Complexité

- Introduction à la complexité
  - en place mémoire utilisée
  - en temps de calcul

Complexité en place mémoire utilisée

En simplifiant

- pour un booléen, caractère, nombre borné : *1*
- pour un tableau ou un ensemble :

nombre d'éléments x taille d'un élément

Complexité

- Introduction à la complexité
  - en temps de calcul
  - en place utilisée

Complexité en temps de calcul

pour simplifier ….

La complexité **théorique** d'un algorithme est le nombre d'opérations élémentaires exécutées par l'algorithme en fonction de la taille de la donnée et dans le plus mauvais des cas.

Complexité

Opération élémentaire

Opération dont le temps d'exécution est borné par une constante (ce temps est indépendant de la donnée)

Exemples

- Affectation de variables simples
- Accès à un élément de tableau, à un attribut
- Opérations booléennes, comparaisons
- Opérations arithmétiques ordinaires

Complexité

Exemple d’un calcul de moyenne

```
public static double moyenne(double a, double b){
        double somme = a+b;
        return somme/2;
}
```

Evaluation de la complexité : constante

- 4 opérations élémentaires
  - 1 affectation, 2 opérations arithmétiques, 1 opération "retourner”
- 4 emplacements mémoire de la taille d’un double

Complexité

Exemple d’une recherche dans un tableau de taille n

```
public static boolean recherche(int[] tab, int v){
        for (int i=0; i<tab.length; i++)
                if (tab[i]==v) return true;
        return false;
}
```

Evaluation de la complexité : linéaire par rapport à n

- opérations élémentaires : au plus 5n+1

  < n affectations de valeurs à i, < n comparaisons de i avec tab.length,
  < n incrémentations de i, < n accès à tab, < n comparaisons,
  1 opération "retourner”

- emplacements mémoire de la taille d’un int : n+2 et 1 booléen

Complexité

Exemple de recherche dans un tableau **trié** de taille n (par dichotomie)

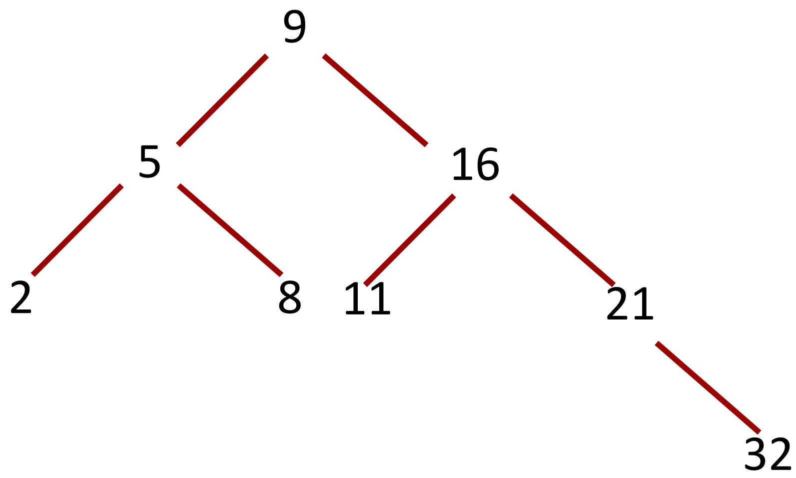
```
public static boolean rechercheDicho(int [] t, int v) {
    boolean stop=false, res=false;
    int indi=0, indf=t.length-1; int indm, valm;
    while ( stop == false ) {
        if ( indi > indf ) stop = true;
        else {
                indm = (indi+indf)/2;
                valm = t[indm];
                if ( valm == v ) {stop = true; res = true; } // on a trouvé !
                else
                 if ( v < valm ) indf = indm-1; // chercher à gauche
                 else indi = indm+1;  } } // chercher à droite
     return res ;
}
```

- Evaluation de la complexité : de l’ordre de $\log_2(n)$

Complexité

Exemple de recherche dans un tableau **trié** de taille n (par dichotomie)

Recherche dans [2, 5, 8, 9, 11, 16, 21, 32]



$n = 2^h$ $h = \log_2(n)$

Dans un arbre binaire complet de hauteur h
nombre de nœuds = floor($2^{h+1}-1$)

