Définitions

- La complexité d'un algorithme est le comportement de celui-ci pour de grandes valeurs de n (taille du problème).
  - Complexité spatiale : espace mémoire utilisé
  - Complexité temporelle : temps d'exécution
- Le temps d'exécution dépend :
  - de la plateforme (CPU)
  - du langage de programmation
  - du compilateur
  - du programmeur
  - de l'algorithme

Complexité temporelle

La complexité temporelle est indépendante des facteurs « extérieurs » :
- CPU
- langage
- compilateur
- programmeur

- La complexité temporelle peut être étudiée
  - dans le pire cas
  - dans le cas moyen
  - dans le meilleur cas

Opérations et énoncés

Tout langage de programmation possède des

1. opérations élémentaires
   a) arithmétiques : $+, -, \ldots$
   b) logiques : $\wedge, \vee, \ldots$
   c) de comparaison : $<, \leq, >, \ldots$
2. énoncés élémentaires
   a) d'affectation : $\longleftarrow$
   b) de branchement : appel de sous-programme, …
3. énoncés structurés
   a) tant que
   b) pour
   c) si

On suppose **en général** que les opérations et les énoncés élémentaires (1. et 2.) consomment chacun(e) 1 unité de temps peu importe la valeur des opérandes. Il faut cependant faire attention aux manipulations sur les « grands » nombres.

Notations

- $T(expression)$: temps pour évaluer l'expression
- $T(énoncés)$: temps pour évaluer les énoncés

- Exemples :
  - $T(x + 4) = 1$
  - $T(y \longleftarrow 2x) = 2$
  - $T(x \longleftarrow 5; x^3) = 3$
  - $T\big((y * x > 6) \wedge (x == 0)\big) = 4$

Trois méthodes de calcul de la complexité temporelle d'un algorithme

1. Calculer le *nombre exact* d'opérations élémentaires et en déduire un ordre de grandeur.
2. Définir une **opération barométrique**, calculer le *nombre exact* d'opérations barométriques et en déduire un ordre de grandeur.
3. Déterminer un ordre de grandeur pour différentes parties de l'algorithme et « combiner » ces ordres de grandeur pour obtenir l'ordre de grandeur total.

Énoncé **si**

si $< condition >$ **alors**
    $< se_1 >$
sinon
    $< se_2 >$
fin si

- Temps dans le pire cas :
  $$T(< condition >) + \max\{T(< se_1 >), T(< se_2 >)\}$$
- Temps dans le meilleur cas :
  $$T(< condition >) + \min\{T(< se_1 >), T(< se_2 >)\}$$
- Temps dans le cas moyen : La formule dépend de chaque problème et n'est pas nécessairement :
  $$T(< condition >) + \frac{1}{2}\big(T(< se_1 >) + T(< se_2 >)\big)$$

Exemple

Soit π une permutation de $[n], n \geq 2$.

si $\pi(1) == 1$ **alors**

$\quad x \leftarrow \pi(1); \pi(1) \leftarrow \pi(2); \pi(2) \leftarrow x;$

fin si

- Temps dans le pire cas :

$\quad 4$

- Temps dans le meilleur cas :

$\quad 1$

- Temps dans le cas moyen :

$$\frac{4}{n} + \frac{n-1}{n} = \frac{n+3}{n}$$

Énoncé **tant que**

tant que $< condition >$ **faire**

$\quad < se >$

fin tant que

- N_{max} : nombre maximal d'itérations
- N_{min} : nombre minimal d'itérations (pas nécessairement 0)
- N_{moy} : nombre moyen d'itérations

La condition est k fois vraie, 1 fois fausse, $k \geq 0$

On suppose que $T(< se >)$ est constant au cours de l'exécution.

