CoursMagistral_3

October 18, 2019

1 Table of Contents

1 ALGO1 : Introduction à l'algorithmique
2 Cours Magistral 3
2.1 Quelques algorithmes "diviser pour régner" classiques
2.1.1 Recherche dans un tableau trié en (log())O(log(n))
2.1.2 Recherche dans une liste triée en ()O(n)
2.1.3 Tri fusion (merge sort)
2.1.4 Enveloppe convexe de points en 2D (quickhull)
2.1.4.1 Un exemple
2.1.4.2 Exemples aléatoires de taille contrôlée
2.1.4.3 Complexité temporelle de ce calcul d'enveloppe convexe
2.2 Algorithme de Gauss-Karatsuba
2.3 Algorithme de Strassen
2.3.1 Méthode naïve, "méthode i k j"
2.3.2 Méthode naïve récursive
2.3.3 Méthode récursive de Strassen
2.4 Transformée de Fourier Rapide (FFT)
2.4.1 Transformée de Fourier Discrète (DFT), implémentation naïve
2.4.2 Implémentation de FFT dans le module numpy
2.4.3 Implémentation de la DFT par multiplication matricielle
2.4.4 Implémentation manuelle de la FFT (Cooley-Tucker)
2.4.5 Tests avec des vecteurs aléatoires
2.5 Conclusion

2 ALGO1 : Introduction à l'algorithmique

- Page du cours : https://perso.crans.org/besson/teach/info1_algo1_2019/
- Magistère d'Informatique de Rennes - ENS Rennes - Année 2019/2020
- Intervenants :
  - Cours : Lilian Besson
  - Travaux dirigés : Raphaël Truffet
- Références :
  - Open Data Structures

3 Cours Magistral 3

- Ce cours traite du paradigme "Diviser pour Régner",
- Un important résultat théorique est donné (et prouvé) :

- Ce notebook commence par donner quelques algorithmes "Diviser pour Régner" assez classiques, et permettent d'illustrer les deux premiers cas du "master theorem" (recherche dans une liste triée ou un tableau trié, tri fusion,

- Ce théorème ne suffit pas à couvrir tous les différents algorithmes "Diviser pour Régner", avec comme contre exemple l'algorithme de tri rapide,

- Puis on implémente les deux algorithmes présentés dans le cours, la multiplication de grands entiers par l'algorithme de Gauss-Karatsuba, et la multiplication de grandes matrices par l'algorithme de Strassen (StraSSen).

3.1 Quelques algorithmes "diviser pour régner" classiques

3.1.1 Recherche dans un tableau trié en $\mathcal{O}(\log(n))$

Si un tableau `T = [a1, ..., an]` est trié, et qu'on donne une valeur `x`, on cherche un indice `i` tel que `T[i] = ai = x`, s'il existe (sans spécification s'il n'est pas unique), et une erreur si `x` n'est pas présent dans le tableau.

En utilisant des indices `left` et `right`, qu'on fait augmenter ou diminuer, on évite de faire des recopies du tableau.

La complexité de cet algorithme est en $\mathcal{O}(\log(n))$.

Avec le *master theorem*, on a $a = 1, b = 2, k = 0$: on divise les entrées en $a = 2$ entrées de tailles $\leq b = 2$ plus petites, sur lesquels on applique un traitement **constant** ($\mathcal{O}(n^{k=0})$ avant l'appel récursif (**aucune recopie, juste des changements d'indices !**).

```
In [643]: def dichotomy_in_array(array, value, left=0, right=None):
              if depth > len(array):     raise KeyError
              if right is None:
                  right = len(array) - 1
              n = right - left + 1
              if n == 0:                 raise KeyError
              elif n == 1:
                  if array[left] == value:
                      return left
                  else:                  raise KeyError
              index_of_middle = left + (n // 2)
              middle_of_list = array[index_of_middle]
              if value < middle_of_list:  # search on left
                  return dichotomy_in_array(array, value, left=left, right=index_of_middle - 1)
              elif value > middle_of_list:  # search on right
                  return dichotomy_in_array(array, value, left=index_of_middle + 1, right=righ
              else:
                  return index_of_middle
```

Et avec juste un peu d'affichage pour (un rappel ou pour) comprendre le fonctionnement :

```
In [206]: def dichotomy_in_array(array, value, left=0, right=None, debug=False, depth=0):
              if depth > len(array):
                  raise KeyError
              if right is None:
                  right = len(array) - 1
              n = right - left + 1
              if debug: print(f"    {'  '*(depth+1)}Searching for {value} in sequence of size
              if n == 0:
                  raise KeyError
              elif n == 1:
                  if array[left] == value:
                      return left
                  else:
                      raise KeyError
              index_of_middle = left + (n // 2)
              middle_of_list = array[index_of_middle]
              #if debug: print(f"    {'  '*(depth+1)}{value} >/</=? {middle_of_list}")
              if value < middle_of_list:  # search on left
                  if debug: print(f"    {'  '*(depth+1)}-> going left...")
                  return dichotomy_in_array(array, value, left=left, right=index_of_middle - 1
              elif value > middle_of_list:  # search on right
                  if debug: print(f"    {'  '*(depth+1)}-> going right...")
                  return dichotomy_in_array(array, value, left=index_of_middle + 1, right=righ
              elif value == middle_of_list:
                  if debug: print(f"    {'  '*(depth+1)}-> found at index {index_of_middle} !"
                  return index_of_middle
              else:
                  raise KeyError
```

Faisons quelques essais :

```
In [85]: n = 16
         example_of_array = list(range(n))

In [86]: for value in example_of_array:
             print(f"\n    Looking for value {value} in {example_of_array} :")
             dichotomy_in_array(example_of_array, value, debug=True)
```

```
    Looking for value 0 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
      Searching for 0 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 
      -> going left...
        Searching for 0 in sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
        -> going left...
          Searching for 0 in sequence of size 4 = [0, 1, 2, 3]
          -> going left...
            Searching for 0 in sequence of size 2 = [0, 1]
```

```
        -> going left...
          Searching for 0 in sequence of size 1 = [0]
```

