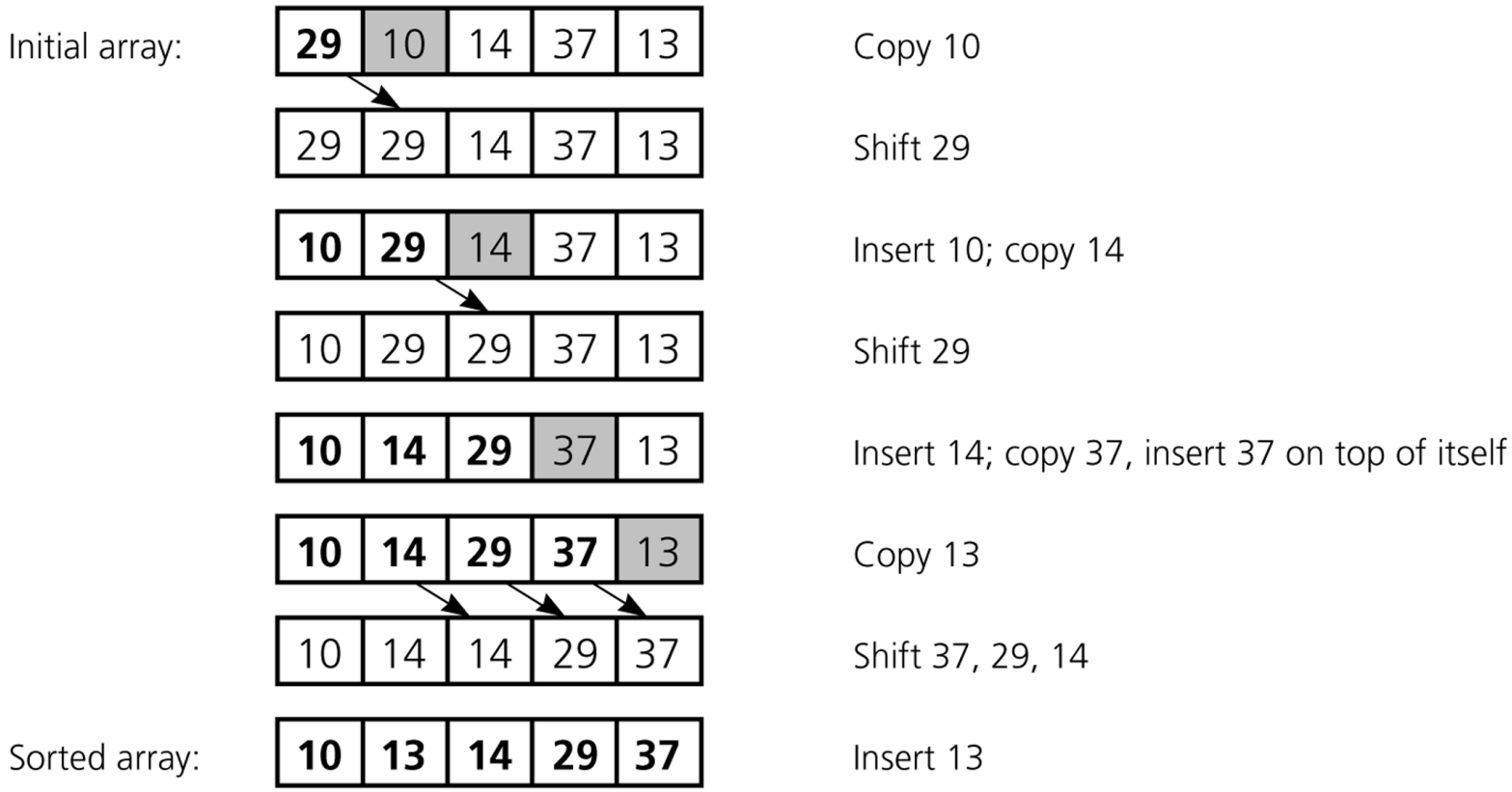
Algorithm Visualisation



Choose Insertion sort on https://visualgo.net/en/sorting

In-class exercise

- Write out all of the steps that insertion sort takes to sort the following sequence: 29 10 14 37 13



Clicker question

- What is the time complexity of Insertion Sort in the best and worst case.
- *A: $O(n^2)$, $O(n^2)$*
- *B: $O(n)$, $O(n^2)$*
- *C: $O(n \lg n)$, $O(n \lg n)$*
- *D: $O(n \lg n)$, $O(n^2)$*
- *E: $O(n)$, $O(n \lg n)$*

Clicker question (answer)

- What is the time complexity of Insertion Sort in the best and worst case.

B: $O(n)$, $O(n^2)$

a1 a2 a3 a4 a5

- Best case $\sum_{i=1}^{n} 1 = n \in O(n)$
- Worst case $\sum_{i=1}^{n} i = n(n+1)/2 \in O(n^2)$
- Average case $\sum_{i=1}^{n} i/2 = n(n+1)/4 \in O(n^2)$

Insertion Sort

```
# Given a disordered list of integers (or any other items),
# rearrange the integers in natural order.
#
# Time Complexity of Solution:
# Best O(n); Average O(n^2); Worst O(n^2).
#
# Approach:
# Given a list, take the current element and insert it at the appropriate
# position of the list, adjusting the list every time you insert
#===============================================================
def insertionsort( aList ):
    for i in range( 1, len( aList ) ):
        tmp = aList[i]
        k = i
        while k > 0 and tmp < aList[k - 1]:
            aList[k] = aList[k - 1]
            k -= 1
        aList[k] = tmp
```

Clicker Question

- Suppose we are sorting an array of ten integers using a sorting algorithm. After four iterations of the algorithm's main loop, the array elements are ordered as shown here:

  1 2 3 4 5 0 6 7 8 9

A. The algorithm might be either selection sort or insertion sort.

B. The algorithm might be selection sort, but could not be insertion sort.

C. The algorithm might be insertion sort, but could not be selection sort.

D. The algorithm is neither selection sort nor insertion sort.

Clicker Question (answer)