- Temps dans le pire cas :

$$(N_{\text{max}} + 1)T(< condition >) + N_{\text{max}}\, T(< se >)$$

- Temps dans le meilleur cas :

$$(N_{\text{min}} + 1)T(< condition >) + N_{\text{min}}\, T(< se >)$$

- Temps dans le cas moyen :

$$(N_{\text{moy}} + 1)T(< condition >) + N_{\text{moy}}\, T(< se >)$$

Exemple

Soit π une permutation de $[n], n \geq 2$, telle que $\boldsymbol{\pi(1) \neq 1}$.

$j \leftarrow 2$

Tant que $< \pi(1) \neq 1 >$ **faire**

$\quad x \leftarrow \pi(1); \pi(1) \leftarrow \pi(j); \pi(j) \leftarrow x; j \leftarrow j + 1;$

fin tant que

- $N_{\text{max}} = n - 1$
- $N_{\text{min}} = 1$
- $N_{\text{moy}} = \sum_{i=1}^{n-1} \frac{i}{n-1} = \frac{n}{2}$

- Temps dans le pire cas : $1 + n + 4(n-1) = 5n - 3$
- Temps dans le meilleur cas : $1 + 2 + 4 = 7$
- Temps dans le cas moyen : $1 + \left(\frac{n}{2} + 1\right) + 4\frac{n}{2} = \frac{5n}{2} + 2$

Rappels de combinatoire

$$\sum_{i=m}^{n} (a_i + b_i) = \sum_{i=m}^{n} a_i + \sum_{i=m}^{n} b_i$$

$$\sum_{i=m}^{n} c a_i = c \sum_{i=m}^{n} a_i$$

$$\sum_{i=m}^{n} c = c(n - m + 1)$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$$

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1} \quad \text{si } x \in \mathbb{R}\backslash\{1\}$$

Énoncé **pour**

pour $variable = < \text{val. init.} >$ **à** $< \text{val. fin.} >$ **faire**

$\quad < se >$

fin pour

- N : nombre d'itérations $= < \text{val. fin.} > - < \text{val. init.} > + 1$
- À chaque itération :
  - mise à jour de la variable de contrôle
  - comparaison avec la valeur finale
- Temps dans tous les cas (si $T(< se >)$ ne dépend pas de la variable de contrôle) :

$$(N + 1)T(\text{m.à.j. et comparaison}) + NT(< se >)$$

Exemple

Soit π une permutation de $[n], n \geq 2$, telle que $\boldsymbol{\pi(1) \neq 1}$.

pour $j = 2$ **à** n **faire**

$\quad$ **si** $\pi(j) == 1$ **alors**

$\quad\quad \pi(j) \leftarrow \pi(1); \pi(1) \leftarrow 1;$

$\quad$ **fin si**

fin pour

- $N = n - 1$
- Temps dans tous les cas :

$$2n + (n - 2) + 3 = 3n + 1$$

Tri par sélection

Pour trier un tableau de taille n :

Pour i de n à 2 en descendant

on cherche le plus grand élément du tableau jusqu'à i et on l'échange avec le i^e élément (le dernier de la partie du tableau pas encore triée).

51	55	69	11	48	35	53	44
51	55	44	11	48	35	53	69

51	55	44	11	48	35	53	69
51	53	44	11	48	35	55	69

51	53	44	11	48	35	55	69
51	35	44	11	48	53	55	69

51	35	44	11	48	53	55	69

Invariant de boucle

- Le tableau vert est trié
- **Tous** les nombres du tableau bleu sont inférieurs ou égaux à **tous** les nombres du tableau vert

Si on voulait prouver l'exactitude du tri par sélection, on utiliserait donc l'invariant de boucle suivant :

