# NSI - Première

Algorithmique : Recherche dichotomique

# Recherche dichotomique dans un tableau trié

# Recherche dichotomique dans un tableau trié

Le tableau `T` contient-il `x` ? À quelle position ?

# Introduction : recherche par balayage

## Introduction : recherche par balayage

Déja abordé lors des *parcours séquentiels*

## Introduction : recherche par balayage

Déja abordé lors des *parcours séquentiels*

```
x = 5
T = [11,  7,  9,  5, 15, 13,  3,  1]
      0   1   2   3   4   5   6   7
```

# Introduction : recherche par balayage

Déja abordé lors des *parcours séquentiels*

```
x = 5
T = [11,  7,  9,  5, 15, 13,  3,  1]
      0   1   2   3   4   5   6   7

def recherche(T, x):
    Pour i allant de 0 à len(T) - 1:
        Si x == T[i]:
            renvoyer i
    renvoyer -1
```

# Déroulé - premier tour

```
x = 5
T = [11,  7,  9,  5, 15, 13,  3,  1]
      0   1   2   3   4   5   6   7

def recherche(T, x):                           Déroulé
    Pour i allant de 0 à len(T) - 1:       | i = 0   5 != 11
        Si x == T[i]:                      |
            renvoyer i                     |
    renvoyer -1                            |
```

# Déroulé - second tour

```
x = 5
T = [11,  7,  9,  5, 15, 13,  3,  1]
      0   1   2   3   4   5   6   7

def recherche(T, x):                              Déroulé
    Pour i allant de 0 à len(T) - 1:      | i = 0   5 != 11
        Si x == T[i]:                     | i = 1   5 != 7
            renvoyer i                    |
    renvoyer -1                           |
```

# Déroulé - troisième tour

```
x = 5
T = [11,  7,  9,  5, 15, 13,  3,  1]
      0   1   2   3   4   5   6   7

def recherche(T, x):                            Déroulé
    Pour i allant de 0 à len(T) - 1:       | i = 0   5 != 11
        Si x == T[i]:                      | i = 1   5 != 7
            renvoyer i                     | i = 2   5 != 9
    renvoyer -1                            |
```

# Déroulé - quatrième tour

```
x = 5
T = [11,  7,  9,  5, 15, 13,  3,  1]
      0   1   2   3   4   5   6   7

def recherche(T, x):                          Déroulé
    Pour i allant de 0 à len(T) - 1:      | i = 0   5 != 11
        Si x == T[i]:                     | i = 1   5 != 7
            renvoyer i                    | i = 2   5 != 9
    renvoyer -1                           | i = 3   5 == 5
```

# Remarques

- Termine toujours (boucle bornée. . . )

# Remarques

- Termine toujours (boucle bornée...)
- **Au pire `len(T)` étapes**

# Recherche dichotomique

# Présentation

On suppose maintenant que le tableau `T` est **trié** par ordre croissant

# Présentation

On suppose maintenant que le tableau `T` est **trié** par ordre croissant

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
      0   1   2   3   4   5   6   7
```

# Présentation

On suppose maintenant que le tableau `T` est **trié** par ordre croissant

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
      0   1   2   3   4   5   6   7
```

**Rappel du Jeu de "+ ou -"** : viser le centre des éléments restant

# Présentation

On suppose maintenant que le tableau `T` est **trié** par ordre croissant

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
      0   1   2   3   4   5   6   7
```

**Rappel du Jeu de “+ ou -”** : viser le centre des éléments restant

Comparer la valeur centrale à `x` et **éliminer la moitié des valeurs restantes**

# Déroulé

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
      g           ^               d
```

# Déroulé

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
      g           ^               d

g = 0, d = 7
```

# Déroulé

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
      g           ^              d

g = 0, d = 7

m = (g + d) // 2 = (0 + 7) // 2 = 3
    x = 5 < T[3] = 7    =>   Chercher à gauche
```

# Déroulé

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
      g           ^               d

g = 0, d = 7

m = (g + d) // 2 = (0 + 7) // 2 = 3
    x = 5 < T[3] = 7    =>   Chercher à gauche
```

On recommence avec

- `d = m - 1 = 2`
- g inchangé

# Déroulé

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
      g       d

g = 0, d = 2
```

