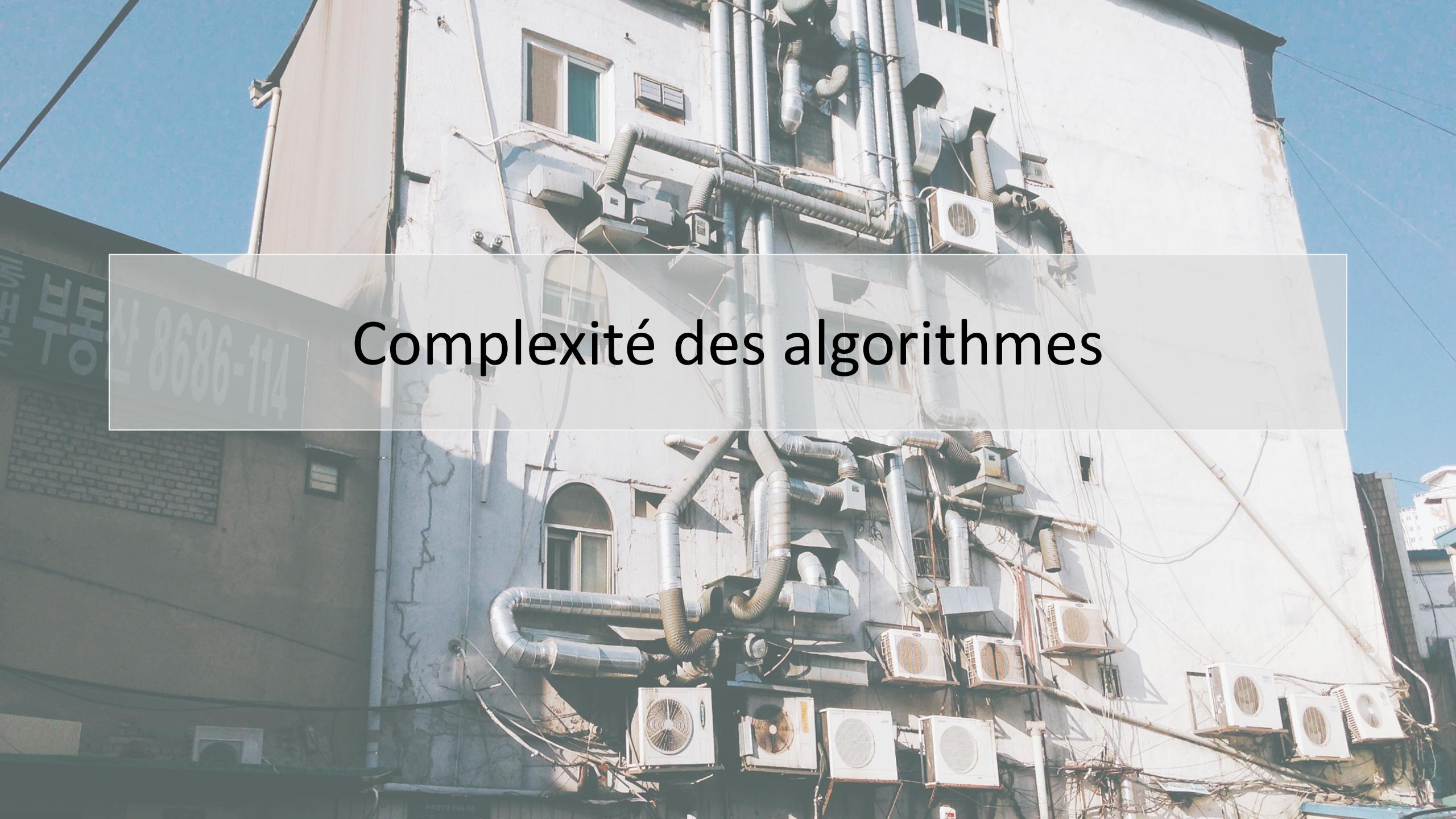


Complexité des algorithmes



Objectif

- Introduction à la notion de complexité

Introduction

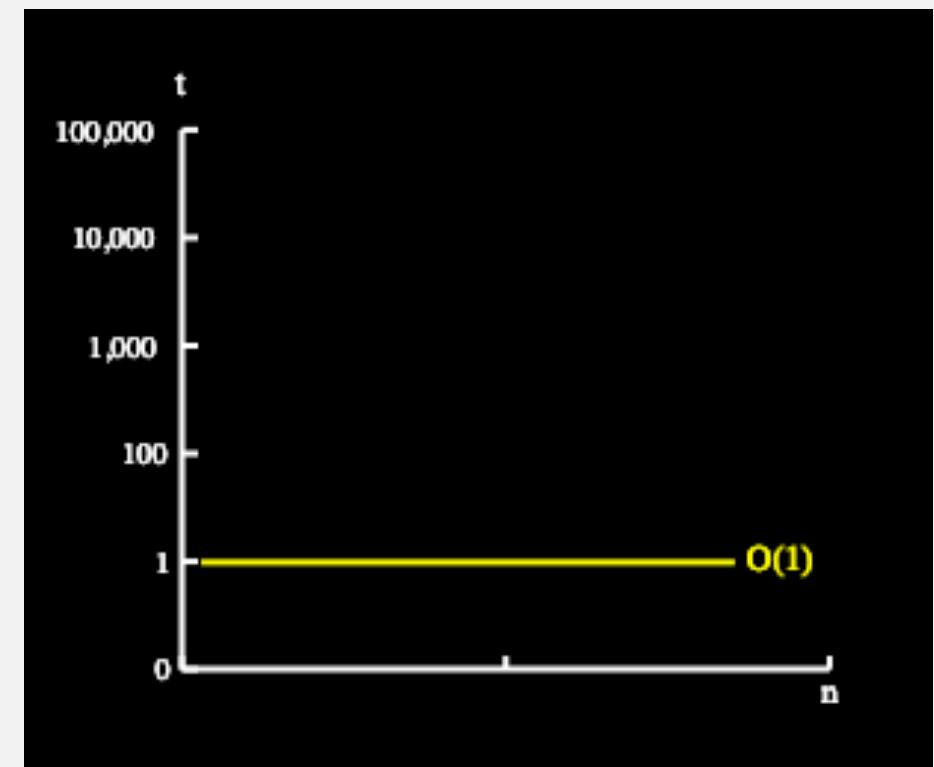
- La question souvent posée est de savoir comment comparer deux algorithmes
- En simplifiant, on cherche à compter le nombre d'instructions élémentaires exécutées dans un cas moyen ou dans le pire des cas
- On fait cette **évaluation par rapport au nombre de données** en entrée. Ce nombre est noté « n »

Introduction

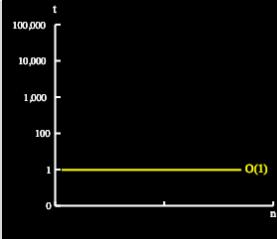
- Dans la recherche de la complexité, **on simplifie le résultat**, on cherche un **ordre de grandeur**
- La notation de Landau en grand O permet de donner un **ordre de grandeur** de la complexité d'un algorithme