Out[86]: 0

```
Looking for value 1 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Searching for 1 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1
  -> going left...
    Searching for 1 in sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
    -> going left...
      Searching for 1 in sequence of size 4 = [0, 1, 2, 3]
      -> going left...
        Searching for 1 in sequence of size 2 = [0, 1]
        -> found at index 1 !
```

Out[86]: 1

```
Looking for value 2 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Searching for 2 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1
  -> going left...
    Searching for 2 in sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
    -> going left...
      Searching for 2 in sequence of size 4 = [0, 1, 2, 3]
      -> found at index 2 !
```

Out[86]: 2

```
Looking for value 3 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Searching for 3 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1
  -> going left...
    Searching for 3 in sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
    -> going left...
      Searching for 3 in sequence of size 4 = [0, 1, 2, 3]
      -> going right...
        Searching for 3 in sequence of size 1 = [3]
```

Out[86]: 3

```
Looking for value 4 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Searching for 4 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1
  -> going left...
```

```
      Searching for 4 in sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
      -> found at index 4 !
```

Out[86]: 4

```
  Looking for value 5 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
    Searching for 5 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1
    -> going left...
      Searching for 5 in sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
      -> going right...
        Searching for 5 in sequence of size 3 = [5, 6, 7]
        -> going left...
          Searching for 5 in sequence of size 1 = [5]
```

Out[86]: 5

```
  Looking for value 6 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
    Searching for 6 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1
    -> going left...
      Searching for 6 in sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
      -> going right...
        Searching for 6 in sequence of size 3 = [5, 6, 7]
        -> found at index 6 !
```

Out[86]: 6

```
  Looking for value 7 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
    Searching for 7 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1
    -> going left...
      Searching for 7 in sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
      -> going right...
        Searching for 7 in sequence of size 3 = [5, 6, 7]
        -> going right...
          Searching for 7 in sequence of size 1 = [7]
```

Out[86]: 7

```
  Looking for value 8 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
    Searching for 8 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1
    -> found at index 8 !
```

Out[86]: 8

```
Looking for value 9 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Searching for 9 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1
  -> going right...
    Searching for 9 in sequence of size 7 = [9, 10, 11, 12, 13, 14, 15]
    -> going left...
      Searching for 9 in sequence of size 3 = [9, 10, 11]
      -> going left...
        Searching for 9 in sequence of size 1 = [9]
```

Out[86]: 9

```
Looking for value 10 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Searching for 10 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
  -> going right...
    Searching for 10 in sequence of size 7 = [9, 10, 11, 12, 13, 14, 15]
    -> going left...
      Searching for 10 in sequence of size 3 = [9, 10, 11]
      -> found at index 10 !
```

Out[86]: 10

```
Looking for value 11 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Searching for 11 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
  -> going right...
    Searching for 11 in sequence of size 7 = [9, 10, 11, 12, 13, 14, 15]
    -> going left...
      Searching for 11 in sequence of size 3 = [9, 10, 11]
      -> going right...
        Searching for 11 in sequence of size 1 = [11]
```

Out[86]: 11

```
Looking for value 12 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Searching for 12 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
  -> going right...
    Searching for 12 in sequence of size 7 = [9, 10, 11, 12, 13, 14, 15]
    -> found at index 12 !
```

Out[86]: 12

```
Looking for value 13 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Searching for 13 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
  -> going right...
    Searching for 13 in sequence of size 7 = [9, 10, 11, 12, 13, 14, 15]
    -> going right...
      Searching for 13 in sequence of size 3 = [13, 14, 15]
      -> going left...
        Searching for 13 in sequence of size 1 = [13]
```

Out[86]: 13

```
Looking for value 14 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Searching for 14 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
  -> going right...
    Searching for 14 in sequence of size 7 = [9, 10, 11, 12, 13, 14, 15]
    -> going right...
      Searching for 14 in sequence of size 3 = [13, 14, 15]
      -> found at index 14 !
```

Out[86]: 14

```
Looking for value 15 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Searching for 15 in sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
  -> going right...
    Searching for 15 in sequence of size 7 = [9, 10, 11, 12, 13, 14, 15]
    -> going right...
      Searching for 15 in sequence of size 3 = [13, 14, 15]
      -> going right...
        Searching for 15 in sequence of size 1 = [15]
```

Out[86]: 15

Et maintenant des essais sur des entrées de tailles croissantes :

In [119]:
```
import random

def random_sorted_sequence(n, minint=0, maxint=1000):
    sequence = [random.randint(minint, maxint) for _ in range(n)]
    return sorted(sequence)
```

In [151]:
```
def test_dichotomy_in_array(n, debug=False, array=None):
    if array is None:
        array = random_sorted_sequence(n)
    value = random.choice(array)
    return dichotomy_in_array(array, value, debug=debug)
```

```
In [152]: test_dichotomy_in_array(16, debug=True)

      Searching for 324 in sequence of size 16 = [31, 73, 120, 137, 165, 188, 223, 226, 324, 35
      -> found at index 8 !


Out[152]: 8

In [153]: import sys
          sys.setrecursionlimit(10000)

In [154]: T = random_sorted_sequence(100)
          %timeit test_dichotomy_in_array(100, array=T)

3.75 ţs ś 759 ns per loop (mean ś std. dev. of 7 runs, 100000 loops each)


In [155]: T = random_sorted_sequence(1000)
          %timeit test_dichotomy_in_array(1000, array=T)

4.84 ţs ś 610 ns per loop (mean ś std. dev. of 7 runs, 100000 loops each)


In [156]: T = random_sorted_sequence(10000)
          %timeit test_dichotomy_in_array(10000, array=T)

5.09 ţs ś 131 ns per loop (mean ś std. dev. of 7 runs, 100000 loops each)


In [157]: T = random_sorted_sequence(100000)
          %timeit test_dichotomy_in_array(100000, array=T)

5.36 ţs ś 417 ns per loop (mean ś std. dev. of 7 runs, 100000 loops each)


In [158]: T = random_sorted_sequence(1000000)
          %timeit test_dichotomy_in_array(1000000, array=T)

5.55 ţs ś 280 ns per loop (mean ś std. dev. of 7 runs, 100000 loops each)
```

La complexité semble bien logarithmique en n (ie., $\mathcal{O}(\log(n))$) !