- Suppose we are sorting an array of ten integers using a sorting algorithm. After four iterations of the algorithm's main loop, the array elements are ordered as shown here:

  1 2 3 4 5 0 6 7 8 9

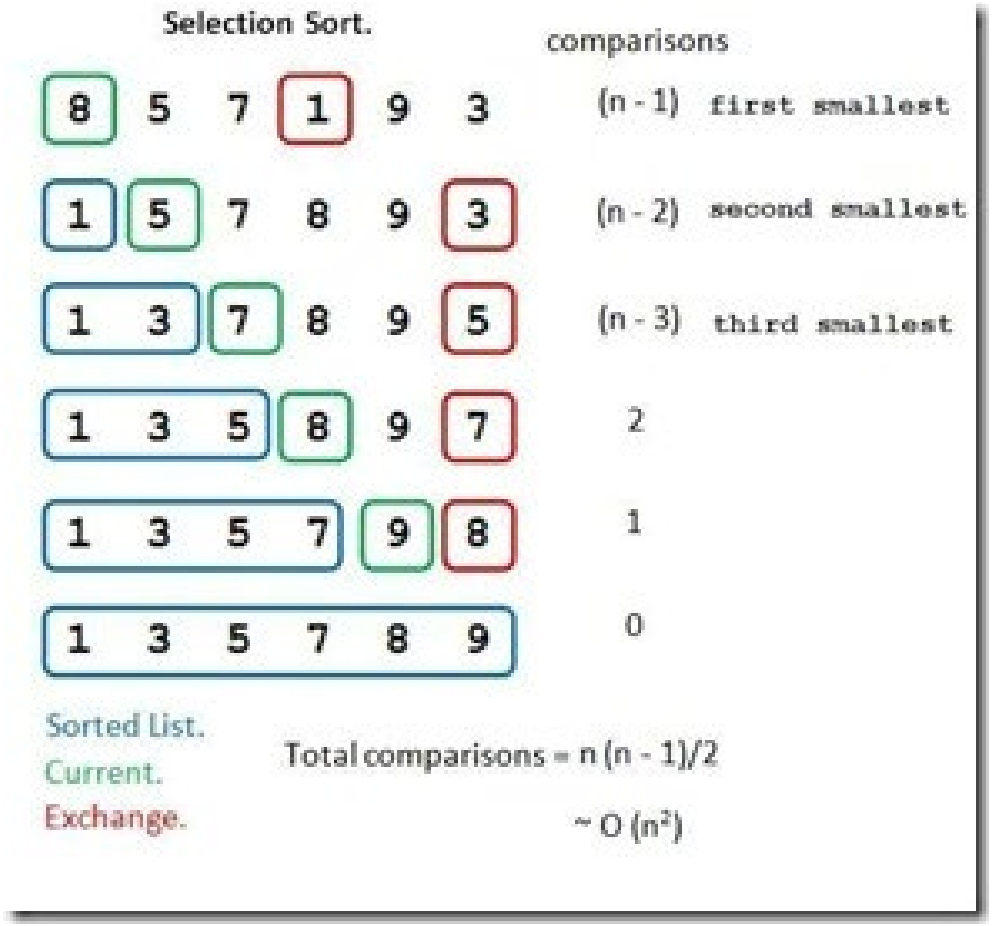
A. The algorithm might be either selection sort or insertion sort.

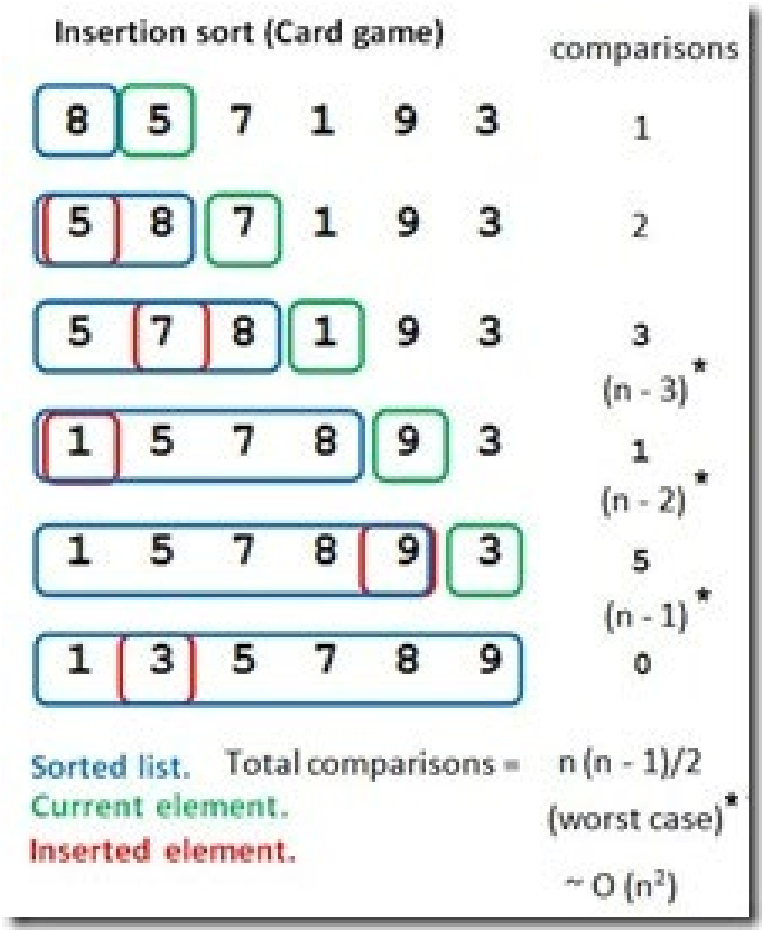
B. The algorithm might be selection sort, but could not be insertion sort.

C. The algorithm might be insertion sort, but could not be selection sort.

D. The algorithm is neither selection sort nor insertion sort.

Selection Sort vs. Insertion Sort





Source

When is the Insertion Sort algorithm used?

- Insertion Sort is the algorithm of choice either when the data is nearly sorted (because it is adaptive) or when the problem size is small (because it has low overhead).

