# Algorithmics

## Divide and Conquer

**Vicente García Díaz – garciavicente@uniovi.es**
*University of Oviedo, 2016*

# Table of Contents

*Divide and Conquer*

# Basic concepts

# Process to obtain solutions

1. Decompose a problem into `n` problems smaller than the original

2. Solve each of the subproblems:
   - General case → recursively
   - Base case → directly

3. Combine the solutions to obtain the solution to the original problem

# What do we need?

1. Find a recursive scheme that will reduce the original problem to the base case

2. Have a simple algorithm, capable of solving the base cases, which is efficient in small cases

3. Provide a method to combine the results of the subproblems

# Pseudocode

```
SolutionType divideAndConquer(int n) {
  ProblemType[] subproblems;
  SolutionType[] subsolutions;

  if (n is sufficiently small)
    return Solve trivial case;
  else {
      subproblems = decompose(n);
      for (int i=0; i< subproblems.length; i++)
        subsolutions[i] = divideAndConquer(newSize);

      return combine(subsolutions);
  }
}
```

# Worth noting...

- The number of `a` subproblems must be small
  - If `a = 1` → The process is called reduction
- The recursive design is more clear and elegant
  - You can do the same using an iterative loop (especially with reduction)

# Divide and conquer by division

- Parameters
  - `a` → number of subproblems
  - `b` → all the subproblems have a size (`n / b`), being `b` a constant and `n` the size of the original problem
  - `k` → assumes that the complexity of the overall scheme excluding recursive calls, i.e. considering only the operations of decomposition and composition is the polynomial type: **$O(n^k)$**

# Division scheme analysis

- Execution time
  - $T(n) = a * T(n/b) + cn^K$ if n > basic case
  - $T(n) = c * n^k$ if n = basic case

- Complexity
  - **$O(n^k)$ if $a < b^k$**
  - **$O(n^k * \log n)$ if $a = b^k$**
  - **$O(n^{\log_b a})$ if $a > b^k$**

**values for Quicksort?**

# Divide and conquer by subtraction

- Parameters
  - `a` → number of subproblems
  - `b` → all the subproblems have a size (`n - b`), being `b` a constant and `n` the size of the original problem
  - `k` → assumes that the complexity of the overall scheme excluding recursive calls, i.e. considering only the operations of decomposition and composition is the polynomial type: **$O(n^k)$**

# Subtraction scheme analysis

- Execution time
  - $T(n) = a * T(n-b) + cn^K$ if $n >$ basic case
  - $T(n) = c * n^k$ if $n =$ basic case

- Complexity
  - **$O(n^k)$** **if $a < 1$ (never happens)**
  - **$O(n^{k+1})$** **if $a = 1$**
  - **$O(a^{n \text{ div } b})$** **if $a > 1$**

**values for Quicksort?**

# Examples of use

# Factorial of a number

$n!$

- Goal
  - Calculate the factorial of a number

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n\,(n-1)! & \text{si } n \neq 0 \end{cases}$$

- Analysis
  - It is divide and conquer by subtraction
    - `a = 1` → number of subproblems
    - `b = 1` → size of each subproblem
    - `k = 0` → decomposition into subproblems costs O(1) → $O(n^0)$
  - a == 1
    - Complexity → $O(n^{k+1})$ → $O(n^1)$ → **O(n)**

`fact2()?`

# Fibonacci series

- Goal
  - Calculate the Fibonacci function (0,1,1,2,3,5,8,13,21,34,55,89,...)

$$f = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f(n-1) + f(n-2) & \text{if } n>1 \end{cases}$$

- Scheme
  - Looks like a scheme by subtraction
    - `a = 2` → number of subproblems
    - `k = 0` → decomposition into subproblems costs O(1) → $O(n^0)$
    - `b = ???` → It is different in the two subproblems!