3.1.2 Recherche dans une liste triée en $\mathcal{O}(n)$

C'est comme la recherche dans un tableau trié, sauf qu'on recopie la liste de gauche ou de droite lors des appels récursifs.

La complexité de cet algorithme est en $\mathcal{O}(n)$.

Avec le *master theorem*, on a $a = 1, b = 2, k = 1$: on divise les entrées en $a = 2$ entrées de tailles $\leq b = 2$ plus petites, sur lesquels on applique un traitement linéaire ($\mathcal{O}(n^{k=1})$ avant l'appel récursif (**à cause des recopies !**).

```
In [207]: def dichotomy_in_list(sequence, value):
              n = len(sequence)
              if n == 0:             raise KeyError
              index_of_middle = n // 2
              middle_of_list = sequence[index_of_middle]
              if value < middle_of_list:  # search on left
                  # creating this list takes O(n/2) time
                  left_list = sequence[:index_of_middle]
                  return dichotomy_in_list(left_list, value, debug=debug, depth=depth+1)
              elif value > middle_of_list:  # search on right
                  # creating this list takes O(n/2) time
                  right_list = sequence[index_of_middle:]
                  return index_of_middle + dichotomy_in_list(right_list, value, debug=debug, de
              else:
                  return index_of_middle
```

Et avec juste un peu d'affichage pour (un rappel ou pour) comprendre le fonctionnement :

```
In [131]: def dichotomy_in_list(sequence, value, debug=False, depth=0):
              n = len(sequence)
              if debug:
                  print(f"    {'  '*(depth+1)}Sequence of size {n} = {sequence}")
              if n == 0:
                  raise KeyError
              index_of_middle = n // 2
              middle_of_list = sequence[index_of_middle]
              if value < middle_of_list:  # search on left
                  left_list = sequence[:index_of_middle]
                  if debug: print(f"    {'  '*(depth+1)}-> going left, in {left_list} of size 
                  return dichotomy_in_list(left_list, value, debug=debug, depth=depth+1)
              elif value > middle_of_list:  # search on right
                  right_list = sequence[index_of_middle:]
                  if debug: print(f"    {'  '*(depth+1)}-> going right, in {right_list} of siz
                  return index_of_middle + dichotomy_in_list(right_list, value, debug=debug, de
              elif value == middle_of_list:
                  if debug: print(f"    {'  '*(depth+1)}-> found at index {index_of_middle} !"
                  return index_of_middle
              else:
                  raise KeyError
```

Faisons quelques essais :

```
In [132]: n = 16
          example_of_list = list(range(n))

In [133]: for value in example_of_list:
              print(f"\n    Looking for value {value} in {example_of_list} :")
              dichotomy_in_list(example_of_list, value, debug=True)
```

```
Looking for value 0 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going left, in [0, 1, 2, 3, 4, 5, 6, 7] of size 8...
    Sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
    -> going left, in [0, 1, 2, 3] of size 4...
      Sequence of size 4 = [0, 1, 2, 3]
      -> going left, in [0, 1] of size 2...
        Sequence of size 2 = [0, 1]
        -> going left, in [0] of size 1...
          Sequence of size 1 = [0]
          -> found at index 0 !
```

Out[133]: 0

```
Looking for value 1 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going left, in [0, 1, 2, 3, 4, 5, 6, 7] of size 8...
    Sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
    -> going left, in [0, 1, 2, 3] of size 4...
      Sequence of size 4 = [0, 1, 2, 3]
      -> going left, in [0, 1] of size 2...
        Sequence of size 2 = [0, 1]
        -> found at index 1 !
```

Out[133]: 1

```
Looking for value 2 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going left, in [0, 1, 2, 3, 4, 5, 6, 7] of size 8...
    Sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
    -> going left, in [0, 1, 2, 3] of size 4...
      Sequence of size 4 = [0, 1, 2, 3]
      -> found at index 2 !
```

Out[133]: 2

```
Looking for value 3 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going left, in [0, 1, 2, 3, 4, 5, 6, 7] of size 8...
    Sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
    -> going left, in [0, 1, 2, 3] of size 4...
      Sequence of size 4 = [0, 1, 2, 3]
```

```
      -> going right, in [2, 3] of size 2...
        Sequence of size 2 = [2, 3]
        -> found at index 1 !
```

Out[133]: 3

```
Looking for value 4 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going left, in [0, 1, 2, 3, 4, 5, 6, 7] of size 8...
    Sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
    -> found at index 4 !
```

Out[133]: 4

```
Looking for value 5 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going left, in [0, 1, 2, 3, 4, 5, 6, 7] of size 8...
    Sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
    -> going right, in [4, 5, 6, 7] of size 4...
      Sequence of size 4 = [4, 5, 6, 7]
      -> going left, in [4, 5] of size 2...
        Sequence of size 2 = [4, 5]
        -> found at index 1 !
```

Out[133]: 5

```
Looking for value 6 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going left, in [0, 1, 2, 3, 4, 5, 6, 7] of size 8...
    Sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
    -> going right, in [4, 5, 6, 7] of size 4...
      Sequence of size 4 = [4, 5, 6, 7]
      -> found at index 2 !
```

Out[133]: 6

```
Looking for value 7 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going left, in [0, 1, 2, 3, 4, 5, 6, 7] of size 8...
    Sequence of size 8 = [0, 1, 2, 3, 4, 5, 6, 7]
    -> going right, in [4, 5, 6, 7] of size 4...
```

```
      Sequence of size 4 = [4, 5, 6, 7]
      -> going right, in [6, 7] of size 2...
        Sequence of size 2 = [6, 7]
        -> found at index 1 !
```

Out[133]: 7

```
Looking for value 8 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> found at index 8 !
```