Name	Best	Average	Worst	Stable	Memory
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	challenging	O(1)
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	Yes	O(1)

Introduction

Selection Sort

Insertion Sort

MergeSort

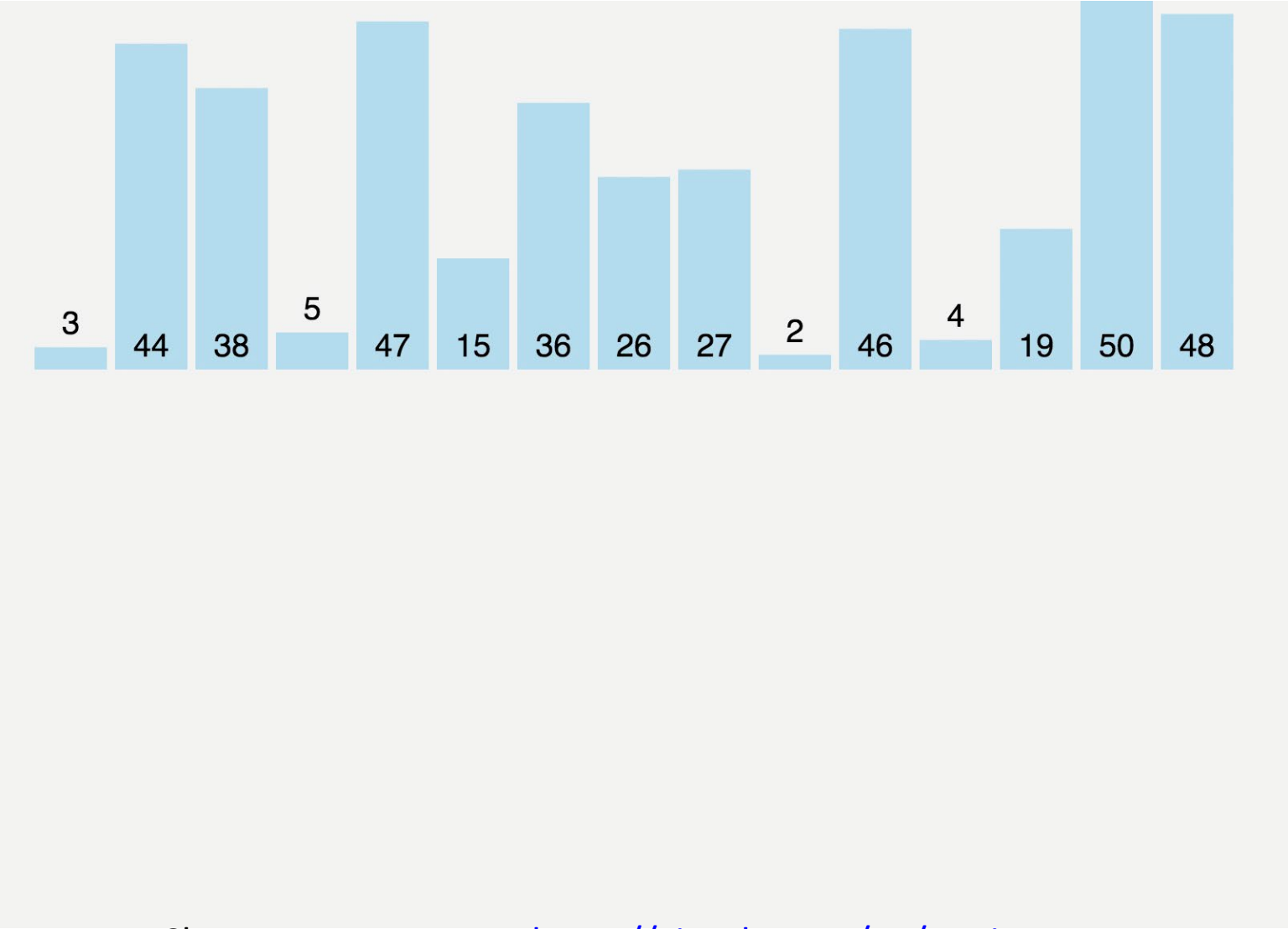
QuickSort

MergeSort

- MergeSort is an example of a divide-and-conquer algorithm that recursively splits the problem into branches, and later combines them to form the solution.
- **Key Steps in MergeSort**:

1. Split the array into two halves.
2. Recursively sort each half.
3. Merge the two (sorted) halves together to produce a bigger, sorted array.
    - Note: The time to merge two sorted sub-arrays of sizes m and n is linear: O(m + n).