- $tab[i \dots n]$ est trié
- et $\forall 1 \leq k < i, tab(k) \leq tab(i)$

fonction indiceDuPlusGrand(i, tab) **retourne** entier

- entrées :
  - i : indice entre 1 et n
  - tab : tableau indicé de 1 à n
- sortie :
  - indice où se trouve le plus grand élément dans $tab[1 \dots i]$

début

1 $jMax \leftarrow 1$
2 **pour** $j \leftarrow 2$ **haut** i **faire**
3 **si** $tab[j\,] > tab[jMax]$ **alors**
4 $jMax \leftarrow j$
5 **fin si**
6 **fin pour**
7 retourner $jMax$

fin indiceDuPlusGrand

fonction indiceDuPlusGrand(i, tab) **retourne** entier

- entrées :
  - i : indice entre 1 et n
  - tab : tableau indicé de 1 à n
- sortie :
  - indice où se trouve le plus grand élément dans $tab[1 \dots i]$

c_1 est la somme des temps d'exécution des instructions 1 et 7

début

			temps d'exécution
1	$jMax \leftarrow 1$	1 fois	c_1
2	**pour** $j \leftarrow 2$ **haut** i **faire**	$(i-1)+1$ fois	c_2
3	**si** $tab[j\,] > tab[jMax]$ **alors**	$(i-1)$ fois	c_3
4	$jMax \leftarrow j$	l fois, $0 \leq l < i$	c_4
5	**fin si**		
6	**fin pour**		
7	retourner $jMax$	1 fois	

fin indiceDuPlusGrand

Analyse

$T(\text{indiceDuPlusGrand}(i)) = c_1 + ic_2 + (i-1)c_3 + lc_4$

<u>Pire cas</u> :

- $l = i - 1$
- $T(\text{indiceDuPlusGrand}(i)) = c_1 + ic_2 + (i-1)c_3 + (i-1)c_4$
  $= (c_2 + c_3 + c_4)i + c_1 - c_3 - c_4$

<u>Meilleur cas</u> :

- $l = 0$
- $T(\text{indiceDuPlusGrand}(i)) = c_1 + ic_2 + (i-1)c_3$
  $= (c_2 + c_3)i + c_1 - c_3$

procedure triSelection(tab)

- entrée :
  - tab : tableau indicé de 1 à n
- sortie :
  - tab : tableau indicé de 1 à n trié

début

			temps d'exécution
1	**pour** $i \leftarrow n$ **bas** 2 **faire**	$(n-1)+1$ fois	c_5
2	$iMax \leftarrow$ indiceDuPlusGrand(i, tab)	$n-1$ fois	c_6
3	**si** $i \neq iMax$	$n-1$ fois	c_7
4	$tab[i] \leftrightarrow tab[iMax]$	p fois, $0 \leq p < n$	c_8
5	**fin si**		
6	**fin pour**		

fin triSelection

Analyse au <u>pire cas</u>

$T(\text{triSelection}) = nc_5 + (n-1)c_6 + (n-1)c_7 + pc_8 + \sum_{i=2}^{n} T(DPG(i))$

- $p = n - 1$
- $\sum_{i=2}^{n} T(DPG(i)) = \sum_{i=2}^{n}\big((c_2 + c_3 + c_4)i + c_1 - c_3 - c_4\big)$

$= (c_2 + c_3 + c_4)\sum_{i=2}^{n} i + \sum_{i=2}^{n}(c_1 - c_3 - c_4)$

$= (c_2 + c_3 + c_4)\left(\frac{n(n+1)}{2} - 1\right) + (c_1 - c_3 - c_4)(n-1)$

$= \frac{c_2+c_3+c_4}{2}n^2 + \frac{2c_1+c_2-c_3-c_4}{2}n - c_1 - c_2$

- $T(\text{triSelection}) = nc_5 + (n-1)c_6 + (n-1)c_7 + (n-1)c_8 + \frac{c_2+c_3+c_4}{2}n^2 + \frac{2c_1 + c_2 - c_3 - c_4}{2}n - c_1 - c_2$

$= \frac{c_2 + c_3 + c_4}{2}n^2 + \frac{2c_1 + c_2 - c_3 - c_4 + 2c_5 + 2c_6 + 2c_7 + 2c_8}{2}n - c_1 - c_2 - c_6 - c_7 - c_8$