# Analysis

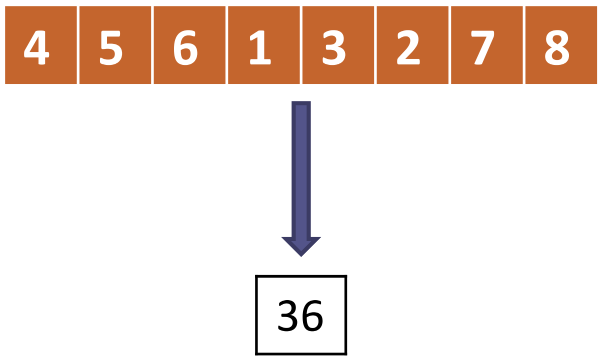
- It is divide and conquer by subtraction
  - If the recursive part would be f = f(n-1) + f(n-1)
    - a = 2, b = 1, k = 0
    - a > 1 ----- (2 > 1)
    - Complexity → $O(a^{n \text{ div } b})$ → $O(2^n)$
  - If the recursive part would be f = f(n-2) + f(n-2)
    - a = 2, b = 2, k = 0
    - a > 1 ----- (2 > 1)
    - Complexity → $O(a^{n \text{ div } b})$ → $O(2^{n \text{ div } 2})$
  - We can conclude:
    - **$O(2^{n \text{ div } 2})$ <= O(Fibonacci) <= $O(2^n)$**
  - More accurately complexity → We should solve the recurrence equation

`fib4()?`

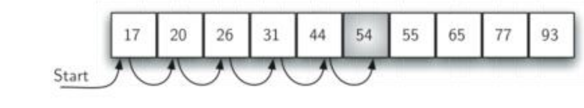
# Sum of elements

$\sum$

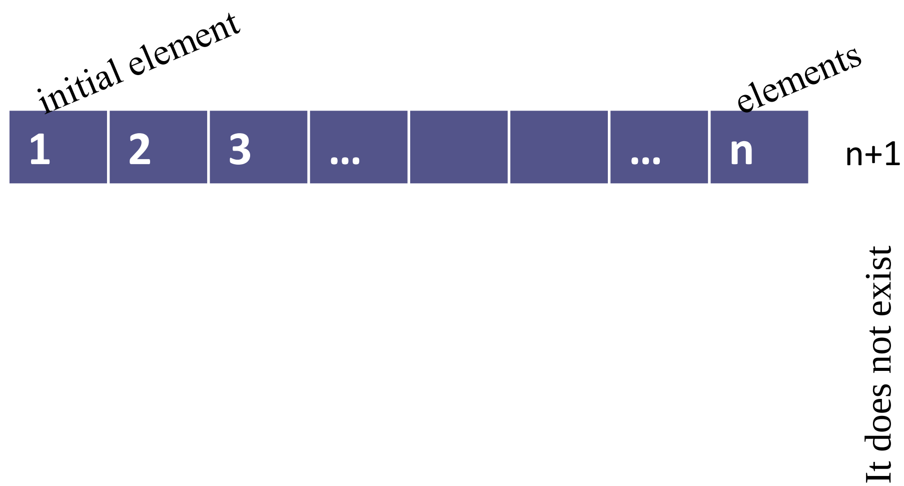
- The idea is to sum all the elements of a vector



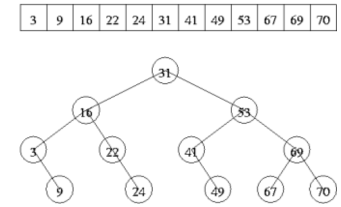
# Sequential search




- The idea is to sequentially find an element (the position) in a vector

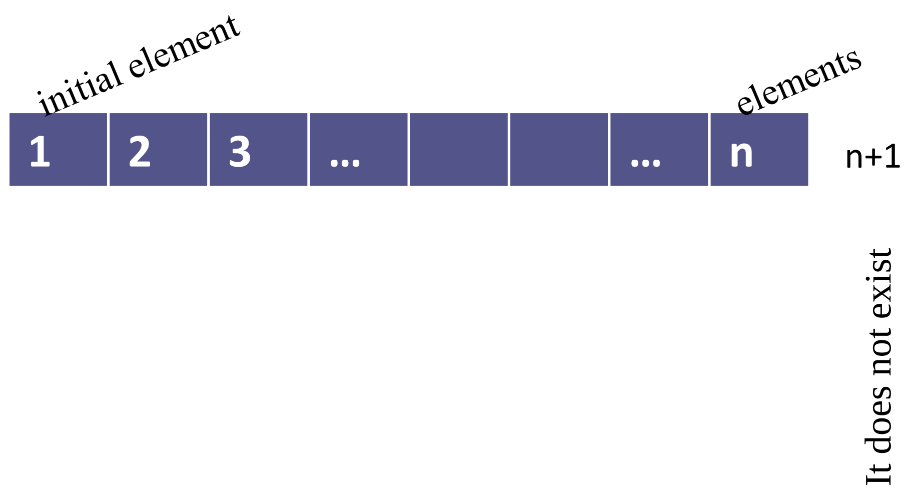



**sequentialSearch2()?**

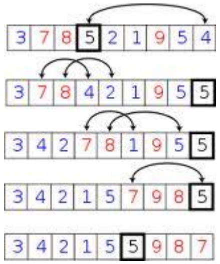



# Binary search

- The idea is to find an element (the position) in a vector using a binary search
  - The list should be sorted beforehand
  - We divide the list into two parts in each iteration