# Déroulé

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
      g       d

g = 0, d = 2

m = (g + d) // 2 = (0 + 2) // 2 = 1
    x = 5 > T[1] = 3    =>   Chercher à droite
```

# Déroulé

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
      g       d

g = 0, d = 2

m = (g + d) // 2 = (0 + 2) // 2 = 1
    x = 5 > T[1] = 3    =>   Chercher à droite
```

On recommence avec

- `d` inchangé
- `g = m + 1 = 2`

# Déroulé

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
         g=d
g = 2, d = 2
```

# Déroulé

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
          g=d

g = 2, d = 2

m = (g + d) // 2 = (2 + 2) // 2 = 2
    x = 5 == T[2] = 5     =>   Trouvé !
```

## Déroulé

```
x = 5
T = [ 1,  3,  5,  7,  9, 11, 13, 15]
          g=d

g = 2, d = 2

m = (g + d) // 2 = (2 + 2) // 2 = 2
    x = 5 == T[2] = 5    =>   Trouvé !
```

On peut renvoyer 2.

# Construction de l'algorithme

# Construction de l'algorithme

| | |
|---|---|
| **Précondition** | `T` un tableau trié par ordre croissant |

# Construction de l'algorithme

| | |
|---|---|
| **Précondition** | `T` un tableau trié par ordre croissant |
| **Plusieurs étapes** | Il faut une boucle |

# Construction de l'algorithme

| | |
|---|---|
| **Précondition** | `T` un tableau trié par ordre croissant |
| **Plusieurs étapes** | Il faut une boucle |
| **Nombre inconnu d'étapes** | Boucle non bornée (`while`) |

# Construction de l'algorithme

| | |
|---|---|
| **Précondition** | `T` un tableau trié par ordre croissant |
| **Plusieurs étapes** | Il faut une boucle |
| **Nombre inconnu d'étapes** | Boucle non bornée (`while`) |
| **Arrêt** | `g > d` $\Rightarrow$ `while g <= d:` |

# Corps de la boucle

# Corps de la boucle

- `m = (g + d) // 2`

## Corps de la boucle

- `m = (g + d) // 2`
- 3 cas :
  - Si `x == T[m]` $\Rightarrow$ `return m`

## Corps de la boucle

- `m = (g + d) // 2`
- 3 cas :
  - Si `x == T[m]` $\Rightarrow$ `return m`
  - Si `x < m` $\Rightarrow$ `d = m - 1`

# Corps de la boucle

- `m = (g + d) // 2`
- 3 cas :
  - Si `x == T[m]` $\Rightarrow$ `return m`
  - Si `x < m` $\Rightarrow$ `d = m - 1`
  - Si `x > m` $\Rightarrow$ `d = m + 1`

## Corps de la boucle

- `m = (g + d) // 2`
- 3 cas :
  - Si `x == T[m]` $\Rightarrow$ `return m`
  - Si `x < m` $\Rightarrow$ `d = m - 1`
  - Si `x > m` $\Rightarrow$ `d = m + 1`
- **Et si la boucle termine ?**

# Corps de la boucle

- `m = (g + d) // 2`
- 3 cas :
  - Si `x == T[m]` $\Rightarrow$ `return m`
  - Si `x < m` $\Rightarrow$ `d = m - 1`
  - Si `x > m` $\Rightarrow$ `d = m + 1`
- **Et si la boucle termine ?**
  - T ne contient pas `x` $\Rightarrow$ `return -1`

# Algorithme

# Algorithme

```python
def recherche_dichotomique(T, x):
    g = 0
    d = len(T) - 1
    while g <= d:
        m = (g + d) // 2
        if x == T[m]:
            return m
        elif x > T[m]:
            g = m + 1
        else:
            d = m - 1
    return -1
```

