Efficacité des algorithmes

Comment choisir parmi les différentes approches pour résoudre un problème?

Exemple: Liste chaînée ou tableau?

2 objectifs à atteindre:

1. Concevoir un algorithme facile à comprendre, coder et déboguer.
2. Concevoir un algorithme qui utilise de façon efficace les ressources de l'ordinateur.

Efficacité des algorithmes(suite)

Objectif (1): concerne le génie logiciel

Objectif (2): Algorithmes et structures de données.

Lorsque l'objectif (2) est important, comment peut-on mesurer l'efficacité d'un algorithme?

Comment mesurer l'efficacité?

2. Comparaison empirique: (exécuter le programme)
3. Analyse assymptotique d'algorithmes

Ressources critiques: temps, espace mémoire,...

Facteurs affectant le temps d'exécution:
machine, language, programmeur, compilateur, algorithme et structure de données.

En général, le temps d'exécution dépend de la longueur de l'entrée.

Ce temps est une fonction **T**(*n*) où *n* est la longueur de l'entrée.

Exemples

Exemple 1.

```
// Retourne l'indice du plus grand élément

int PlusGrand(int T[], int n) {
  int max = 0;
  for (int i=1; i<n; i++)
    if (T[max] < T[i])
      max = i;
  return max;
}
```

Exemples (suite)

Exemple 2: x=3;

Exemple 3:

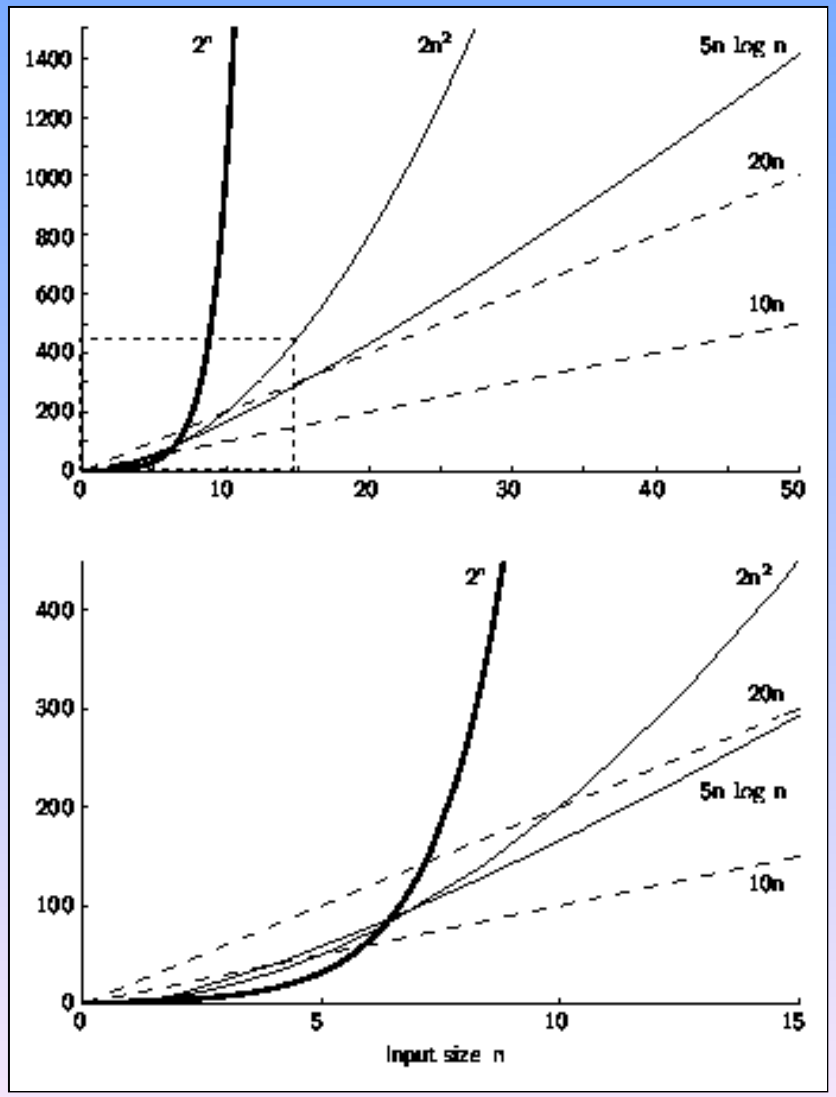
```
sum = 0;
for (i=1; i<=n; i++)
  for (j=1; j<=n; j++)
    sum++;
}
```

Meilleur algorithme ou ordinateur?