Algorithm Visualisation

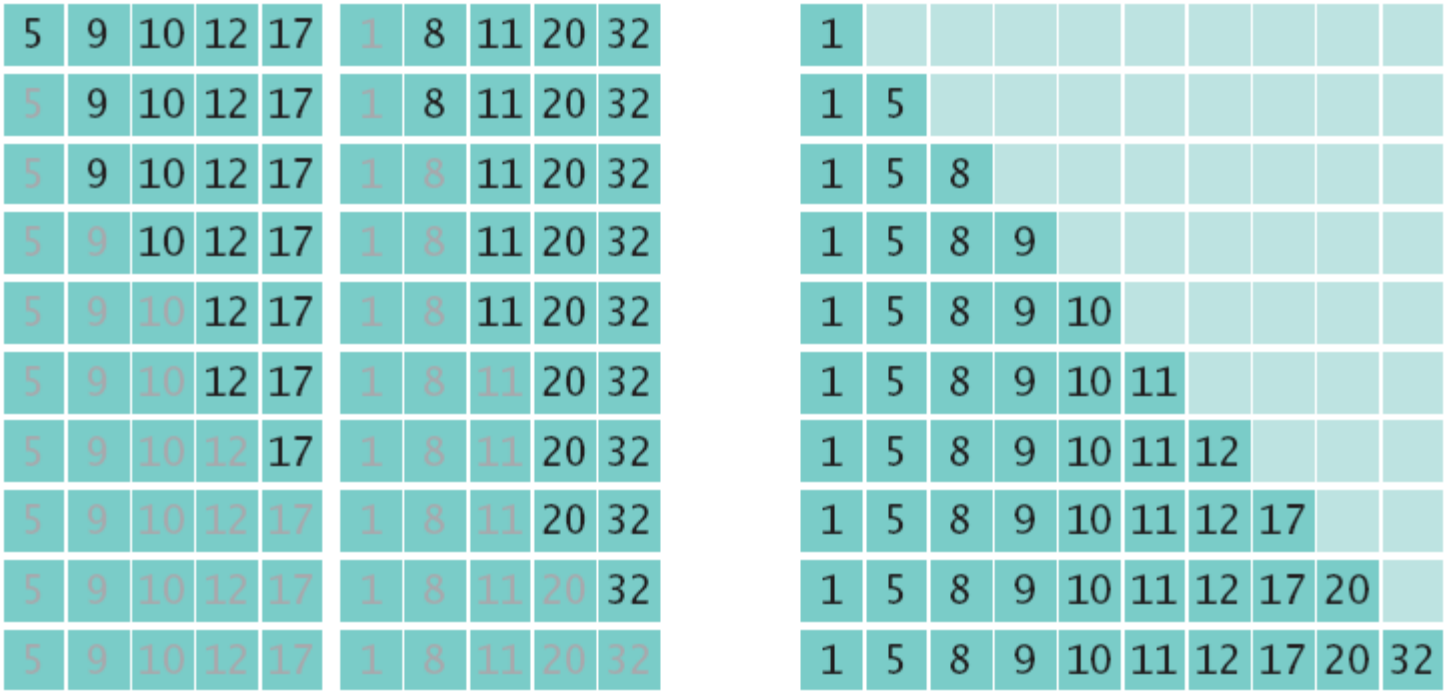


Choose Merge sort on https://visualgo.net/en/sorting

Merging two sorted arrays

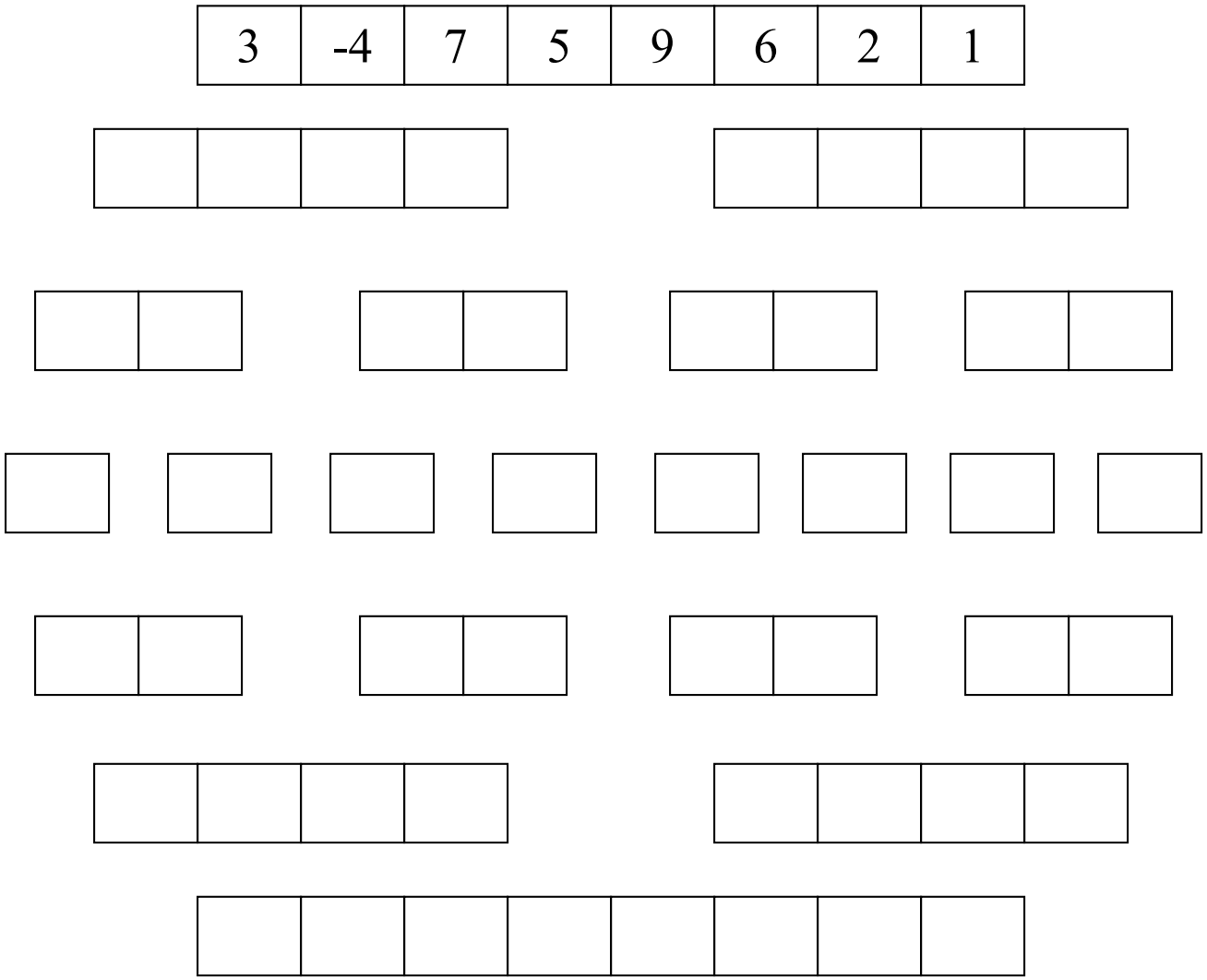
- Divide an array in half and sort each half