# Déroulé de l’algorithme

# Déroulé Premier tour

```
T = [ 1,  3,  5,  7, 11, 13, 15]
      g                       d
def recherche_dichotomique(T, x):
    g = 0                    | 1. g = 0 <= d = 7
    d = len(T) - 1           |
    while g <= d:            |
        m = (g + d) // 2     |
        if x == T[m]:        |
            return m         |
        elif x > T[m]:       |
            g = m + 1        |
        else:                |
            d = m - 1        |
    return -1                |
```

# Déroulé Premier tour

```
T = [ 1,  3,  5,  7, 11, 13, 15]
      g           ^           d
def recherche_dichotomique(T, x):
    g = 0                       | 1. g = 0 <= d = 7
    d = len(T) - 1              |    m = (0 + 7) // 2 = 3
    while g <= d:               |
        m = (g + d) // 2        |
        if x == T[m]:           |
            return m            |
        elif x > T[m]:          |
            g = m + 1           |
        else:                   |
            d = m - 1           |
    return -1                   |
```

# Déroulé Premier tour

```
T = [ 1,  3,  5,  7, 11, 13, 15]
      g        ^           d
def recherche_dichotomique(T, x):
    g = 0                  | 1. g = 0 <= d = 7
    d = len(T) - 1         |    m = (0 + 7) // 2 = 3
    while g <= d:          |   5 < 7  => d = 3 - 1 = 2
        m = (g + d) // 2   |
        if x == T[m]:      |
            return m       |
        elif x > T[m]:     |
            g = m + 1      |
        else:              |
            d = m - 1      |
    return -1              |
```

## Déroulé Second tour

```
T = [ 1,  3,  5,  7, 11, 13, 15]
      g       d
def recherche_dichotomique(T, x):
    g = 0                      | 1. g = 0 <= d = 7
    d = len(T) - 1             |    m = (0 + 7) // 2 = 3
    while g <= d:              |   5 < 7  => d = 3 - 1 = 2
        m = (g + d) // 2       |
        if x == T[m]:          | 2. g = 0 <= d = 2
            return m           |
        elif x > T[m]:         |
            g = m + 1          |
        else:                  |
            d = m - 1          |
    return -1                  |
```

# Déroulé Second tour

```
T = [ 1,  3,  5,  7, 11, 13, 15]
      g   ^   d
def recherche_dichotomique(T, x):
    g = 0                      | 1. g = 0 <= d = 7
    d = len(T) - 1             |    m = (0 + 7) // 2 = 3
    while g <= d:              |   5 < 7  => d = 3 - 1 = 2
        m = (g + d) // 2       |
        if x == T[m]:          | 2. g = 0 <= d = 2
            return m           |    m = (0 + 2) // 2 = 1
        elif x > T[m]:         |
            g = m + 1          |
        else:                  |
            d = m - 1          |
    return -1                  |
```

# Déroulé Second tour

```
T = [ 1,  3,  5,  7, 11, 13, 15]
      g   ^   d
def recherche_dichotomique(T, x):
    g = 0                       | 1. g = 0 <= d = 7
    d = len(T) - 1              |    m = (0 + 7) // 2 = 3
    while g <= d:               |   5 < 7  => d = 3 - 1 = 2
        m = (g + d) // 2        |
        if x == T[m]:           | 2. g = 0 <= d = 2
            return m            |    m = (0 + 2) // 2 = 1
        elif x > T[m]:          |   5 > 3  => g = 1 + 1 = 2
            g = m + 1           |
        else:                   |
            d = m - 1           |
    return -1                   |
```

# Déroulé Troisième tour

```
T = [ 1,  3,  5,  7, 11, 13, 15]
            g=d
def recherche_dichotomique(T, x):
    g = 0                       | 1. g = 0 <= d = 7
    d = len(T) - 1              |    m = (0 + 7) // 2 = 3
    while g <= d:               |   5 < 7  => d = 3 - 1 = 2
        m = (g + d) // 2        |
        if x == T[m]:           | 2. g = 0 <= d = 2
            return m            |    m = (0 + 2) // 2 = 1
        elif x > T[m]:          |   5 > 3  => g = 1 + 1 = 2
            g = m + 1           |
        else:                   | 3. g = 2 <= d = 2
            d = m - 1           |
    return -1                   |
```

