Unit 2

The Divide-and-Conquer Strategy (D&C)

1. **The D&C approach**: Why (reasons)? What (definition and general scheme)? When is it appropriate to use it ("cost cookbook")?
2. **D&C solutions to Fast-Sorting and Fast-Selection**: **Which are the best D&C solutions to the Sorting problem?** What about the Selection problem?
   - **Lab. 2** Effciency's empirical analysis of two sorting D&C algorithms
1. **Reduce-and-Conquer Exercises** : Why not D&C exercises?

Bibliography

- Weiss, M.A. *Data Structures and Problem Solving Using Java, 4th Edition.* Adisson-Wesley, 2010. Chapter 8, sections 5, 6 and 7
- Galiano I. and Prieto N. *Notes from the "Estructuras de Datos y Algoritmos" course*. Previous Computer Science Curriculum. Available in PoliformaT
  - Apuntes - Diseño recursivo y eficiente: soluciones Divide y Vencerás para la Ordenacioón y la Selección

1. The D&C approach

Why (reasons)? Recursion: the bad, the good and the fair

Recursion is a powerful problem-solving approach that yields compact but readable, elegant and very efficient algorithms

The Sorting Problem:

Recursion ($O(x \log x)$) **VS** Iteration ($O(x^2)$)

Divide and Conquer

Too much recursion can be dangerous!

The Fibonacci numbers:
Recursion ($O(2^x)$) **VS** Iteration ($O(x)$)

MORAL

Use recursion ONLY to solve problems complex enough to deserve it!

1. The D&C approach

What? *Definition*

The D&C Strategy involves three steps at each level of the recursion

- DIVIDE the problem of size **x** into a number of subproblems **a** that are smaller instances of the same problem (**a > 1**)

  WARNING: at least two **DISJOINT** subproblems

  TIPS: The size of the subproblems **should** reduce the size of the original problem …

  - **GEOMETRICALLY**, or by the same constant factor **c**: **x / c**
  - In the most **BALANCED** way possible: **a = c**

- CONQUER the subproblems by solving them recursively, **except, of course, the base cases**
- COMBINE the solutions to the subproblems into the solution for the original problem

1. The D&C approach

What? *General scheme and its Recurrence equation*

```
public static ResultType conquer(DataType x) {
    ResultType method_res, call_1_res, ..., call_a _res;
    if (x == x_base) { method_res = baseCaseSolution(x); }
    else {
        int c = divide(x);
        call_1_res = conquer(x / c);
        ...
        call_a _res = conquer(x / c);

        method_res = combine(x, call_1_res, ..., call_a _res);
    }
    return method_res;
}
```

But... Can a Strategy Recurrence Equation be solved? How?

Recurrence Equation for the general (recursive) case

$$T_{conquer}(x > x_{base}) = \underset{\text{Number of recursive calls}}{a} * T_{conquer}(x \underset{\text{x decreases geometrically}}{/} c) + \underbrace{T_{divide}(x) + T_{combine}(x)}_{\text{Call overhead}}$$

1. Divide & Conquer

When is it appropriate to use it ? "Cost Cookbook"

Theorem 1: $T_{recursiveMethod}(x) = a \cdot T_{recursiveMethod}(x-c) + b, \ \ b \geq 1$

- If a=1, $T_{recursiveMethod}(x) \in \Theta(x)$
- If a>1, $T_{recursiveMethod}(x) \in \Theta(a^{x/c})$

Theorem 2: $T_{recursiveMethod}(x) = a \cdot T_{recursiveMethod}(x-c) + b \cdot x + d$, b and d≥1

- If a=1, $T_{recursiveMethod}(x) \in \Theta(x^2)$
- If a>1, $T_{recursiveMethod}(x) \in \Theta(a^{x/c})$

Theorem 3: $T_{recursiveMethod}(x) = a \cdot T_{recursiveMethod}(x/c) + b, \ b \geq 1$

- If a=1, $T_{recursiveMethod}(x) \in \Theta(\log_c x)$ **Reduce & Conquer** (Binary Search)
- If a>1, $T_{recursiveMethod}(x) \in \Theta(x^{\log_c a})$

Theorem 4: $T_{recursiveMethod}(x) = a \cdot T_{recursiveMethod}(x/c) + b \cdot x + d$, b and d≥1

- If a<c, $T_{recursiveMethod}(x) \in \Theta(x)$
- If a=c, $T_{recursiveMethod}(x) \in \Theta(x \cdot \log_c x)$ **Divide & Conquer** (Fast Sorting)
- If a>c, $T_{recursiveMethod}(x) \in \Theta(x^{\log_c a})$

$$T_{conquer}(x > x_{base}) = a * T_{conquer}(x / c) \ + \ T_{divide}(x) \ + \ T_{combine}(x)$$

1. Divide & Conquer

How to use the "Cost Cookbook"? Examples

To review what you learned last year about the analysis of recursive methods and, at the same time, to know how to use the "cost cookbook", the slide-show at left shows some examples

