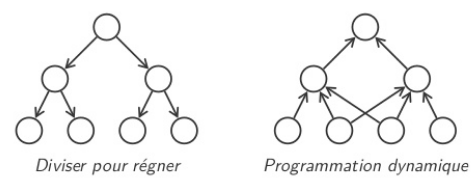
Université IBN KHALDOUN –TIARET- Faculté de Mathématiques et Informatique

Département d'informatique, année universitaire 2019/2020.

**Cours**: Structures de données avancées et algorithme

**Travaux pratiques: complexité algorithmique**

* La programmation dynamique s'appuie sur le principe d'optimalité de Bellman : une solution optimale d'un problème s'obtient en combinant des solutions optimales à des sous-problèmes.
* La méthode de programmation dynamique, comme la méthode diviser pour régner, résout des problèmes en combinant des solutions de sous-problèmes. Les algorithmes diviser-pour-régner partitionnent le problème en sous-problèmes indépendants qu’ils résolvent récursivement, puis combinent leurs solutions pour résoudre le problème initial. La méthode diviser-pour-régner est inefficace si on doit résoudre plusieurs fois le même sous-problème. Par exemple, l'algorithme suivant est inefficace :



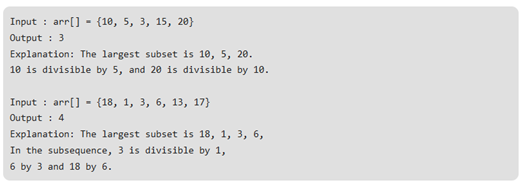
**Concevoir une procédure de programmation dynamique en quatre étapes :**

1. Définir les sous-problèmes.
2. Identifier une relation de récurrence entre les solutions des sous-problèmes.
3. En déduire un algorithme récursif avec mémorisation ou une approche du bas vers le haut
4. Résoudre le problème original à partir des solutions des sous-problèmes

**Problème 1 : Plus grand sous-ensemble de paires divisibles ou Largest divisible pairs subset**

Étant donné un tableau de n éléments distincts, trouvez la longueur du plus grand sous-ensemble tel que chaque paire du sous-ensemble soit telle que l'élément le plus grand de la paire soit divisible par un élément plus petit.

**Exemple :**



**Complexité temporelle :** O(n\*n)

**The problem:**

Finding the nth **Fibonacci number**is class dynamic programming problem. Not only can dynamic programming (DP) gracefully find a solution to the problem, but the DP state table is very intuitive to understand.

The Fibonacci sequence are numbers in the following integer series:  
**0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144…**where the n-th number is the sum of the n-1-th and n-2-th number. That is

F(n) = F(n-1) + F(n-2) where F(0) =0 and F(1) =1.

# Recursive Solution First :

The **recursive solution** involves a **top-down** approach where we recursively break down the main problem into smaller and smaller sub-problems until we reach the base cases. Then starting from the base-case, we climb back down the call stack doing some computation at each recursive call.

public int findFibonacci(int n) {  
 if (n == 0) {  
 return 0;  
 }else if (n == 1) {  
 return 1;  
 }  
   
 return findFibonacci(n-2) + findFibonacci(n-1);  
}



**Comparaison des performances de plusieurs algorithmes de tri écrit en C :**

|  |
| --- |
|  |
|  | - tri par sélection |
|  | - tri par sélection récursif |
|  | - tri par insertion |
|  | - tri fusion |
|  | - tri bulle |
|  | - tri rapide |



Merge sort in 3 minutes