Out[133]: 8

```
Looking for value 9 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going right, in [8, 9, 10, 11, 12, 13, 14, 15] of size 8...
    Sequence of size 8 = [8, 9, 10, 11, 12, 13, 14, 15]
    -> going left, in [8, 9, 10, 11] of size 4...
      Sequence of size 4 = [8, 9, 10, 11]
      -> going left, in [8, 9] of size 2...
        Sequence of size 2 = [8, 9]
        -> found at index 1 !
```

Out[133]: 9

```
Looking for value 10 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going right, in [8, 9, 10, 11, 12, 13, 14, 15] of size 8...
    Sequence of size 8 = [8, 9, 10, 11, 12, 13, 14, 15]
    -> going left, in [8, 9, 10, 11] of size 4...
      Sequence of size 4 = [8, 9, 10, 11]
      -> found at index 2 !
```

Out[133]: 10

```
Looking for value 11 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going right, in [8, 9, 10, 11, 12, 13, 14, 15] of size 8...
    Sequence of size 8 = [8, 9, 10, 11, 12, 13, 14, 15]
    -> going left, in [8, 9, 10, 11] of size 4...
      Sequence of size 4 = [8, 9, 10, 11]
```

```
    -> going right, in [10, 11] of size 2...
      Sequence of size 2 = [10, 11]
      -> found at index 1 !
```

Out[133]: 11

```
Looking for value 12 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going right, in [8, 9, 10, 11, 12, 13, 14, 15] of size 8...
    Sequence of size 8 = [8, 9, 10, 11, 12, 13, 14, 15]
    -> found at index 4 !
```

Out[133]: 12

```
Looking for value 13 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going right, in [8, 9, 10, 11, 12, 13, 14, 15] of size 8...
    Sequence of size 8 = [8, 9, 10, 11, 12, 13, 14, 15]
    -> going right, in [12, 13, 14, 15] of size 4...
      Sequence of size 4 = [12, 13, 14, 15]
      -> going left, in [12, 13] of size 2...
        Sequence of size 2 = [12, 13]
        -> found at index 1 !
```

Out[133]: 13

```
Looking for value 14 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going right, in [8, 9, 10, 11, 12, 13, 14, 15] of size 8...
    Sequence of size 8 = [8, 9, 10, 11, 12, 13, 14, 15]
    -> going right, in [12, 13, 14, 15] of size 4...
      Sequence of size 4 = [12, 13, 14, 15]
      -> found at index 2 !
```

Out[133]: 14

```
Looking for value 15 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] :
  Sequence of size 16 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
  -> going right, in [8, 9, 10, 11, 12, 13, 14, 15] of size 8...
    Sequence of size 8 = [8, 9, 10, 11, 12, 13, 14, 15]
    -> going right, in [12, 13, 14, 15] of size 4...
```

```
Sequence of size 4 = [12, 13, 14, 15]
-> going right, in [14, 15] of size 2...
  Sequence of size 2 = [14, 15]
  -> found at index 1 !
```

```
Out[133]: 15
```

Et maintenant des essais sur des entrées de tailles croissantes :

```
In [134]: import random

          def random_sorted_sequence(n, minint=0, maxint=1000):
              sequence = [random.randint(minint, maxint) for _ in range(n)]
              return sorted(sequence)

In [161]: def test_dichotomy_in_list(n, debug=False, sequence=None):
              if sequence is None:
                  sequence = random_sorted_sequence(n)
              value = random.choice(sequence)
              return dichotomy_in_list(sequence, value, debug=debug)

In [162]: test_dichotomy_in_array(16, debug=True)
```

```
Searching for 217 in sequence of size 16 = [32, 42, 72, 85, 107, 175, 217, 343, 396, 446
-> going left...
  Searching for 217 in sequence of size 8 = [32, 42, 72, 85, 107, 175, 217, 343]
  -> going right...
    Searching for 217 in sequence of size 3 = [175, 217, 343]
    -> found at index 6 !
```

```
Out[162]: 6

In [163]: import sys
          sys.setrecursionlimit(10000)

In [164]: L = random_sorted_sequence(100)
          %timeit test_dichotomy_in_list(100, sequence=L)
```

```
2.13 ţs ś 84 ns per loop (mean ś std. dev. of 7 runs, 100000 loops each)
```

```
In [165]: L = random_sorted_sequence(1000)
          %timeit test_dichotomy_in_list(1000, sequence=L)
```

```
6.73 ţs ś 185 ns per loop (mean ś std. dev. of 7 runs, 100000 loops each)
```

```
In [169]: L = random_sorted_sequence(10000)
          %timeit test_dichotomy_in_list(10000, sequence=L)
```

```
40.3 ţs ś 1.01 ţs per loop (mean ś std. dev. of 7 runs, 10000 loops each)
```

```
In [168]: L = random_sorted_sequence(100000)
          %timeit test_dichotomy_in_list(100000, sequence=L)
```

```
853 ţs ś 89.3 ţs per loop (mean ś std. dev. of 7 runs, 1000 loops each)
```

```
In [170]: L = random_sorted_sequence(1000_000)
          %timeit test_dichotomy_in_list(1000_000, sequence=L)
```

```
34.2 ms ś 2.7 ms per loop (mean ś std. dev. of 7 runs, 10 loops each)
```

```
In [171]: L = random_sorted_sequence(10_000_000)
          %timeit test_dichotomy_in_list(10_000_000, sequence=L)
```

```
436 ms ś 46.6 ms per loop (mean ś std. dev. of 7 runs, 10 loops each)
```

La complexité semble bien linéaire en n (ie., $\mathcal{O}(n)$) !

3.1.3 Tri fusion (*merge sort*)

- Problème :
  - entrée = tableau `T = [a1, ..., an]` de `n` valeurs
  - sortie = tableau trié par ordre croissant
- Algorithme :
  - on divise en deux le tableau, `gauche = [a1, ..., a_n/2]` et `droite = [a_n/2+1, ..., an]`,
  - on trie récursivement les deux sous tableaux,
  - on fusionne (*merge*) les deux sous tableaux en un.