On descend dans le pire des cas sur une branche de la profondeur de l'arbre

Evaluation de la complexité : de l'ordre de $\log_2(n)$

Complexité

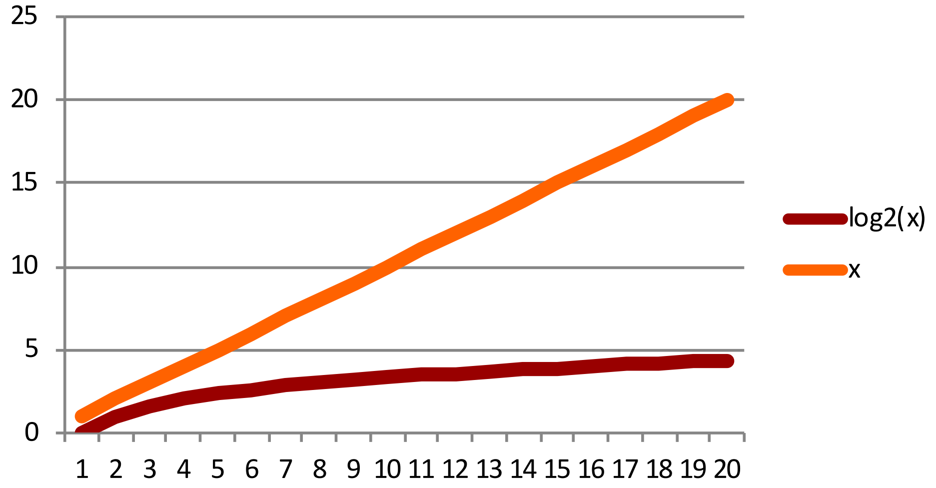
Exemple du tri par sélection d’un tableau de taille n

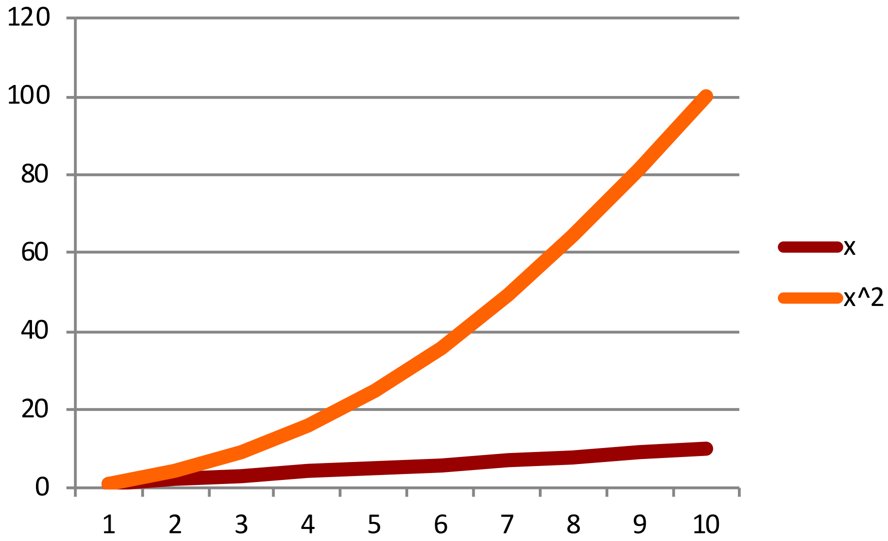
```
public static void triSelection(int []arr){
   int indiceDuMin = 0;
   for(int i = 0; i < arr.length; i++) {
       indiceDuMin = i;
       for(int j = i + 1; j < arr.length; j++)
            if(arr[j] < arr[indiceDuMin])
                indiceDuMin = j;
       int temp = arr[i]; arr[i] = arr[indiceDuMin]; arr[indiceDuMin] = temp;
   }
}
```