	$\mathbf{n = 10}$	$\mathbf{n = 1000}$	$\mathbf{n = 100000}$	
$\mathbf{n}$	10	1000	10^5	secondes
$\mathbf{10n}$	100	10^4	10^6	
$\mathbf{100n}$	1000	10^5	10^7	
$\mathbf{n^2}$	100	10^6	10^{10}	heures
$\mathbf{10n^2}$	1000	10^7	10^{11}	
$\mathbf{100n^2}$	10^4	10^8	10^{12}	
$\mathbf{n^3}$	1000	10^9	10^{15}	années
$\mathbf{10n^3}$	10^4	10^{10}	10^{16}	
$\mathbf{100n^3}$	10^5	10^{11}	10^{17}	
$\mathbf{2^n}$	1024	$> 10^{301}$	∞	
$\mathbf{10 \cdot 2^n}$	$> 10^4$	$> 10^{302}$	∞	
$\mathbf{100 \cdot 2^n}$	$> 10^5$	$> 10^{303}$	∞	
$\lg \mathbf{n}$	3.3	9.9	16.6	
$\mathbf{10} \lg \mathbf{n}$	33.2	99.6	166.1	
$\mathbf{100} \lg \mathbf{n}$	332.2	996.5	1661.0	

On suppose que l'ordinateur utilisé peut effectuer 10^6 opérations à la seconde

Taux de croissance



Analyse assymptotique: Grand O

Définition: **T**(n) est dans O(f(n)) s'il existe deux constantes positives c et n_0 t.q.

T(n) ≤ cf(n) pour tout $n > n_0$.

Utilité: Simplifie l'analyse

Signification: Pour toutes les grandes entrées (i.e., $n \geq n_0$), on est assuré que l'algorithme ne prend pas plus de cf(n) étapes.

⇒Borne supérieure.

Notation grand-O (suite)

La notation grand-O indique une **borne supérieure** sur le temps d'exécution.

Exemple: Si $\mathbf{T}(n) = 3n^2 + 2$

alors $\mathbf{T}(n) \in O(n^2)$.

On désire le plus de précision possible:

Bien que $\mathbf{T}(n) = 3n^2 + 2 \in O(n^3)$,

on préfère $O(n^2)$.

Grand-O: Exemples

Exemple 1: Initialiser un tableau d'entiers

for (int i=0; i<n; i++) Tab[i]=0;

- Il y a n itérations
- Chaque itération nécessite un temps ≤ c,

où c est une constante.

- Le temps est donc $T(n) \leq cn$
- Donc $T(n) \in O(n)$

Grand-O: Exemples

Exemple 2: $\mathbf{T}(n) = c_2n^2 + c_1n + c_0$.

$$c_2n^2 + c_1n + c_0 \leq c_2n^2 + c_1n^2 + c_0n^2$$
$$= (c_2 + c_1 + c_0)n^2$$

pour tout $n \geq 0$.

$\mathbf{T}(n) \leq cn^2$ où $c = c_2 + c_1 + c_0$

Donc, $\mathbf{T}(n)$ est dans $O(n^2)$.

Grand-O: Exemples

Exemple 3:

Lorsque $T(n) = c$, où c est une constante,

on écrit $T(n) \in O(1)$.

Grand-Omega

Definition: On a $\mathbf{T}(n) \in \Omega(g(n))$ s'il existe deux constantes positives c et n_0 telles que $\mathbf{T}(n) \geq cg(n)$ pour tout $n > n_0$.

Signification: Pour de grandes entrées, l'exécution de l'algorithme nécessite au moins $cg(n)$ étapes.

$\Rightarrow$Borne inférieure.

Grand-Omega: Exemple

$\mathbf{T}(n) = c_2 n^2 + c_1 n + c_0$

$c_2 n^2 + c_1 n + c_0 \geq c_1 n^2$ pour tout $n > 0$.

$\mathbf{T}(n) \geq cn^2$ pour $c = c_1$ et $n > 0$.

Donc, $\mathbf{T}(n) \in \Omega(n^2)$ par la définition.

La notation Theta

Lorsque le grand-O et le grand-omega d'une fonction coïncident, on utilise alors la notation grand-theta.