La fusion est simple à réaliser : on commence tout à gauche des deux tableaux, on avance dans le tableau de gauche ou le tableau de droite, séquentiellement, jusqu'à avoir épuiser leurs valeurs, en prenant la valeur de gauche tant qu'elle est plus petite que celle de droite.

La complexité de cet algorithme est en $\mathcal{O}(n \log(n))$.

Avec le *master theorem*, on a $a = 2, b = 2, k = 1$: on divise les entrées en $a = 2$ entrées de tailles $\leq b = 2$ plus petites, sur lesquels on applique un traitement linéaire ($\mathcal{O}(n^{k=1})$ avant et après l'appel récursif.

```
In [197]: def merge(left, right):
              result = []
              left_idx, right_idx = 0, 0  # growing pointers
              while left_idx < len(left) and right_idx < len(right):
                  # this loop terminates because left_idx + right_idx is strictly increasing
                  # and bounded by len(left) + len(right)
```

```
            # change the direction of this comparison to change the direction of the sor
            if left[left_idx] <= right[right_idx]:
                result.append(left[left_idx])
                left_idx += 1
            else:
                result.append(right[right_idx])
                right_idx += 1
        # we still have values to take on the left
        if left_idx < len(left):
            result.extend(left[left_idx:])
        # we still have values to take on the right
        if right_idx < len(right):
            result.extend(right[right_idx:])
        return result
```

```
In [198]: def merge_sort(m):
              if len(m) <= 1:
                  return m

              middle = len(m) // 2
              # separating the array in two pieces is easy with Python
              # but keep in mind, this takes a linear time to copy the arrays!
              left = m[:middle]
              right = m[middle:]

              sorted_left = merge_sort(left)
              sorted_right = merge_sort(right)
              return list(merge(sorted_left, sorted_right))
```

Quelques tests :

```
In [199]: L = random_sorted_sequence(100)
          %timeit merge_sort(shuffle(L))

305 ţs ś 18.2 ţs per loop (mean ś std. dev. of 7 runs, 1000 loops each)
```

```
In [200]: L = random_sorted_sequence(1000)
          %timeit merge_sort(shuffle(L))

4.7 ms ś 733 ţs per loop (mean ś std. dev. of 7 runs, 100 loops each)
```

```
In [201]: L = random_sorted_sequence(10_000)
          %timeit merge_sort(shuffle(L))

61.4 ms ś 9.57 ms per loop (mean ś std. dev. of 7 runs, 10 loops each)
```

```
In [202]: L = random_sorted_sequence(100_000)
          %timeit merge_sort(shuffle(L))
```

```
661 ms ś 41 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)
```

```
In [203]: L = random_sorted_sequence(1000_000)
          %timeit merge_sort(shuffle(L))
```

```
8.27 s ś 213 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)
```

On pourrait vérifier que la complexité est bien en $\mathcal{O}(n \log(n))$.

```
In [433]: import timeit
          try:
              from tqdm import tqdm_notebook as tqdm
          except ImportError:
              def tqdm(iterator, *args, **kwargs):
                  return iterator
```

```
In [357]: values_n = np.array(np.floor(np.logspace(2, 6.5, num=50)), dtype=int)
          values_L = [ random_sorted_sequence(n) for n in values_n ]
```

```
In [358]: values_times = [
              timeit.timeit(
                  stmt=f"merge_sort(shuffle({L}))",
                  number= 10000 if n <= 1000 else (1000 if n <= 10000 else (100 if n <= 100000
                  globals=globals()
              )
              for (n, L) in tqdm(list(zip(values_n, values_L)))
          ]
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
  2%|          | 1/50 [00:03<02:27,  3.01s/it]
  4%|          | 2/50 [00:07<02:42,  3.38s/it]
  6%|          | 3/50 [00:12<03:03,  3.90s/it]
  8%|          | 4/50 [00:18<03:27,  4.51s/it]
 10%|          | 5/50 [00:26<04:06,  5.47s/it]
 12%|         | 6/50 [00:35<04:59,  6.80s/it]
 14%|         | 7/50 [00:48<06:11,  8.63s/it]
 16%|         | 8/50 [01:04<07:36, 10.87s/it]
 18%|         | 9/50 [01:25<09:22, 13.73s/it]
 20%|         | 10/50 [01:53<12:05, 18.15s/it]
 22%|        | 11/50 [02:31<15:39, 24.08s/it]
 24%|        | 12/50 [02:35<11:25, 18.03s/it]
 26%|        | 13/50 [02:40<08:41, 14.10s/it]
 28%|        | 14/50 [02:46<07:02, 11.73s/it]
```

```
 30%|      | 15/50 [02:54<06:08, 10.54s/it]
 32%|      | 16/50 [03:05<06:01, 10.63s/it]
 34%|      | 17/50 [03:18<06:15, 11.38s/it]
 36%|      | 18/50 [03:35<06:55, 12.99s/it]
 38%|      | 19/50 [03:56<08:00, 15.49s/it]
 40%|      | 20/50 [04:22<09:22, 18.76s/it]
 42%|     | 21/50 [04:56<11:13, 23.23s/it]
 44%|     | 22/50 [05:38<13:29, 28.91s/it]
 46%|     | 23/50 [05:44<09:51, 21.89s/it]
 48%|     | 24/50 [05:51<07:35, 17.53s/it]
 50%|     | 25/50 [05:59<06:05, 14.62s/it]
 52%|    | 26/50 [06:09<05:19, 13.32s/it]
 54%|    | 27/50 [06:22<05:04, 13.26s/it]
 56%|    | 28/50 [06:43<05:38, 15.39s/it]
 58%|    | 29/50 [07:06<06:12, 17.72s/it]
 60%|    | 30/50 [07:33<06:50, 20.53s/it]
 62%|   | 31/50 [08:10<08:06, 25.59s/it]
 64%|   | 32/50 [08:52<09:06, 30.37s/it]
 66%|   | 33/50 [09:46<10:35, 37.41s/it]
 68%|   | 34/50 [09:53<07:33, 28.33s/it]
 70%|   | 35/50 [10:01<05:35, 22.36s/it]
 72%|  | 36/50 [10:13<04:27, 19.12s/it]
 74%|  | 37/50 [10:26<03:43, 17.21s/it]
 76%|  | 38/50 [10:47<03:40, 18.35s/it]
 78%|  | 39/50 [11:09<03:33, 19.42s/it]
 80%|  | 40/50 [11:37<03:41, 22.17s/it]
 82%| | 41/50 [12:13<03:56, 26.31s/it]
 84%| | 42/50 [13:02<04:23, 32.95s/it]
 86%| | 43/50 [14:09<05:03, 43.42s/it]
 88%| | 44/50 [15:23<05:15, 52.60s/it]
 90%| | 45/50 [17:08<05:41, 68.21s/it]
 92%|| 46/50 [19:03<05:28, 82.13s/it]
 94%|| 47/50 [21:16<04:52, 97.63s/it]
 96%|| 48/50 [24:13<04:02, 121.18s/it]
 98%|| 49/50 [27:56<02:31, 151.72s/it]
100%|| 50/50 [32:35<00:00, 190.13s/it]
```

```
In [363]: import numpy as np

          import matplotlib as mpl
          mpl.rcParams['figure.figsize'] = (8, 5)
          mpl.rcParams['figure.dpi'] = 120

          import matplotlib.pyplot as plt

          import seaborn as sns
          sns.set(context="notebook", style="whitegrid", palette="hls", font="sans-serif", fon
```

```
In [364]: plt.figure()
```

```
plt.xlabel("Size of the input array $n$")
plt.ylabel("Time in second")
plt.title("Time complexity of the merge sort algorithm (naive code in Python)")
plt.plot(values_n, values_times, "d-")
plt.show()
```

