# Chapitre 3

## L’approche force brute

# Caractéristiques de l'approche force brute

- La force brute fait référence à l'ordinateur et non au cerveau du concepteur car ce dernier fait un minimum d'effort intellectuel (alors que l'on exploite au maximum la « force » de l'ordinateur)
- En effet, l'approche force brute s'appuie directement sur l'énoncé du problème et la définition des concepts associés au problème
  - Exemple: l'algorithme pgcdFB(m,n) utilise simplement la définition du pgcd(m,n). On essaie d'abord avec n ($\leq$ m par hypothèse) et si ça ne marche pas, on essaie ensuite avec n - 1, et n - 2 … jusqu'à ce que l'on trouve un entier divisant m et n
- Le principal avantage de l'approche force brute est qu'il nous permet de trouver rapidement un algorithme correct (qui solutionne notre problème en un temps fini)
  - Or, il arrive souvent que cet algorithme est inefficace et même très inefficace (comme le pgcdFB(m,n))
  - Cependant, cela peut nous satisfaire si nous allons seulement utiliser l'algorithme sur un faible nombre d'instances; chacune ayant une taille relativement faible

# Caractéristiques de l'approche force brute (suite)

- Il arrive parfois que la performance d'un algorithme force brute soit acceptable (pour le peu d'effort intellectuel qu'on y a investi) pour des instances de taille relativement grande.
- Considérez l'algorithme du tri à bulles pour le tri d'un tableau A[0..n-1]
  - Cet algorithme des paires d'éléments adjacent A[j] et A[j + 1] et les permute si les éléments ne sont pas dans l'ordre croissant.
  - Après une passe sur le vecteur, on a la certitude que le plus grand élément a remonté jusqu'à la dernière position.
  - Après une i ème passe sur le vecteur, le i ème plus grand élément a remonté jusqu'à sa position finale.

# Analyse du tri à bulles

**ALGORITHME** BubbleSort(A[0..n-1])
// Entrée: le tableau A
// Sortie: le tableau A trié

**for** i ← 0 **to** n-2 **do**
  **for** j ← 0 **to** n-2-i **do**
    **if** A[j] > A[j + 1]
      swap A[j] and A[j + 1]

- Choisissons A[j] > A[j + 1] pour l'opération de base
- Le nombre de fois que cette opération est effectuée dépend uniquement de n
- Alors: $C_{worst}(n) = C_{best}(n) = C(n)$
  - Le temps d'exécution du meilleur cas est identique à celui du pire cas

# Analyse du tri à bulles (suite)

- La valeur de C(n) est alors obtenue de la sommation suivante:

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$$

$$\text{car} \quad \lim_{n\to\infty} \frac{n(n-1)}{2n^2} = \lim_{n\to\infty} \left( \frac{1}{2} - \frac{1}{2n} \right) = \frac{1}{2}$$

- Cet algorithme n'est donc pas si « mauvais » mais il en existe de meilleurs…
- Au lieu de présenter une série d'algorithmes force brute (donc ennuyants par définition), passons directement aux techniques de conception générant des algorithmes excitants et efficaces!
  - Nous comparerons ces algorithmes excitants aux algorithmes ennuyants pour démontrer ce qu'un brin de réflexion peut accomplir.

# Lecture (Levitin)

- Chapitre 3
  - 3.1 Selection Sort and Bubble Sort
    - (Le Selection Sort n’est pas matière à examen)