Analyse au <u>meilleur cas</u>

$T(\text{triSelection}) = nc_5 + (n-1)c_6 + (n-1)c_7 + pc_8 + \sum_{i=2}^{n} T(DPG(i))$

- $p = 0$
- $\sum_{i=2}^{n} T(DPG(i)) = \sum_{i=2}^{n}\big((c_2 + c_3)i + c_1 - c_3\big)$

$= (c_2 + c_3)\sum_{i=2}^{n} i + \sum_{i=2}^{n}(c_1 - c_3)$

$= (c_2 + c_3)\left(\frac{n(n+1)}{2} - 1\right) + (c_1 - c_3)(n-1)$

$= \frac{c_2+c_3}{2}n^2 + \frac{2c_1+c_2-c_3}{2}n - c_1 - c_2$

- $T(\text{triSelection}) = nc_5 + (n-1)c_6 + (n-1)c_7 + 0c_8 + \frac{c_2+c_3}{2}n^2 + \frac{2c_1 + c_2 - c_3}{2}n - c_1 - c_2$

$= \frac{c_2 + c_3}{2}n^2 + \frac{2c_1 + c_2 - c_3 + 2c_5 + 2c_6 + 2c_7}{2}n - c_1 - c_2 - c_6 - c_7$

Tri par insertion

Pour trier un tableau de taille n :

Pour i de 2 à n :

on place le i^e élément à la « bonne place » dans le tableau $tab[1 \ldots i]$

51	55	69	11	48	35	53	44

51	55	69	11	48	35	53	44

51	55	69	11	48	35	53	44
11	51	55	69	48	35	53	44

11	51	55	69	48	35	53	44
11	48	51	55	69	35	53	44

11	48	51	55	69	35	53	44
11	35	48	51	55	69	53	44

procedure triInsertion(tab)

- entrée :
  - tab : tableau indicé de 1 à n
- sortie :
  - tab : tableau indicé de 1 à n trié

début

		temps d'exécution
pour $i \leftarrow 2$ **haut** n **faire**	$(n-1)+1$ fois	c_1
$cle \leftarrow tab[i]$	$n-1$ fois	c_2
$j \leftarrow i$	$n-1$ fois	
tant que $(j > 1)$ et $(tab[j-1] > cle)$ **faire**	$1 \le q_i \le i$ fois	c_3
$tab[j] \leftarrow tab[j-1]$	$q_i - 1$ fois	c_4
$j \leftarrow j - 1$	$q_i - 1$ fois	
fin tant que		
$tab[j] \leftarrow cle$	$n-1$ fois	
fin pour		

fin triInsertion

Analyse au <u>pire cas</u>

$$T(\text{triInsertion}) = nc_1 + (n-1)c_2 + \sum_{i=2}^{n} q_i c_3 + \sum_{i=2}^{n}(q_i - 1)c_4$$

$\forall\, 2 \le i \le n,\ q_i = i$

$$T(\text{triInsertion}) = nc_1 + (n-1)c_2 + \sum_{i=2}^{n} i c_3 + \sum_{i=2}^{n}(i - 1)c_4$$

$$T(\text{triInsertion}) = nc_1 + (n-1)c_2 + c_3\sum_{i=2}^{n} i + c_4\sum_{i=1}^{n-1} i$$

$$T(\text{triInsertion}) = nc_1 + (n-1)c_2 + c_3\left(\frac{n(n+1)}{2} - 1\right) + c_4\frac{n(n-1)}{2}$$

$$T(\text{triInsertion}) = \frac{c_3 + c_4}{2}n^2 + \frac{2c_1 + 2c_2 + c_3 - c_4}{2}n - (c_2 + c_3)$$

Analyse au <u>meilleur cas</u>

$$T(\text{triInsertion}) = nc_1 + (n-1)c_2 + \sum_{i=2}^{n} q_i c_3 + \sum_{i=2}^{n}(q_i - 1)c_4$$