**binarySearch2()?**



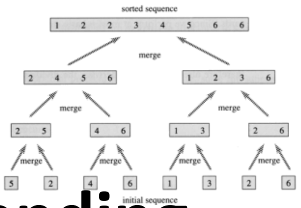
# The Quicksort algorithm

- Idea of the algorithm

REPEAT UNTIL ALL THE ELEMENTS ARE SORTED → O(log n) …O(n)

CHOOSE A PIVOT → Using median-of-3 is O(1) First part

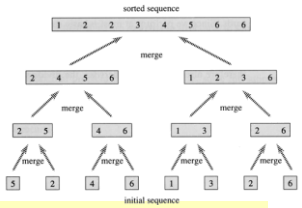
PARTITIONING THE PIVOT THROUGH A PARTITIONING STRATEGY → Typical case O(n) Second part




# Goal

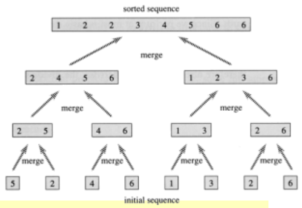
- **The idea is to sort a collection of integers in ascending order**
- Divide the array into to halves (we will take the middle of the collection)
- Recursively sort each half
- Merge two halves to make a sorted whole
  - To combine two halves, we will start at each collection at the beginning, picking the object which is smaller and inserting it into the new collection

# Pseudocode (I)

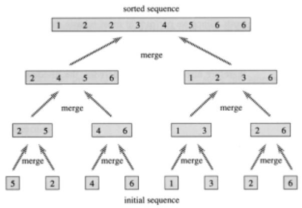



```
void mergesort(int left, int right, int[] elements) {
  if (right > left){
      //Get the index of the element in the middle
      int center = (right + left) / 2;
      //Sort the left side of the array
      mergesort(left, center);
      //Sort the right side of the array
      mergesort(center+1, right);
      //Combine both parts
      combine(left, center, center+1, right, elements);
  }
}
```

# Pseudocode (II)

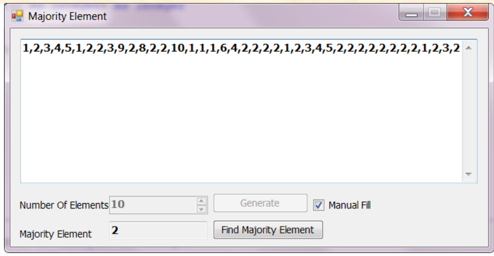



```
void combine(int x1, int x2, int y1, int y2, int[]
  elements) {
  int sizeX = x2-x1+1;
  int sizeY = y2-y1+1;
  //Copy the elements from left to center into a helper
  for (int i = 0; i < sizeX; i++){
      x[i] = elements[x1+i];
  }
  //Copy the elements from center+1 to right into a helper
  for (int i = 0; i < sizeY; i++){
      y[i] = elements[y1+i];
  }

  //Copy the smallest elements from either the left or the
  right side to the elements collection
  …

  //Copy the rest of the elements into the collection
  …
}
```




# Analysis

- It is divide and conquer by division
  - a = 2 → Number of subproblems
  - b = 2 → Size of each subproblem (n/2)
  - k = 1 → Decomposition into subproblems costs **$O(n^1)$**
  - $a = b^k$ ----- ($2 = 2^1$)
  - Complexity → $O(n^k * \log n)$ → **$O(n * \log n)$**




# The majoritarian element

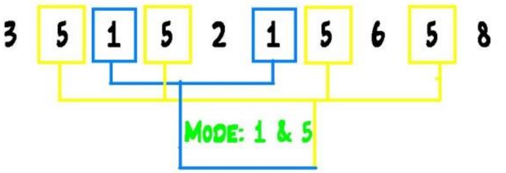
- Is there a majoritarian element in `n` elements?
  - To be the majoritarian element, it should be at least `n/2+1` times