$O(1)$ – temps constant

- $O(1)$ signifie que l'algorithme s'exécute en temps constant
- Généralement $O(1)$ correspond à un instruction ou une suite d'instructions :
 - Affectation
 - Opération mathématiques
 - Accès aux tableaux



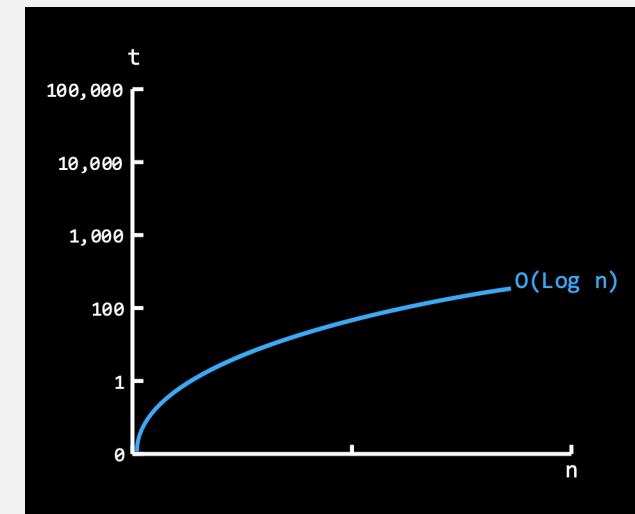
Exemple - O(1)



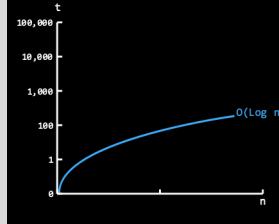
```
public Facture()
{
    this.LignesFacture = List<LigneFacture>();
}
```

$O(\log_2(n))$ – temps logarithmique

- $O(\log_2(n))$ signifie que le temps d'exécution de l'algorithme est en temps logarithmique. Si on augmente par deux son nombre de données, le temps n'évoluera que de très peu
- $T(n) = T(1) + T(n/2)$
- Généralement $O(\log_2(n))$ correspond à des divisions du nombre de données par 2 à chaque itération



