Complexité d'un algorithme

Soit f(n) une fonction de n.

On dira qu'un algorithme est en O(f(n)) si son temps d'exécution est proportionnel à f(n).

On dira qu'un algorithme est en O(f(n)) s'il existe deux nombres positifs constants k et n_0 tels que $T(n) \le k*f(n)$ quand $n = n_0$.

La fonction f(n) est appelée l'ordre de complexité de l'algorithme

Les cas les plus fréquemment rencontrés sont les suivants :

O(1): durée indépendante de n

O(log n): complexité logarithmique

O(n): complexité linéaire

O(n * log n) : complexité quasi-linéaire

O(n²): complexité quadratique

O(n³): complexité cubique

O(n^d): complexité polynomiale

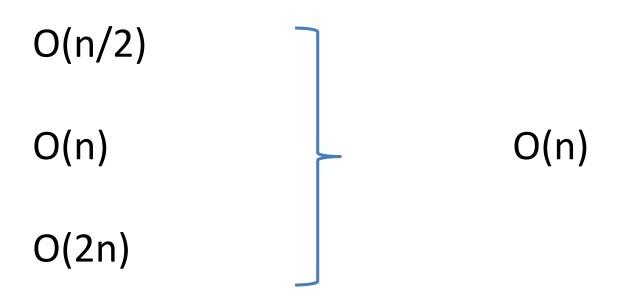
O(2ⁿ): complexité exponentielle

O(n!): complexité factorielle

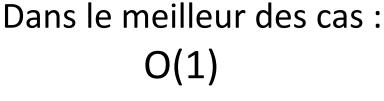


n ↓	1	log n	n	n*log n	n²	n³	2 ⁿ	n!
10	1	3	10	30	100	1000	1000	3628800
20	1	4	20	80	400	8000	1000000	2*10 ¹⁸
100	1	7	100	700	10000	1000000	10 ³⁰	
1000	1	10	1000	10000	10 ⁶	10 ⁹	10 ³⁰⁰	
10000	1	13	10000	130000	10 ⁸	10 ¹²		
10 ⁶	1	20	1000000	20000000	10 ¹²	10 ¹⁸		
10 ⁹	1	30	10 ⁹	30*10 ⁹	10 ¹⁸	10 ²⁷		

Le facteur de proportionnalité est négligé. Il ne sera utilisé que pour départager deux algorithmes qui ont le même ordre de complexité.



```
public boolean contient(int unEntier) {
   for (int i = 0; i < this.nombreDEntiers; i++) {</pre>
       if(this.tableDEntiers[i] == unEntier)
          return true;
   return false;
}
```



3 9 8 5 1

```
public boolean contient(int unEntier) {
   for (int i = 0; i < this.nombreDEntiers; i++) {</pre>
       if (this.tableDEntiers[i] == unEntier)
          return true;
   return false;
}
               Dans le pire des cas :
                 3
                      9
                            8
                                 5
```

```
public boolean contient(int unEntier) {
   for (int i = 0; i < this.nombreDEntiers; i++) {</pre>
       if (this.tableDEntiers[i] == unEntier)
          return true;
   return false;
}
              Coût moyen:
                         O(n/2)
                3
                      9
                           8
                                 5
```

```
public boolean contient(int unEntier) {
   for (int i = 0; i < this.nombreDEntiers; i++) {
      if(this.tableDEntiers[i]==unEntier)
          return true;
   }
   return false;
}</pre>
```

En ignorant le facteur de proportionnalité:

O(n)

3 9 8 5 1

```
public boolean contient(int unEntier) {
    return trouverIndice(unEntier) != -1;
}
O(n)
```

O(n)

```
public boolean contientExAequo(){
   for (int i = 0; i < nombreDEntiers - 1; i++) {</pre>
       for (int j = i+1; j < nombreDEntiers; j++) {</pre>
            if (tableDEntiers[i] == tableDEntiers[j])
               return true;
       }
   return false;
```

```
public boolean supprimer(int unEntier) {
   for (int i = 0; i < nombreDEntiers; i++) {</pre>
      if(tableDEntiers[i] == unEntier) {
          for (int j = i; j < nombreDEntiers-1; j++)</pre>
             tableDEntiers[j]=tableDEntiers[j+1];
         nombreDEntiers--;
          return true;
   return false;
```

```
public boolean supprimer(int unEntier) {
  int indice = trouverIndice(unEntier);
  if(indice == -1)        O(n)
      return false;
  supprimerALIndice(indice);
  return true;
  O(n)
}
```

Coût amorti = 1!!!

Moyenne de n ajout

L'agrandissement de la table se fait tous les n ajouts si taille x 2