# Déroulé Troisième tour

```
T = [ 1,  3,  5,  7, 11, 13, 15]
              ^

def recherche_dichotomique(T, x):
    g = 0                       | 1. g = 0 <= d = 7
    d = len(T) - 1              |    m = (0 + 7) // 2 = 3
    while g <= d:               |   5 < 7  => d = 3 - 1 = 2
        m = (g + d) // 2        |
        if x == T[m]:           | 2. g = 0 <= d = 2
            return m            |    m = (0 + 2) // 2 = 1
        elif x > T[m]:          |   5 > 3  => g = 1 + 1 = 2
            g = m + 1           |
        else:                   | 3. g = 2 <= d = 2
            d = m - 1           |    m = (2 + 2) // 2 = 2
    return -1                   |
```

# Déroulé Troisième tour

```
T = [ 1,  3,  5,  7, 11, 13, 15]
              ^

def recherche_dichotomique(T, x):
    g = 0                       | 1. g = 0 <= d = 7
    d = len(T) - 1              |    m = (0 + 7) // 2 = 3
    while g <= d:               |   5 < 7  => d = 3 - 1 = 2
        m = (g + d) // 2        |
        if x == T[m]:           | 2. g = 0 <= d = 2
            return m            |    m = (0 + 2) // 2 = 1
        elif x > T[m]:          |   5 > 3  => g = 1 + 1 = 2
            g = m + 1           |
        else:                   | 3. g = 2 <= d = 2
            d = m - 1           |    m = (2 + 2) // 2 = 2
    return -1                   |   5 == 5  => return 2
```

# Remarques

# Remarques

## Précondition

# Remarques

**Précondition**

`T` **doit être trié par ordre croissant**

# Remarques

**Précondition**

`T` **doit être trié par ordre croissant**

**Terminaison**

# Remarques

**Précondition**

`T` **doit être trié par ordre croissant**

**Terminaison**

- `while d <= g:`

# Remarques

**Précondition**

`T` **doit être trié par ordre croissant**

**Terminaison**

- `while d <= g:`
- `d - g` strictement décroissant

# Remarques

**Précondition**

`T` **doit être trié par ordre croissant**

**Terminaison**

- `while d <= g:`
- `d - g` strictement décroissant
- En nombre fini d'étapes, `d > g` et l'algo s'arrête toujours

**Coût**

# Remarques

**Précondition**

`T` **doit être trié par ordre croissant**

**Terminaison**

- `while d <= g:`
- `d - g` strictement décroissant
- En nombre fini d'étapes, `d > g` et l'algo s'arrête toujours

**Coût**

- `d - g` est *grossièrement* divisé par 2 à chaque étape

# Remarques

**Précondition**

`T` **doit être trié par ordre croissant**

**Terminaison**

- `while d <= g:`
- `d - g` strictement décroissant
- En nombre fini d'étapes, `d > g` et l'algo s'arrête toujours

**Coût**

- `d - g` est *grossièrement* divisé par 2 à chaque étape
- Ex. Si `len(T)` $= 16 = 2^4$, il faut ~**4 étapes**.

## Conclusion

- La recherche dichotomique permet de gagner beaucoup d'étapes par rapport au parcours séquentiel du tableau.
- Elle nécessite d'avoir un tableau **trié** sans quoi on ne peut l'appliquer.

# Remarques sur le coût

- Si on ne souhaite l'appliquer qu'une seule fois, il n'est pas intéressant de trier le tableau pour chercher. C'est généralement trop long...
- Mais si on doit souvent effectuer des recherches dans le tableau, alors c'est indispensable.

# Nombre d'étapes

- **parcours séquentiel** : autant que d'éléments dans le tableau dans le pire des cas.

  Le parcours séquentiel prend (dans le pire des cas) $n$ étapes.

- **recherche dichotomique** : $\log_2 n$ étapes.

  $\log_2 n \approx$ le nombre de divisions entières de $n$ par 2 qu'on peut effectuer avant de trouver un quotient nul

  $\log_2 n \approx$ le nombre de bits de $n$ en binaire.