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 5 | 5 | 1 | 5 | 2 | 5 | 5 |

$\boxed{5}$

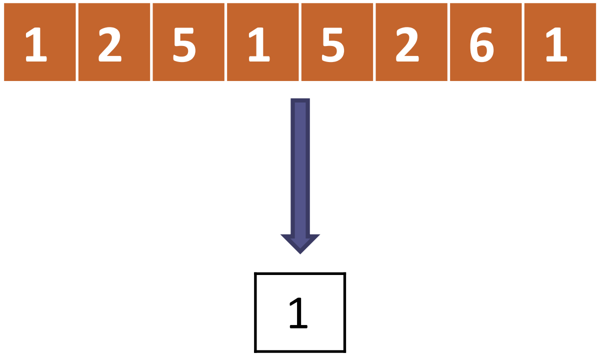
**majoritarian2()?**

Mode of a set of numbers



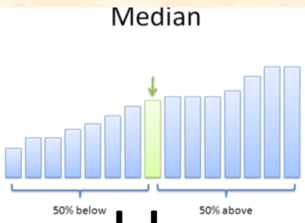

# Mode of a set of numbers

- The mode is the element that is repeated more times
- That is, the predominant element




`mode2()?`

# Median of a set of numbers




- The median of a finite list of numbers can be found by arranging all the observations from lowest value to highest value and picking the middle one

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

↓

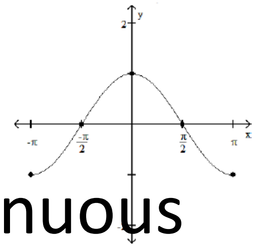
| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

↓

$\boxed{4}$

**median1()?**

# Maximum sum of subsequences

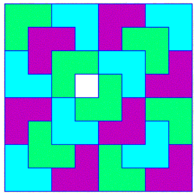


- We need to find the maximum sum of all the continuous subsequences of `n` elements

| | | | | | |
|---|---|---|---|---|---|
| 5 | -4 | 3 | 2 | 5 | -1 |

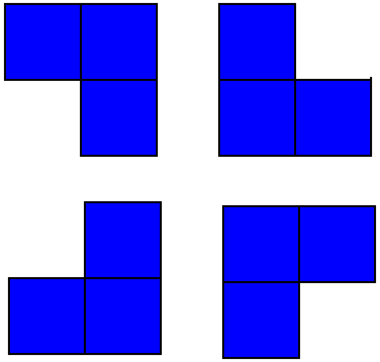
11

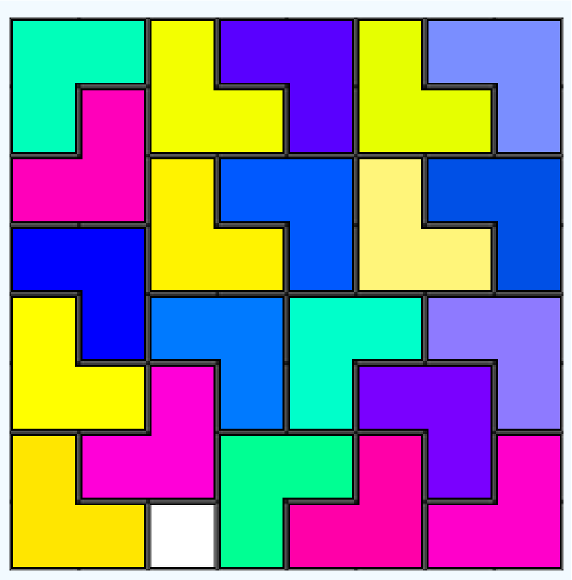
**`maxSum2()?`**



# The Tromino puzzle

- A Tromino is a geometric figure formed by three squares of size 1x1 L-shaped





- We have a board of size `nxn`
- The goal is to cover all board positions with Trominoes
- …except one position that will be an empty (or black) cell
  - http://www3.amherst.edu/~nstarr/trom/puzzle-8by8/

# *Bibliography*


JUAN RAMÓN PÉREZ PÉREZ; (2008) ***Introducción al diseño y análisis de algoritmos en Java***. Issue 50. ISBN: 8469105957, 9788469105955 (Spanish)