- Merge the two sorted arrays into a single sorted array



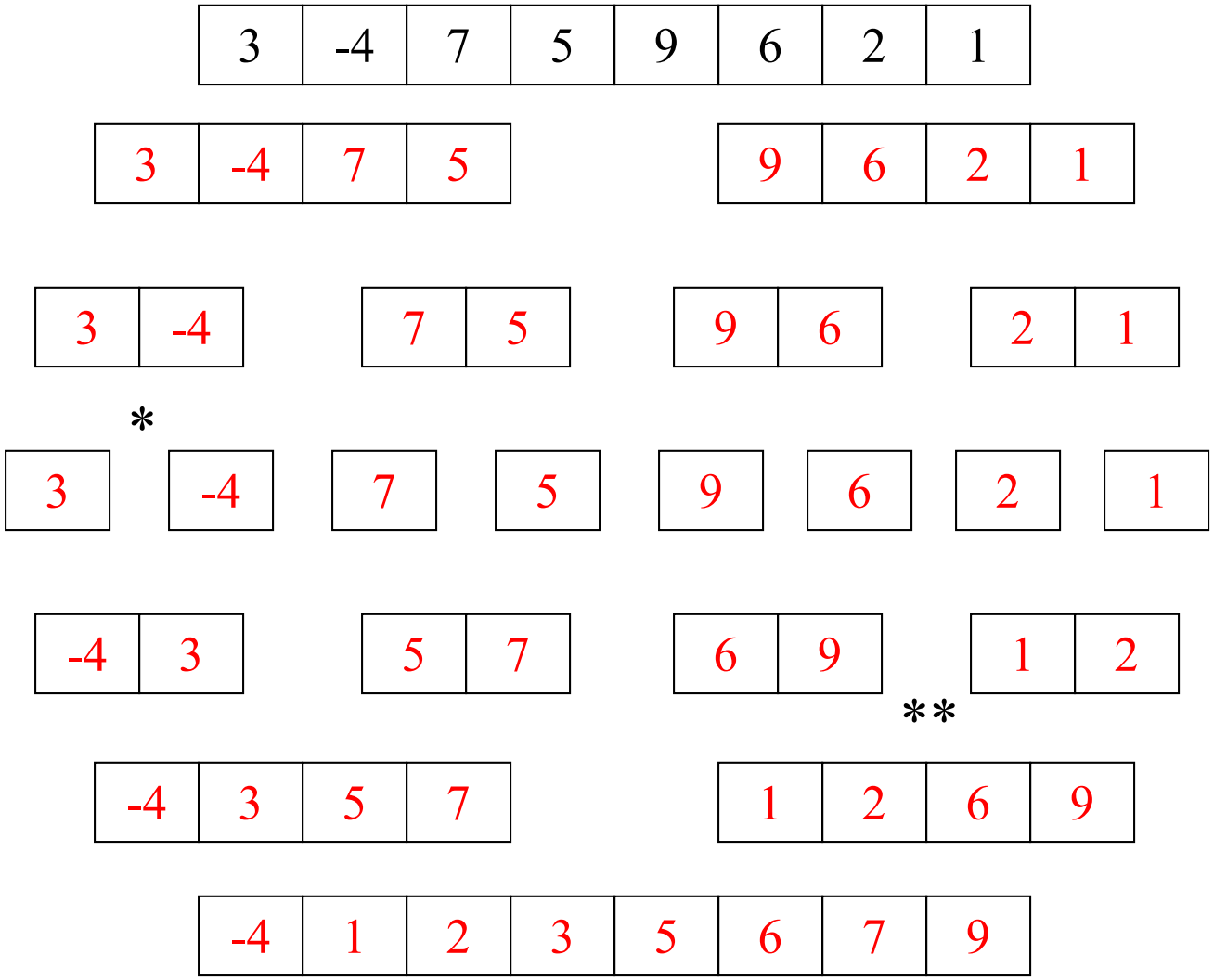
In-class exercise

- Write out all the steps that MergeSort takes to sort the following sequence:



In-class exercise

- Write out all the steps that MergeSort takes to sort the following sequence:



Clicker question

- MergeSort makes two recursive calls. Which statement is true after these recursive calls finish, but before the merge step?

  A. The array elements form a heap.

  B. Elements in each half of the array are sorted amongst themselves.

  C. Elements in the first half of the array are less than or equal to elements in the second half of the array.

  D. None of the above

Clicker question

- MergeSort makes two recursive calls. Which statement is true after these recursive calls finish, but before the merge step?

  A. The array elements form a heap.

  B. Elements in each half of the array are sorted amongst themselves.

  C. Elements in the first half of the array are less than or equal to elements in the second half of the array.

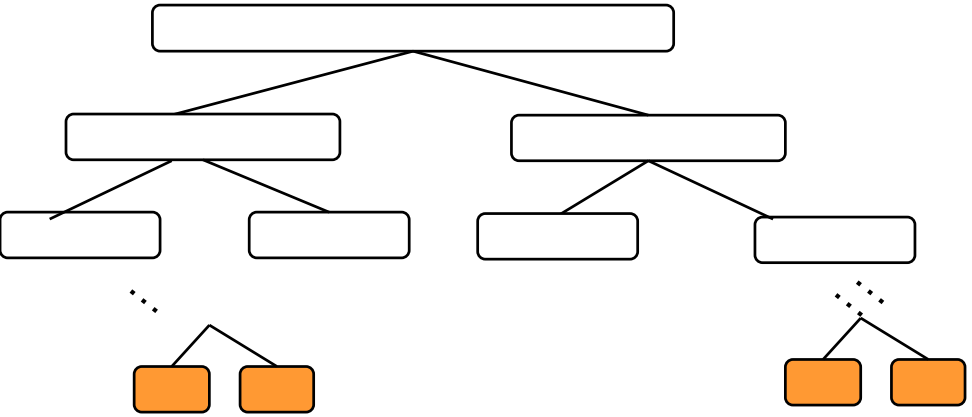
  D. None of the above

```
# Given a disordered list of integers (or any other items),
# rearrange the integers in natural order.
#
# Time Complexity of Solution:
# Best = Average = Worst = O(nlog(n)).
#
# Approach:
# recursively splits the problem into branches, and later combines them # to form the solution.
```

```
def merge_sort(A):
    n = len(A)
    if n>1:
        mid = n//2
        A1 = A[:mid]
        A2 = A[mid:]
        # recursive calls:
        merge_sort(A1)
        merge_sort(A2)
        # merge solutions
        i=0
        j=0
    ...
```

```
    ...
        while i<mid and mid+j<n:
            if A1[i]<A2[j]:
                A[i+j]=A1[i]
                i+=1
            else:
                A[i+j]=A2[j]
                j+=1
        while i<mid:
            A[i+j]=A1[i]
            i+=1
        while mid+j<n:
            A[i+j]=A2[j]
            j+=1
```