$\forall\, 2 \le i \le n,\ q_i = 1$

$$T(\text{triInsertion}) = nc_1 + (n-1)c_2 + \sum_{i=2}^{n} c_3$$

$$T(\text{triInsertion}) = nc_1 + (n-1)c_2 + (n-1)c_3$$

$$T(\text{triInsertion}) = (c_1 + c_2 + c_3)n - (c_2 + c_3)$$

Comparaison des deux algorithmes de tri

- Pire cas :
  - Sélection : $\frac{c_2+c_3+c_4}{2}n^2 + \frac{2c_1+c_2-c_3-c_4+2c_5+2c_6+2c_7+2c_8}{2}n - c_1 - c_2 - c_6 - c_7 - c_8$
  - Insertion : $\frac{c_3+c_4}{2}n^2 + \frac{2c_1+2c_2+c_3-c_4}{2}n - (c_2 + c_3)$
- Meilleur cas :
  - Sélection : $\frac{c_2+c_3}{2}n^2 + \frac{2c_1+c_2-c_3+2c_5+2c_6+2c_7}{2}n - c_1 - c_2 - c_6 - c_7$
  - Insertion : $(c_1 + c_2 + c_3)n - (c_2 + c_3)$

Opération ou instruction barométrique

Opération barométrique : opération ou instruction qui est exécutée au moins aussi souvent que n'importe quelle autre instruction ou opération de l'algorithme.
On lui affecte un temps d'exécution de 1 par la suite.

Avantage : simplification de l'analyse asymptotique d'un algorithme en laissant tomber les détails « négligeables ». Cela permet de trouver un **ordre de grandeur** de la complexité.

fonction indiceDuPlusGrand(i, tab) **retourne** entier
- entrées :
  - i : indice entre 1 et n
  - tab : tableau indicé de 1 à n
- sortie :
  - indice où se trouve le plus grand élément dans $tab[1 \dots i]$

« $T(\text{indiceDuPlusGrand}(i)) = i$ » dans tous les cas

début

$jMax \leftarrow 1$	1 fois
pour $j \leftarrow 2$ **haut** i **faire**	i fois
si $tab[j] > tab[jMax]$ **alors**	$(i-1)$ fois
$jMax \leftarrow j$	l fois, $0 \le l < i$
fin si	
fin pour	
retourner $jMax$	1 fois

fin indiceDuPlusGrand

procedure triSelection(tab)
- entrée :
  - tab : tableau indicé de 1 à n
- sortie :
  - tab : tableau indicé de 1 à n trié

Même opération barométrique que indiceDuPlusGrand

début

pour $i \leftarrow n$ **bas** 2 **faire**	n fois
$iMax \leftarrow \text{indiceDuPlusGrand}(i, tab)$	$n-1$ fois
si $i \neq iMax$	$n-1$ fois
$tab[i] \leftrightarrow tab[iMax]$	p fois, $0 \le p < n$
fin si	
fin pour	

fin triSelection

$$T(n) = \sum_{i=2}^{n} i = \frac{n(n+1)}{2} - 1 = \frac{1}{2}n^2 + \frac{n}{2} - 1$$

procedure triInsertion(tab)
- entrée :
  - tab : tableau indicé de 1 à n
- sortie :
  - tab : tableau indicé de 1 à n trié

début

pour $i \leftarrow 2$ **haut** n **faire**	$(n-1)+1$ fois
$cle \leftarrow tab[i]$	$n-1$ fois
$j \leftarrow i$	$n-1$ fois
tant que $(j > 1)$ et $(tab[j-1] > cle)$ **faire**	$1 \le q_i \le i$ fois
$tab[j] \leftarrow tab[j-1]$	$q_i - 1$ fois
$j \leftarrow j - 1$	$q_i - 1$ fois
fin tant que	
$tab[j] \leftarrow cle$	$n-1$ fois
fin pour	

fin triInsertion

Analyse simplifiée

- Pire cas :

$$T(n) = \sum_{i=2}^{n} \sum_{j=1}^{i} 1 = \sum_{i=2}^{n} i = \frac{n(n+1)}{2} - 1 = \frac{1}{2}n^2 + \frac{n}{2} - 1$$