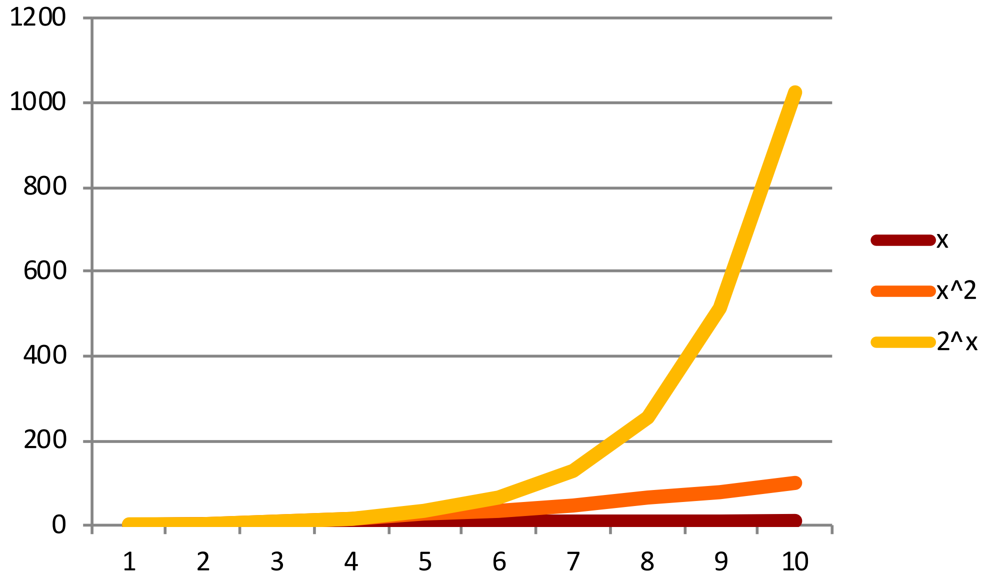
Evaluation de la complexité : de l’ordre de n^2, quadratique par rapport à n

Examen des comparaisons :
i=0 : n-1 comparaisons de arr[j] et de arr[indiceDuMin]
i=1 : n-2 comparaisons de arr[j] et de arr[indiceDuMin]
….. environ n(n+1)/2 comparaisons

Complexité


25
20
15
10
5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
log2(x)
x

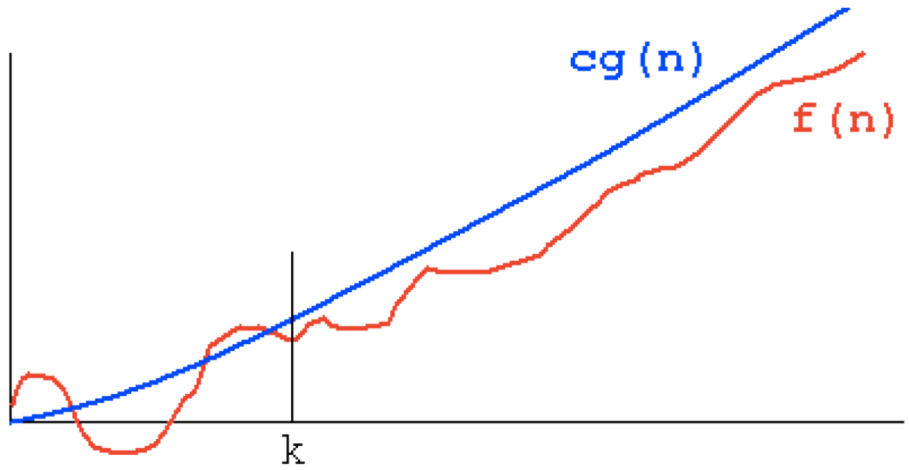

120
100
80
60
40
20
0
1 2 3 4 5 6 7 8 9 10
x
x^2


1200
1000
800
600
400
200
0
1 2 3 4 5 6 7 8 9 10
x
x^2
2^x

Complexité

- Des courbes, on tire que ce qui est important pour l'évaluation et la comparaison des algorithmes est la tendance quand la taille de la donnée augmente

- Pour $g(n)$, $O(g(n))$ est l'ensemble des fonctions $f(n)$ telles qu'il existe un réel $c > 0$ et un entier $k > 0$ tel que pour tout $n > k$, $f(n) < c \times g(n)$.



Complexité

- Si on s'intéresse au pire des cas
  - Recherche séquentielle **O**(n)
  - Recherche dichotomique **O**($\log_2(n)$)
  - Tri par sélection **O**(n^2)
- On peut faire aussi des analyses
  - en moyenne (ex. coût moyen d'une recherche)
  - amorties (ex. coût de l'ajout de n éléments)