Analyzing the MergeSort Algorithm



O(n) operations at each level

We have lg n levels therefore,

O(n lg n)

depth	# instances	Size of instances	# read/write operations
0	1	n	n → n
1	2	n/2	2 * n/2 → n
2	4	n/4	4 * n/4 →n
…	…	…	
k	2^k	$n/2^k$	2^k * $n/2^k$ → n
…	…	…	
lg n	$2^{\lg n}$ → n	$n/2^{\lg n}$ → 1	$2^{\lg n}$ *1 → n

Clicker Question

- Is MergeSort stable?
- A: Yes
- B: No
- C: I don’t know

Clicker Question

- Is MergeSort stable?
- A: Yes
- B: No
- C: I don't know

prefer the "left" of the two sorted sublists on ties

When is the MergeSort algorithm used?

- External sorting is a term for a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually RAM) and instead they must reside in the slower external memory (usually a hard drive).
- MergeSort is suitable for external sorting.

Name	Best	Average	Worst	Stability	Memory
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	challenging	O(1)
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	Yes	O(1)
MergeSort	O(n lg n)	O(n lg n)	O(n lg n)	Yes	O(n)

Introduction

Selection Sort

Insertion Sort

MergeSort

QuickSort

QuickSort

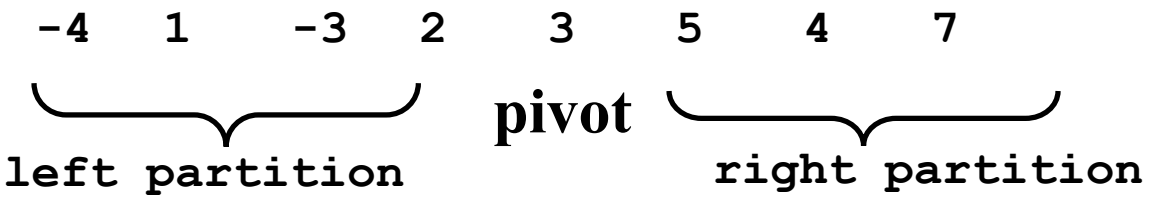
- In practice, one of the fastest sorting algorithms is Quicksort, developed in 1961 by C.A.R. Hoare.
- Comparison-based: examines elements by comparing them to other elements
- Divide-and-conquer: divides into “halves” (that may be very unequal) and recursively sorts

QuickSort algorithm

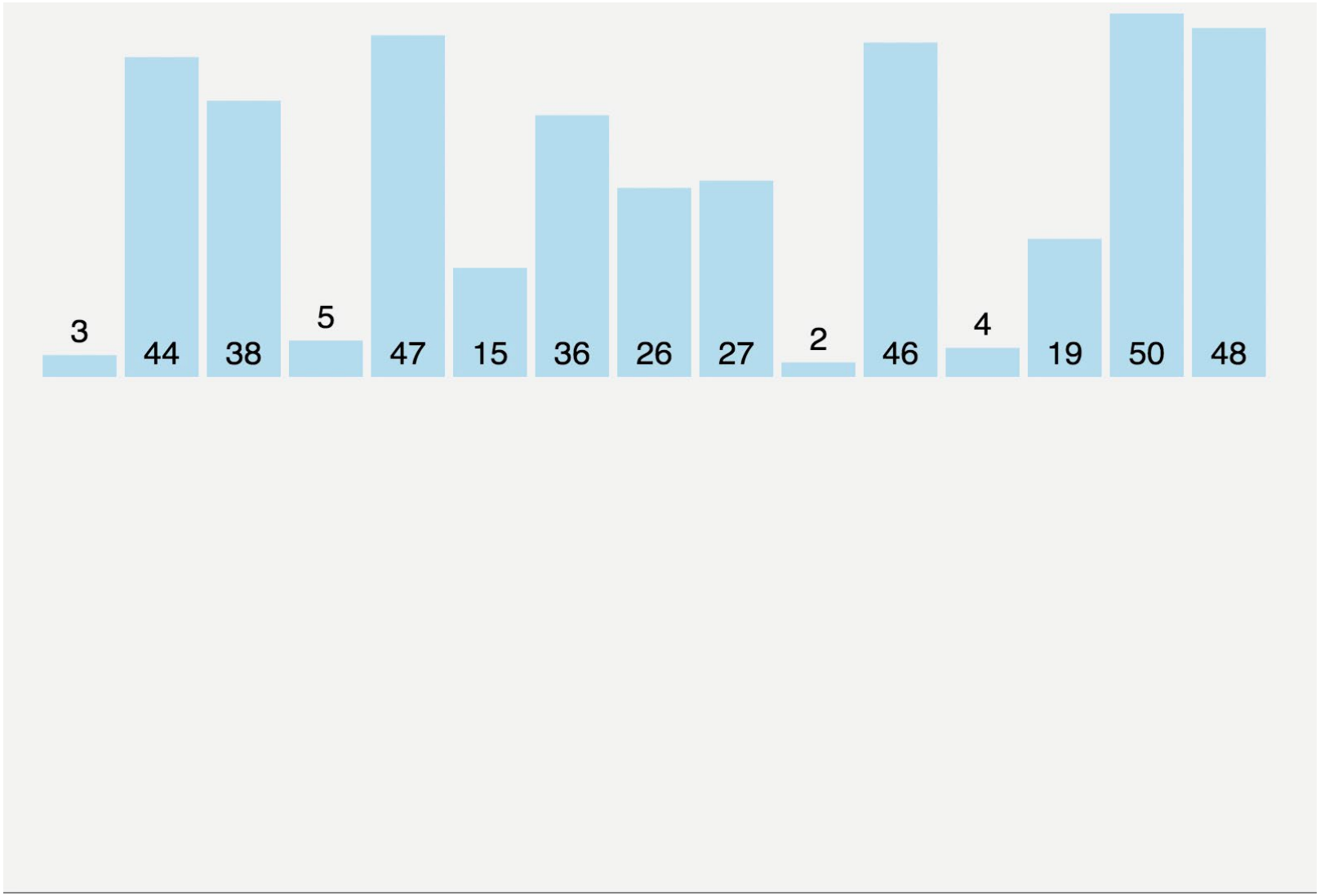
- Pick a pivot
- Reorder the list such that all elements < pivot are on the left, while all elements >= pivot are on the right
- Recursively sort each side