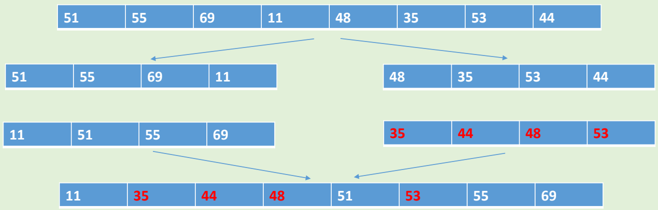
- Meilleur cas :

$$T(n) = \sum_{i=2}^{n} 1 = n - 1$$

Tri par fusion

Algorithme **récursif** :

- On coupe en deux le tableau
- On trie chacun des deux tableaux ainsi obtenu
- On fusionne les deux tableaux triés



procedure fusionner (*tab, debut,* fin_1, fin_2, *temp*)

- entrées :
  - tab : tableau indicé de 1 à n
  - *debut,* fin_1, fin_2 indices entre 1 et n
- sortie :
  - $temp$: tableau indicé de 1 à n : fusion de $tab[debut \dots fin_1]$ et $tab[fin_1 + 1 \dots fin_2]$

début

$i \leftarrow debut$; $j \leftarrow fin_1 + 1$ — 1 fois

pour $k \leftarrow debut$ **haut** fin_2 **faire** — $fin_2 - debut + 2$ fois

  si $i \leq fin_1$ **et** $(j > fin_2$ **ou** $tab[i] < tab[j])$ **alors** — $fin_2 - debut + 1$ fois

    $temp[k] \leftarrow tab[i]$; $i \leftarrow i + 1$

  sinon

    $temp[k] \leftarrow tab[j]$; $j \leftarrow j + 1$

  fin si

fin pour

fin fusionner

Temps d'exécution : c_1, c_2, c_3

Analyse dans tous les cas

$$T(\text{fusionner}) = c_1 + c_2(fin_2 - debut + 2) + c_3(fin_2 - debut + 1)$$

$$= c_1 + c_2 + (c_2 + c_3)(fin_2 - debut + 1)$$

Donc si $fin_2 = n$ et $debut = 1$:

$T(\text{fusionner}) = (c_2 + c_3)n + c_1 + c_2$

procedure triFusion(tab, bi, bs)

- entrées :
  - tab : tableau indicé de 1 à n
  - bi, bs indices entre 1 et n
- sortie :
  - tab : tableau indicé de 1 à n trié

début — temps d'exécution

si $bi < bs$ **alors** — 1 fois — c_4

  $milieu \leftarrow \frac{bi+bs}{2}$ — 1 fois

  triFusion$(tab, bi, milieu)$

  triFusion$(tab, milieu + 1, bs)$

  fusionner$(tab, bi, milieu, bs, tabTemp)$

  pour $i \leftarrow bi$ **haut** bs **faire** — $bs - bi + 2$ fois — c_5

    $tab[i] \leftarrow tabTemp[i]$ — $bs - bi + 1$ fois — c_6

  fin pour

fin si

fin triFusion

Analyse de triFusion dans tous les cas

Si $bi = 1$ et $bs = n$, et si on suppose que n est pair :

$$T(n) = 2T\left(\frac{n}{2}\right) + (c_2 + c_3)n + c_1 + c_2 + c_4 + c_5(n + 1) + c_6 n$$

$$T(n) = 2T\left(\frac{n}{2}\right) + (c_2 + c_3 + c_5 + c_6)n + c_1 + c_2 + c_4 + c_5 = ???$$