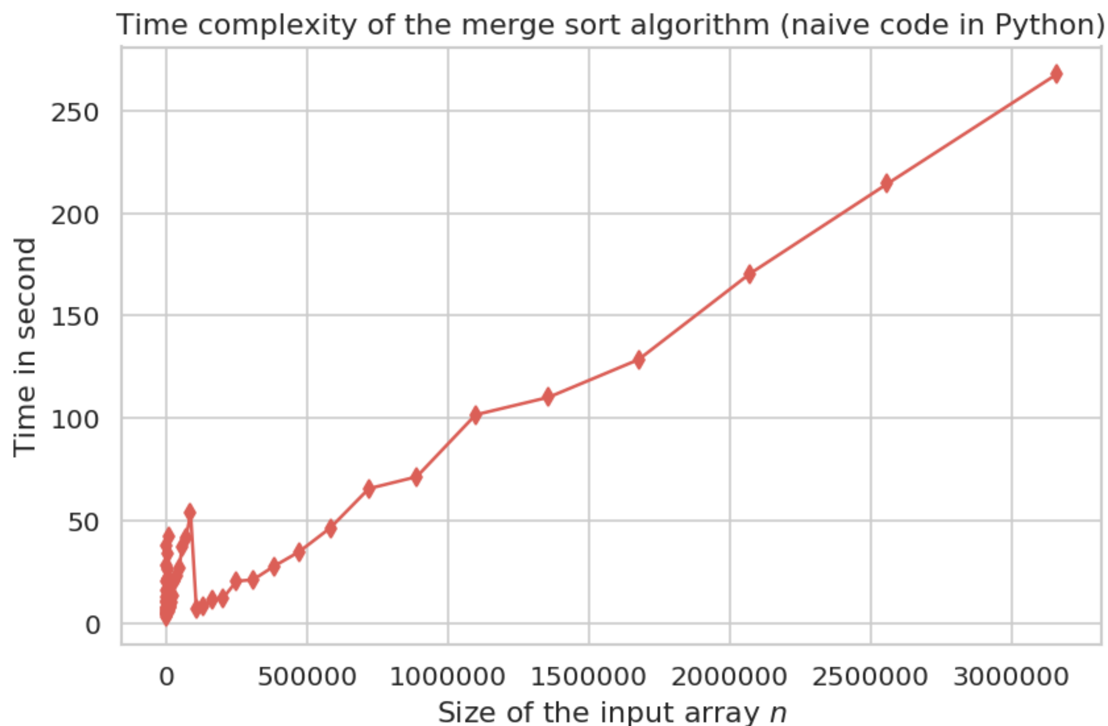
```
Out[364]: <Figure size 960x600 with 0 Axes>

Out[364]: Text(0.5, 0, 'Size of the input array $n$')

Out[364]: Text(0, 0.5, 'Time in second')

Out[364]: Text(0.5, 1.0, 'Time complexity of the merge sort algorithm (naive code in Python)')

Out[364]: [<matplotlib.lines.Line2D at 0x7f8227b8af28>]
```




```
In [368]: plt.figure()
plt.xlabel("Size of the input array $n$")
plt.ylabel("Time in milli-second, normalized by $n \log(n)$")
plt.title("Time complexity of the merge sort algorithm (naive code in Python)")
normalized_values_times = 1e6 * np.array(values_times) / (values_n * np.log(values_n
min_time = 1e5
plt.plot(values_n[values_n >= min_time], normalized_values_times[values_n >= min_tim
plt.show()
```