Partitioning

- The act of splitting up an array according to the pivot is called partitioning
- Consider the following:

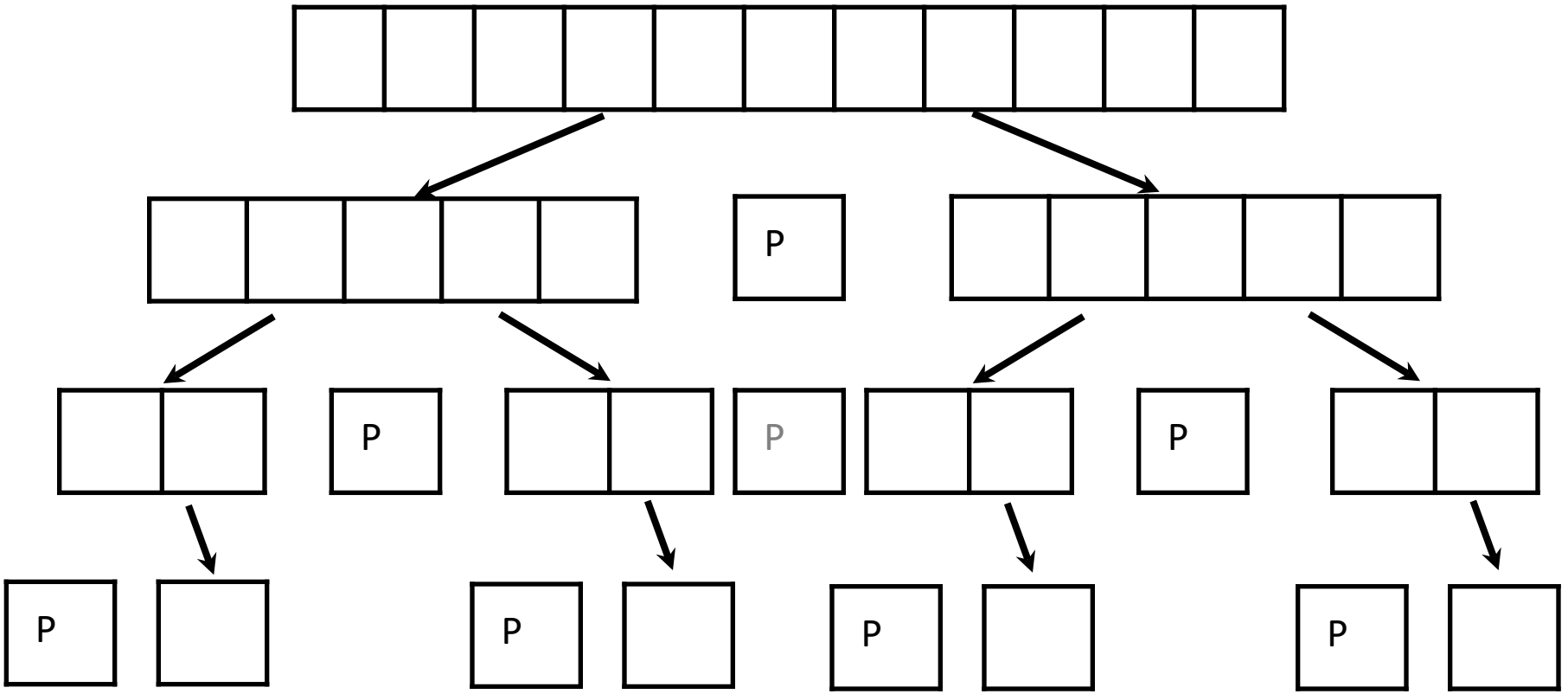


Algorithm Visualisation



Choose Quick sort on https://visualgo.net/en/sorting

QuickSort Visually



Sorted!

QuickSort Example

```
2  -4  6  1  5  -3  3  7
```

Clicker question

- Here is an array which has just been partitioned by the first step of QuickSort:

  3, 0, 2, 4, 5, 8, 7, 6, 9

Which of these elements could be the pivot?

- a. 3
- b. 4
- c. 5
- d. 6
- e. (b) or (c)

Clicker question (answer)

- Here is an array which has just been partitioned by the first step of QuickSort:

  3, 0, 2, 4, 5, 8, 7, 6, 9

Which of these elements could be the pivot?

- a. 3
- b. 4
- c. 5
- d. 6
- e. (b) or (c)