Définition: Le temps d'exécution d'un algorithme est dans Θ ($h(n)$) s'il est à la fois dans O($h(n)$) et dans Ω ($h(n)$).

Exemple

	$n = 10$	$n = 1000$	$n = 100000$	
$\Theta(n)$	n	10	1000	10^5	secondes
	$10n$	100	10^4	10^6	
	$100n$	1000	10^5	10^7	
$\Theta(n^2)$	n^2	100	10^6	10^{10}	heures
	$10n^2$	1000	10^7	10^{11}	
	$100n^2$	10^4	10^8	10^{12}	
$\Theta(n^3)$	n^3	1000	10^9	10^{15}	années
	$10n^3$	10^4	10^{10}	10^{16}	
	$100n^3$	10^5	10^{11}	10^{17}	
$\Theta(2^n)$	2^n	1024	$> 10^{301}$	∞	
	$10 \cdot 2^n$	$> 10^4$	$> 10^{302}$	∞	
	$100 \cdot 2^n$	$> 10^5$	$> 10^{303}$	∞	
$\Theta(\lg n)$	$\lg n$	3.3	9.9	16.6	
	$10 \lg n$	33.2	99.6	166.1	
	$100 \lg n$	332.2	996.5	1661.0	

$$O(\lg n) \subset O(n) \subset O(n^2) \subset O(n^3) \subset O(2^n)$$

Pire cas, meilleur cas et cas moyen

Toutes les entrées d'une longueur donnée ne nécessitent pas le même temps d'exécution.

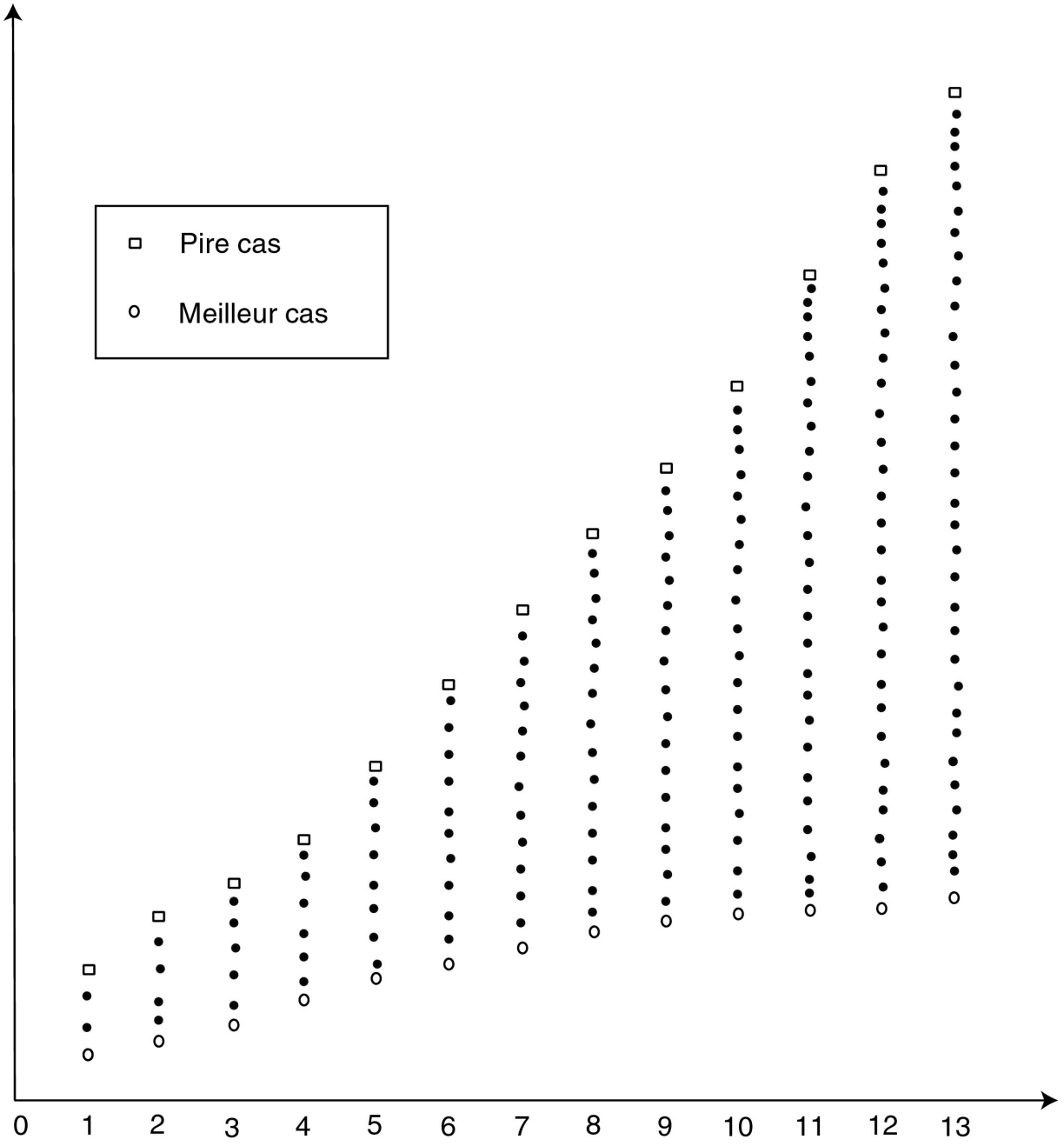
Exemple: Recherche séquentielle dans un tableau de taille n:

Commencer au début du tableau et considérer chaque élément jusqu'à ce que l'élément cherché soit trouvé.

Meilleur cas: $\Theta(1)$

Pire cas: $\Theta(n)$

Cas moyen: $\Theta(n)$


Pire cas
Meilleur cas
0
1
2
3
4
5
6
7
8
9
10
11
12
13

Une erreur fréquente

"Le meilleur cas pour mon algorithme est *n*=1 car c'est le plus rapide." FAUX!

On utilise la notation grand-O parce qu'on s'intéresse au comportement de l'algorithme lorsque *n* augmente.

Meilleur cas: on considère toutes les entrées de longueur n.

Une erreur fréquente

Confondre le pire cas avec la borne supérieure.

La borne supérieur réfère au taux de croissance.

Le pire cas réfère à l'entrée produisant le plus long temps d'exécution parmi toutes les entrées d'une longueur donnée.

Règles de simplification 1

Si

$$f(n) \in O(g(n))$$

et

$$g(n) \in O(h(n)),$$

alors

$$f(n) \in O(h(n)).$$

Règles de simplification 2

Si

$$f(n) \in O(kg(n))$$

où $k > 0$ est une constante

alors

$$f(n) \in O(g(n)).$$

Règles de simplification 3

Si

$$f_1(n) \in O(g_1(n))$$

et

$$f_2(n) \in O(g_2(n)),$$

alors

$$f_1(n) + f_2(n) \in O(\max(g_1(n), g_2(n))).$$

Règles de simplification 4

Si

$$f_1(n) \in O(g_1(n))$$

et

$$f_2(n) \in O(g_2(n))$$

alors

$$f_1(n) * f_2(n) \in O(g_1(n) * g_2(n))$$

Règles de simplification 5

Si $p(n)$ est un polynôme de degré k alors $p(n) \in O(n^k)$

Règles de simplification 6

$$\log^k n \in O(n^\varepsilon)$$

pour toutes constantes $k>0$ et $\varepsilon > 0$

Exemples

Exemple 1: `a = b;`

Temps constant: $\Theta(1)$.

Exemple 2:

```
somme = 0;
for (i=1; i<=n; i++)
  somme += n;
```

Temps: $\Theta(n)$

Exemples

Exemple 3:

```
somme = 0;
for (i=1; i<=n; i++)
  for (j=1; j<=i; j++)
    somme++;
```

Temps: $\Theta(1) + O(n^2) = O(n^2)$

On peut montrer : $\Theta(n^2)$

Exemples

Exemple 4:

```
somme = 0;
for (j=1; j<=n; j++)
  for (i=1; i<=n; i++)
  somme++;
for (k=0; k<n; k++)
  A[k] = k;
```

Temps: $\Theta(1) + \Theta(n^2) + \Theta(n) = \Theta(n^2)$

Exemples

Example 5:

```
somme = 0;
for (k=1; k<=n; k*=2)
  for (j=1; j<=n; j++)
    somme++;
```