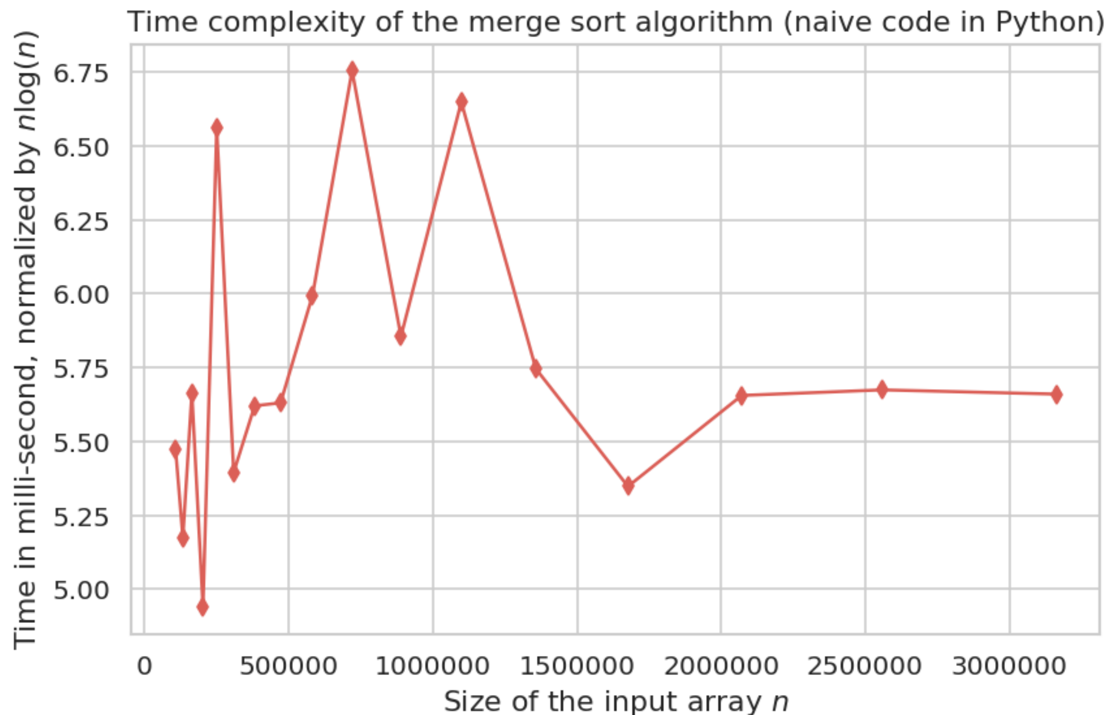
```
Out[368]: <Figure size 960x600 with 0 Axes>

Out[368]: Text(0.5, 0, 'Size of the input array $n$')

Out[368]: Text(0, 0.5, 'Time in milli-second, normalized by $n \\log(n)$')

Out[368]: Text(0.5, 1.0, 'Time complexity of the merge sort algorithm (naive code in Python)')

Out[368]: [<matplotlib.lines.Line2D at 0x7f81fe4877f0>]
```



3.1.4 Enveloppe convexe de points en 2D (*quickhull*)

- Problème :
  - entrée = tableau `T = [xy1, ..., xyn]` de `n` points dans le plan (`xy = (x, y)`)
  - sortie = tableau `hull = [xy_i1, ..., xy_ip]` des `p` points constituant l'enveloppe convexe des `n` points d'entrée
- Algorithme :
  - on identifie le point le plus en bas à gauche P_{bg}, le point le plus en haut à droite P_{hd},
  - la diagonale D est le segment orienté qui va de P_{bg} à P_{hd},
  - on sépare l'ensemble en deux, les points à gauche de D (coin bas droite), ceux à droite de D,
  - on calcule les deux enveloppes convexes E_g et E_d des deux ensembles de points (récursivement),

- on fusionne les deux enveloppes convexes E_g et E_d en les ordonnant comme il faut (et en enlevant l'arête $P_{hd} \to P_{pg}$ qui est présente dans E_g et $P_{pg} \to P_{hd}$ présente dans E_d.

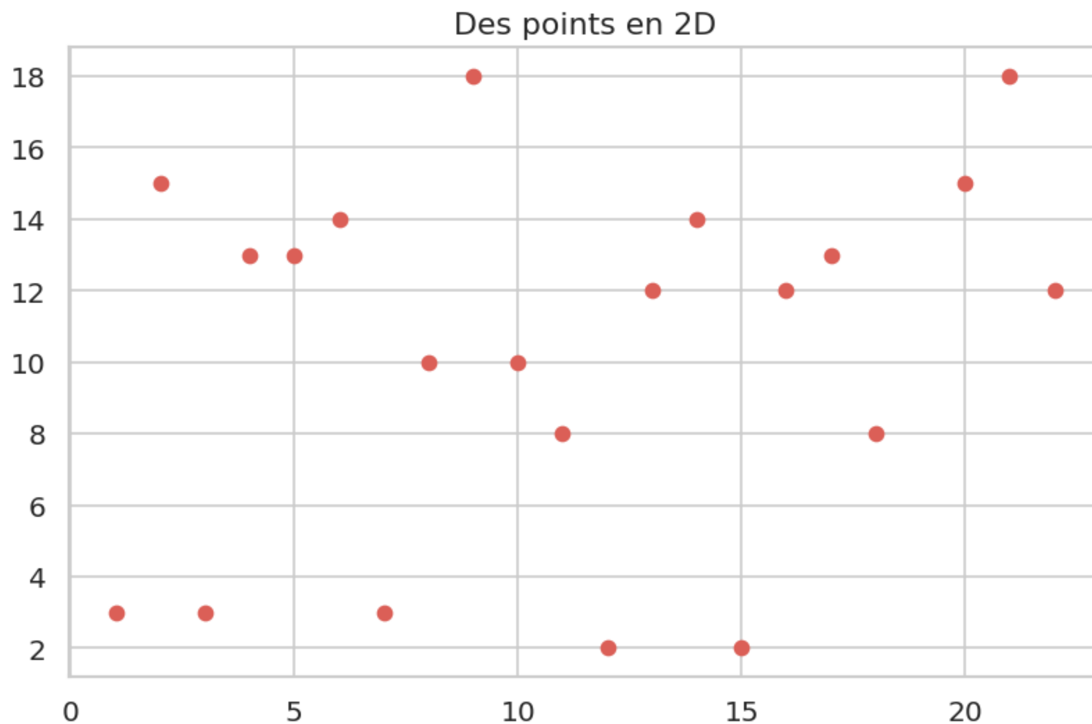
La fusion est naïve simple à réaliser. Le test pour savoir si un point est à gauche, à droite, ou sur D, se fait en temps $O(1)$ avec un calcul de produit scalaire.

```
In [374]: exemple_points = [(1, 3) for _ in range(21)]
          for i in range(1, 21):
              x, y = exemple_points[i-1]
              exemple_points[i] = (x * 17) % 23, (y * 17) % 19
```

```
In [407]: plt.figure()
          plt.title("Des points en 2D")
          Xs = [p[0] for p in exemple_points]
          Ys = [p[1] for p in exemple_points]
          ax = plt.scatter(Xs, Ys)
          plt.show()
```

```
Out[407]: <Figure size 960x600 with 0 Axes>
```

```
Out[407]: Text(0.5, 1.0, 'Des points en 2D')
```




```
In [376]: def alpha(a, b, c):
              xa, ya = a
```

```
        xb, yb = b
        xc, yc = c
        return (xb - xa) * (yc - ya) - (xc - xa) * (yb - ya)
```