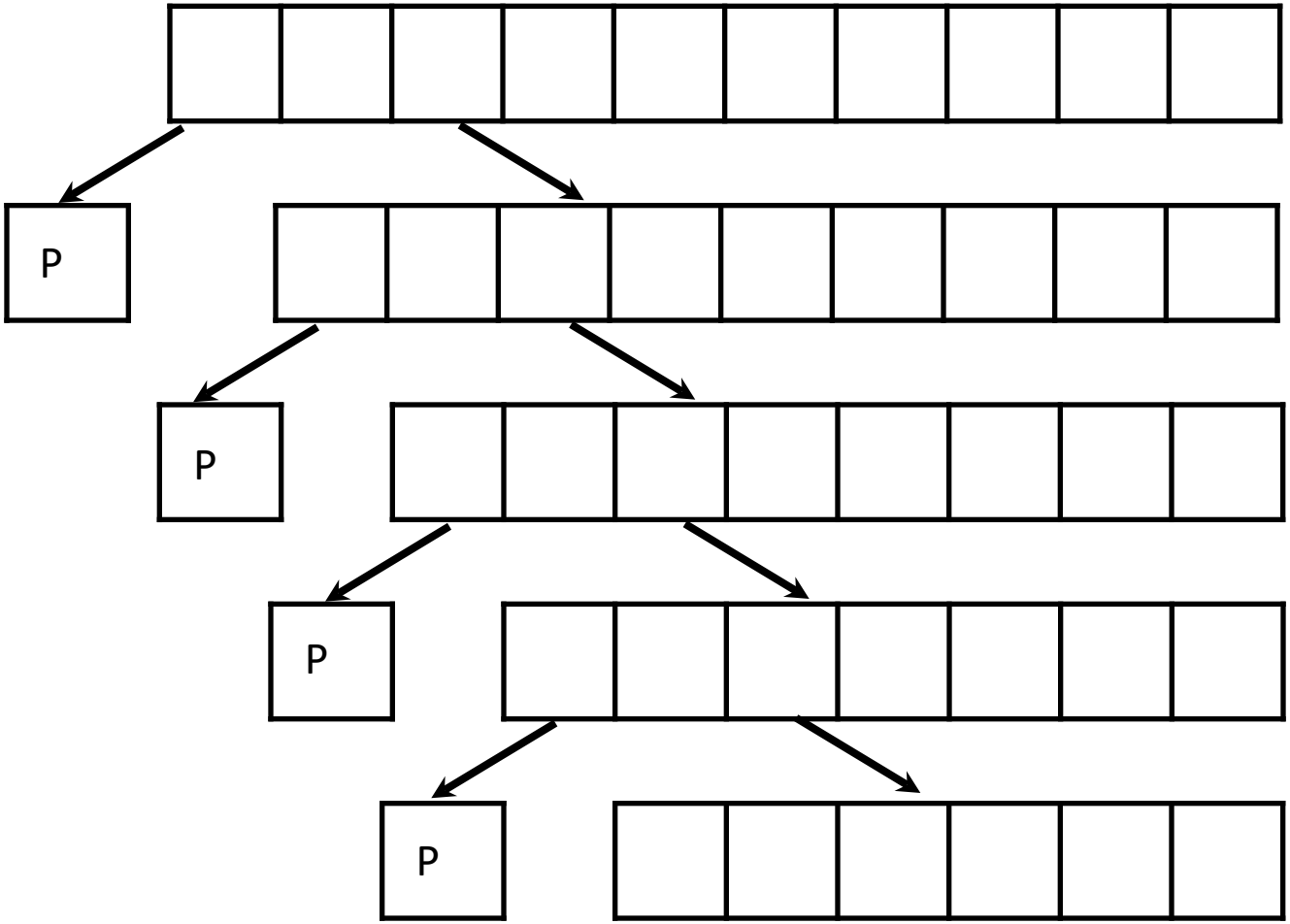
Quick Sort

```
# Time Complexity of Solution:
# Best = Average = O(nlog(n)); Worst = O(n^2).
#
# Approach: divides into “halves” (that may be very unequal) and
# recursively sorts

def partition(A, p, r):
        x = A[r]
        i = p-1
        for j in range(p,r):
                if A[j] <= x:
                        i+=1
                        swap(A, i, j)
        swap(A, i+1, r)
        return i+1

def quick_sort(A, p=0, r=None):
        if r==None:
                r=len(A)-1
        if p<r:
                q = partition(A, p, r)
                quick_sort(A, p, q-1)
                quick_sort(A, q+1, r)
```

QuickSort: Complexity

- In our partitioning task, we compared each element to the pivot
  - Thus, the total number of comparisons is N
  - As with MergeSort, if one of the partitions is about half (or *any* constant fraction of) the size of the array, complexity is $\Theta(n \lg n)$.
- In the worst case, however, we end up with a partition with a 1 and n-1 split

QuickSort Visually: Worst case



QuickSort: Worst Case

- In the overall worst-case, this happens at every step…
  - Thus we have N comparisons in the first step
  - N-1 comparisons in the second step
  - N-2 comparisons in the third step
  - :

$$n + (n-1) + \cdots + 2 + 1 = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$$

…or $O(n^2)$

MergeSort vs. QuickSort

- QuickSort in practice tends to run faster than MergeSort, but its worst-case complexity is $O(n^2)$.

- That worst-case behaviour can usually be avoided by using more clever ways of finding the pivot (not just using the first element).
  - Randomized algorithms can be used to prove that the average case for Quicksort is *O(n lg n)*

QuickSort: Average Case (Intuition)

- Clearly pivot choice is important
  - It has a direct impact on the performance of the sort
  - Hence, QuickSort is fragile, or at least "attackable"

- So how do we pick a good pivot?

QuickSort: Average Case (Intuition)

- Let's assume that pivot choice is random
  - Half the time the pivot will be in the center half of the array. Thus at worst the split will be n/4 and 3n/4

- We can apply this to the notion of a good split
  - Every "good" split: 2 partitions of size n/4 and 3n/4
    - Or divides N by 4/3

- This can be used to prove that average case runs in O(n log n)

Comparison of different sorting algorithms

- Quicksort algorithm is one of the best sorting algorithms and is widely used

Name	Best	Average	Worst	Stability	Memory
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	challenging	O(1)
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	Yes	O(1)
MergeSort	O(n lg n)	O(n lg n)	O(n lg n)	Yes	O(n)
QuickSort	O(n lg n)	O(n lg n)	$O(n^2)$	Challenging	O(lg n)

Learning Objective

Description	Tag
Describe and apply selection sort.	selection-sort
Describe and apply insertion sort.	insertion-sort
Describe and apply merge sort.	merge-sort
Describe and apply quicksort.	quicksort
Compare and contrast the trade-offs of different sorting algorithms.	compare-sorts
Explain what is meant by a stability in sorting algorithms and determine which sorting algorithms are stable.	stability-sorting
State differences in performance for large files versus small files on various sorting algorithms.	sort-large-small
Analyse the complexity of different sorting algorithms.	sorting-analysis
Manipulate data using various sorting algorithms (irrespective of any implementation).	sort-data-manipulation
Reflect on the importance of sorting in DBMS	sort-dbms