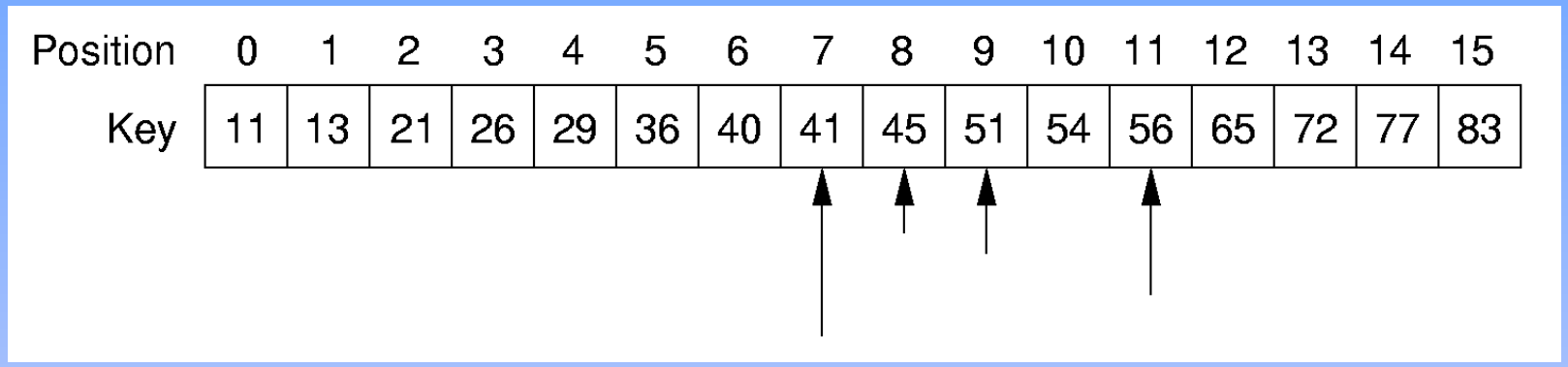
Temps: Θ (nlog n)

Recherche dychotomique

```
// Retourne la position de l'élément K
// dans un tableau trié de taille n

int dycho(int tab[], int n, int K) {
  int l = 0;
  int r = n-1;
  while (l <= r) {
    int m = (l+r)/2;
    if (tab[m] < K) l=m+1;
    else if (tab[m] > K) r=m-1;
    else return m;
  }
  return n; // K n'est pas dans tab
}
```

Recherche dychotomique



Combien d'éléments sont examinés dans le pire cas?

Autres exemples

Instruction `if`: maximum entre le `if` et le `then`

`switch`: maximum entre les différents cas

Appels de fonction: Temps d'exécution de la fonction

Analyse de problèmes

Borne supérieure: Borne supérieur sur le meilleur algorithme connu.

Borne inférieure: Borne inférieure sur tous les algorithmes possibles.

Analyse de problèmes : Exemple

Remarque: Il n'y a aucune différence entre les bornes inférieure et supérieure lorsque le temps d'exécution exact est connu.

Exemple de connaissance imparfaite:

1. Coût des E/S: $\Omega(n)$.
2. Tri par insertion et selection: $O(n^2)$.
3. Meilleurs tris (Quicksort, Mergesort, Heapsort, etc.): $O(n \log n)$.

Espace mémoire

La quantité d'espace mémoire utilisé peut aussi être étudiée à l'aide de l'analyse assymptotique.

En général:

Temps: Algorithme manipulant les structures de données.

Espace: Structure de données

Le principe du compromis espace/temps

Il est souvent possible de réduire le temps d'exécution au prix d'augmenter l'espace mémoire et vice versa.

- Fichiers compressés
- Table des valeurs d'une fonctions
- Insérer un nœud dans une liste chaînée

Espace disque: Un programme sera d'autant plus rapide qu'il requiert peu d'accès disque.

Retour aux listes
Comparaison des implémentations

Listes avec tableau:

- Insertion et retrait sont $\Theta(n)$.
- Précédent et accès direct sont $\Theta(1)$.
- Tout l'espace est alloué à l'avance.
- Pas d'espace autre que les valeurs.

Listes chaînées:

- Insertion et retrait sont $\Theta(1)$.
- Précédent et accès direct sont $\Theta(n)$.
- L'espace augmente avec la liste.
- Chaque élément requiert de l'espace pour les pointeurs.

Comparaison de l'espace utilisé

E: Espace mémoire pour les valeurs.
P: Espace mémoire pour les pointeurs.
D: Nombre d.éléments dans le tableau.

$$DE = n(P + E)$$

$$n = \frac{DE}{P + E}$$