Exemple - $O(\log_2(n))$

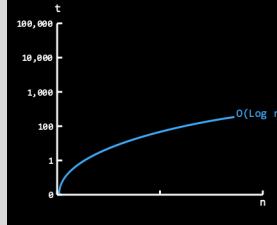


```
public static bool RechercherValeurDichotomie<TypeElement>(
    List<TypeElement> p_valeurs, TypeElement p_valeurAChercher,
    Func<TypeElement, TypeElement, bool> p_sontEgales, Func<TypeElement, TypeElement, bool> p_estPlusPetitEgaleA)
{
    bool estTrouvee = false;
    int indicePremier = 0;
    int indiceDernier = p_valeurs.Count - 1;
    int indiceMilieu = 0;

    while (!estTrouvee && indicePremier <= indiceDernier)
    {
        indiceMilieu = (indicePremier + indiceDernier) / 2;
        if (p_sontEgales(p_valeurs[indiceMilieu], p_valeurAChercher))
        {
            estTrouvee = true;
        }
        else if (p_estPlusPetitEgaleA(p_valeurs[indiceMilieu], p_valeurAChercher))
        {
            indicePremier = indiceMilieu + 1;
        }
        else
        {
            indiceDernier = indiceMilieu - 1;
        }
    }

    return estTrouvee;
}
```

Exemple - $O(\log_2(n))$



```
public static bool RechercherValeurDichotomie<TypeElement>(
    List<TypeElement> p_valeurs, TypeElement p_valeurAChercher,
    Func<TypeElement, TypeElement, bool> p_sontEgales, Func<TypeElement, TypeElement, bool> p_estPlusPetitEgaleA)
{
    bool estTrouvee = false;
    int indicePremier = 0;
    int indiceDernier = p_valeurs.Count - 1;
    int indiceMilieu = 0;

    while (!estTrouvee && indicePremier <= indiceDernier)
    {
        indiceMilieu = (indicePremier + indiceDernier) / 2;
        if (p_sontEgales(p_valeurs[indiceMilieu], p_valeurAChercher))
        {
            estTrouvee = true;
        }
        else if (p_estPlusPetitEgaleA(p_valeurs[indiceMilieu], p_valeurAChercher))
        {
            indicePremier = indiceMilieu + 1;
        }
        else
        {
            indiceDernier = indiceMilieu - 1;
        }
    }

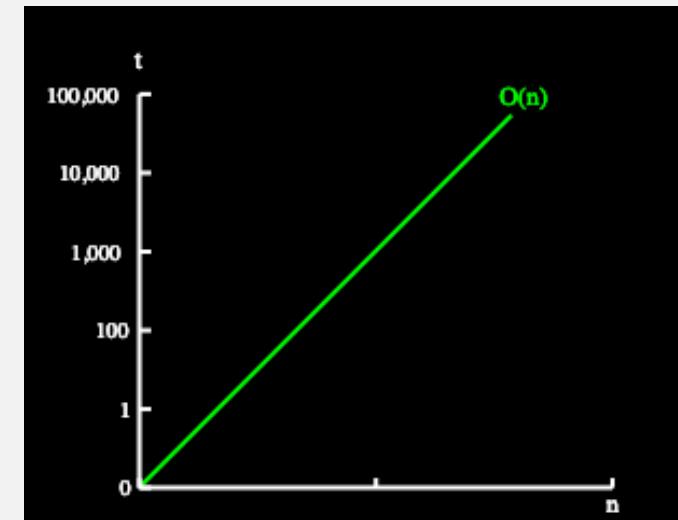
    return estTrouvee;
}
```

Tour 1 : n données
Tour 2 : $n/2$ données
Tour 3 : $n/4$ données
Tour 4 : $n/8$ données
Tour 5 : $n/16$ données
...
Tour $\log_2(n)$: 1 données

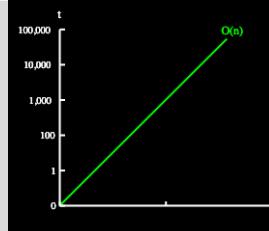
$O(n)$ – temps linéaire

- $O(n)$ signifie que le temps d'exécution de l'algorithme est en temps linéaire. Par exemple, si l'algorithme prend 1 minute pour traiter 1000 éléments, il en mettra 2 pour en traiter 2000.
- $T(n) = T(1) + T(n - 1)$
- Généralement $O(n)$ correspond à l'utilisation d'un boucle
- Exemple :

```
for (int numeroElement = 0;
      numeroElement < nbElements;
      ++numeroElement) {
  ...
}
```