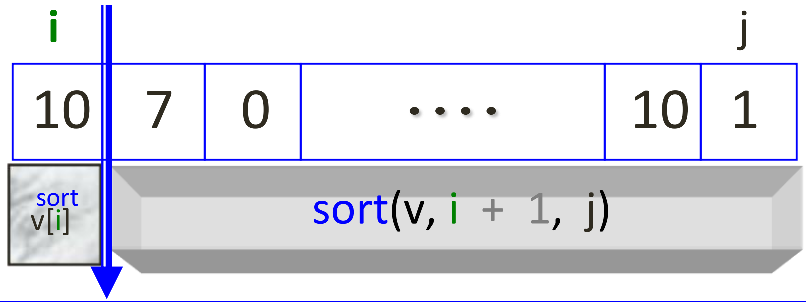
2. D&C solutions to Fast-Sorting

Two recursive approaches to Sorting

Restriction: sorting one element out of the total takes linear time in both the worst and the average cases

"Conservative" approach

sort(v, i , j) =

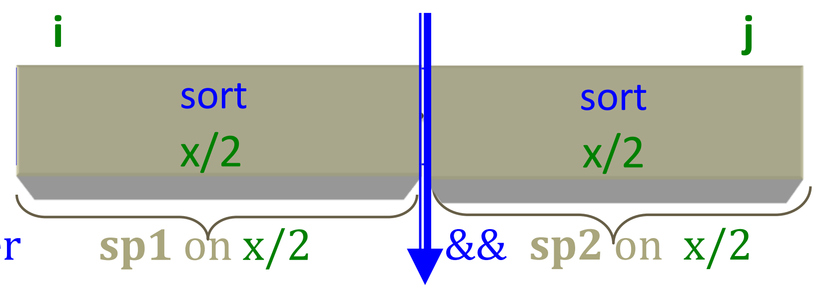


Average cost: by **T2** (Linear Overhead) with c = a = 1, $T^{\mu}_{sort}(x) \in \Theta(x^2)$

D&C approach?

IFF $T_{Divide}(x) + T_{Combine}(x) = k \cdot x$ **&&** sp1 size ≈ sp2 size (c = a)

Divide "properly" the original problem of size x (sort on x)



Conquer **sp1** on x/2 && **sp2** on x/2

Combine "properly" the sp1 & sp2 solutions into the solution to sort on x

Average cost: by **T4** (Linear Overhead) with c = a (= 2), $T^{\mu}_{sort}(x) \in \Theta(x \cdot \log x)$

2. D&C solutions to Fast-Sorting

Merge Sort: D&C approach and a-priori cost

IFF $T_{Divide}(x) + T_{Combine}(x) = k \cdot x$ **&&** sp1 size ≈ sp2 size (c = a)

By **T4** (Linear Overhead) with $c = a = 2$, $T^{\mu}_{sort}(x) \in \Theta(x \cdot \log x)$

Divide "properly" the original problem of size x

m = (i + j) / 2; $T_{Divide}(x) \in \Theta(1)$

i **j**

mergeSort	**merge**Sort
v[i, m]	v[m + 1, j]

Conquer sp1 on x/2 Conquer sp2 on x/2

v[i, m] y v[m+1, j] Sorted

Combine "properly" the sp1 & sp2 solutions into the solution for sort on x

Sort v[i, j] by ***merging*** the already sorted v[i, m] and v[m+1, j]: $T_{Combine}(x) \in \Theta(x)$

```java
public static <T extends Comparable<T>> void merge(T[] v, int i, int f, int m) {
    int a = i, b = m + 1, k = 0; T[] aux = (T[]) new Comparable[f - i + 1];
     while (a <= m && b <= f) {
         if (v[a].compareTo(v[b]) < 0) { aux[k++] = v[a++]; }
         else                          { aux[k++] = v[b++]; }
     }
     while (a <= m) { aux[k++] = v[a++]; }
     while (b <= f) { aux[k++] = v[b++]; }
     for (a = i, k = 0; a <= f; a++, k++) { v[a] = aux[k]; }
}
```

2. D&C solutions to Fast-Sorting

Merge Sort: D&C approach and a-priori cost

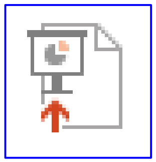
```
private static <T extends Comparable <T>> void mergeSort(T[] v, int i, int j) {
    if (i < j) {
        int m = (i + j) / 2;            // DIVIDE
        mergeSort(v, i, m);             // CONQUER
        mergeSort(v, m + 1, j);         // CONQUER
        merge(v, i, j, m);              // COMBINE
    }
}

public static <T extends Comparable<T>> void mergeSort(T v[]) {
    mergeSort(v, 0, v.length - 1);
}
```

As expected, $T_{mergeSort}(x) \in \Theta(x \cdot \log x)$ by T4:

$$T_{conquer}(x > 1) = \underset{\substack{\downarrow \\ a\,=\,2}}{a} * T_{conquer}(x/\underset{\substack{\downarrow \\ c\,=\,2}}{c}) + \underbrace{T_{divide}(x) + T_{combine}(x)}_{\Theta(x)}$$

2. D&C solutions to Fast-Sorting

How does Merge Sort works? *Unrolling the recursion*



To understand `mergeSort` method, it is worthwhile to consider carefully the dynamics of the method calls. The slide-show at left shows this dynamics "at the click of a mouse" -by tracing the call to its driver method with the array v = {5, 2, 4, 6, 1, 3, 8, 7} as argument