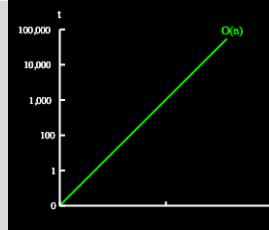
Exemple - $O(n)$



```
public static TypeElement[] CopierTableau<TypeElement>(TypeElement[] p_valeurs)
{
    TypeElement[] copieTableau = new TypeElement[p_valeurs.Length];
    for (int indiceValeurCourante = 0;
        indiceValeurCourante < p_valeurs.Length;
        indiceValeurCourante++)
    {
        copieTableau[indiceValeurCourante] = p_valeurs[indiceValeurCourante];
    }

    return copieTableau;
}
```

Exemple - O(n) – Calcul intersection avec listes triées

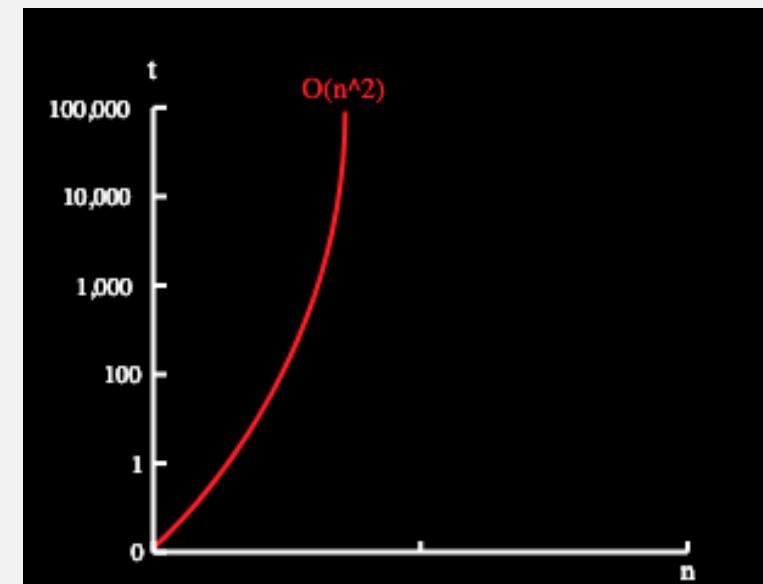


```
public static List<TypeElement> CalculerIntersection<TypeElement>(List<TypeElement> p_ensemble1,  
                                         List<TypeElement> p_ensemble2)  
    where TypeElement : IComparable<TypeElement>  
{  
    List<TypeElement> intersection = new List<TypeElement>();  
    int indiceEnsemble1 = 0;  
    int indiceEnsemble2 = 0;  
  
    while (indiceEnsemble1 < p_ensemble1.Count && indiceEnsemble2 < p_ensemble2.Count) {  
        TypeElement valeurCouranteEnsemble1 = p_ensemble1[indiceEnsemble1];  
        TypeElement valeurCouranteEnsemble2 = p_ensemble2[indiceEnsemble2];  
        int comparaison = valeurCouranteEnsemble1.CompareTo(valeurCouranteEnsemble2);  
        if (comparaison == 0) {  
            intersection.Add(valeurCouranteEnsemble1);  
            indiceEnsemble1++;  
            indiceEnsemble2++;  
        }  
        else if (comparaison < 0) {  
            indiceEnsemble1++;  
        }  
        else { // comparaison > 0  
            indiceEnsemble2++;  
        }  
    }  
  
    return intersection;  
}
```

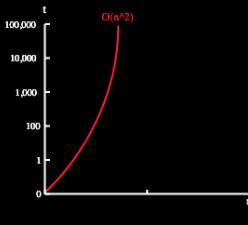
$O(n^2)$ – Temps quadratique

- $O(n^2)$ signifie que le temps augmente très rapidement dès que vous ajoutez des données
- $T(n) = n + T(n - 1)$
- Généralement $O(n^2)$ se déduit d'une boucle imbriquée dans une autre
- Exemple :

```
for (int i=0; i < n; i++) {  
    for(int j=0; j< n; j++)  
    { ... }  
}
```



Exemple - $O(n^2)$



```
public static List<TypeElement> CalculerIntersectionN2<TypeElement>(List<TypeElement> p_ensemble1,  
                                         List<TypeElement> p_ensemble2)  
{  
    List<TypeElement> intersection = new List<TypeElement>();  
  
    foreach (TypeElement element in p_ensemble1)  
    {  
        if (p_ensemble2.Contains(element))  
        {  
            intersection.Add(element);  
        }  
    }  
  
    return intersection;  
}
```

Autres complexités