In [380]:
```
p = exemple_points
for i in range(2, 21):
    for j in range(1, i):
        for k in range(0, j):
            if alpha(p[i], p[j], p[k]) == 0:
                print(f"Les trois points #{i}, #{j} et #{k} sont alignés ({p[i]}, {p
```

```
Les trois points #5, #2 et #0 sont alignés ((21, 18), (13, 12) et (1, 3)).
Les trois points #11, #8 et #6 sont alignés ((22, 12), (18, 8) et (12, 2)).
Les trois points #12, #6 et #4 sont alignés ((6, 14), (12, 2) et (8, 10)).
Les trois points #16, #15 et #10 sont alignés ((2, 15), (15, 2) et (4, 13)).
Les trois points #17, #3 et #2 sont alignés ((11, 8), (14, 14) et (13, 12)).
Les trois points #18, #9 et #0 sont alignés ((3, 3), (7, 3) et (1, 3)).
Les trois points #18, #13 et #3 sont alignés ((3, 3), (10, 10) et (14, 14)).
Les trois points #19, #10 et #1 sont alignés ((5, 13), (4, 13) et (17, 13)).
Les trois points #20, #9 et #1 sont alignés ((16, 12), (7, 3) et (17, 13)).
Les trois points #20, #11 et #2 sont alignés ((16, 12), (22, 12) et (13, 12)).
```

In [381]:
```
plt.figure()
plt.title("Des points en 2D, certains sont alignés")
Xs = [p[0] for p in exemple_points]
Ys = [p[1] for p in exemple_points]
plt.scatter(Xs, Ys)
p = exemple_points
for i in range(2, 21):
    for j in range(1, i):
        for k in range(0, j):
            if alpha(p[i], p[j], p[k]) == 0:
                plt.plot([p[i][0], p[j][0], p[k][0]], [p[i][1], p[j][1], p[k][1]], '
plt.show()
```

Out[381]: <Figure size 960x600 with 0 Axes>

Out[381]: Text(0.5, 1.0, 'Des points en 2D, certains sont alignés')

Out[381]: <matplotlib.collections.PathCollection at 0x7f81fe18edd8>

Out[381]: [<matplotlib.lines.Line2D at 0x7f81fe304438>]

Out[381]: [<matplotlib.lines.Line2D at 0x7f81fe19c1d0>]

Out[381]: [<matplotlib.lines.Line2D at 0x7f81fe19c550>]

Out[381]: [<matplotlib.lines.Line2D at 0x7f81fe19c8d0>]

Out[381]: [<matplotlib.lines.Line2D at 0x7f81fe19cc50>]

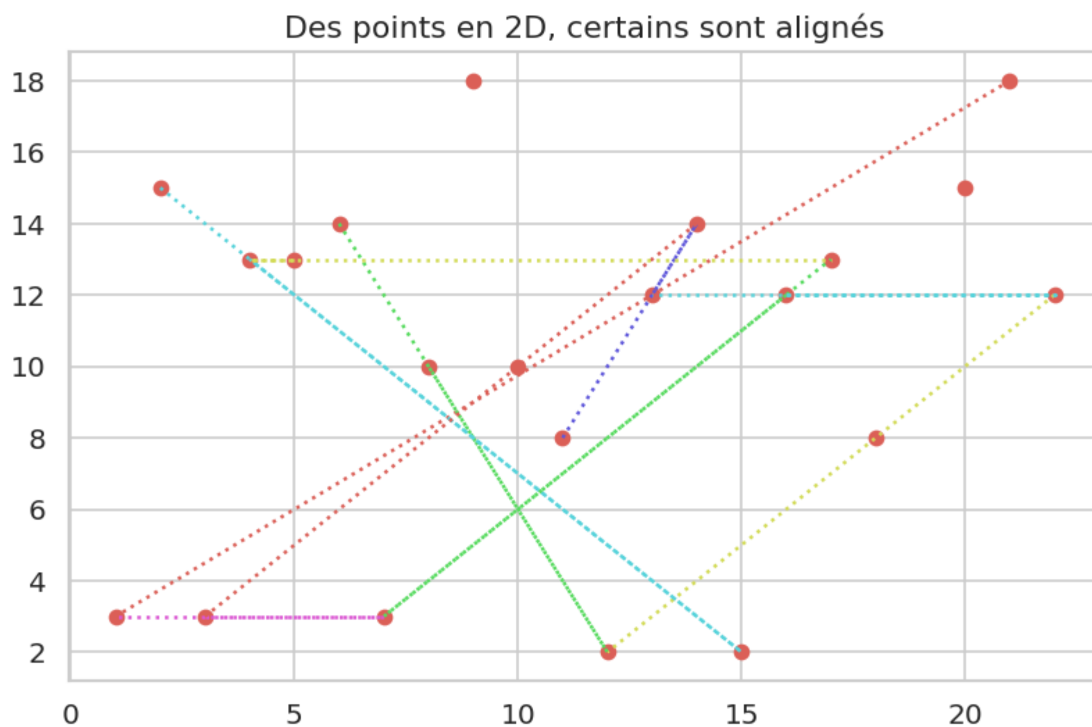
Out[381]: [<matplotlib.lines.Line2D at 0x7f81fe19cfd0>]

Out[381]: [<matplotlib.lines.Line2D at 0x7f81fe24dd68>]

Out[381]: [<matplotlib.lines.Line2D at 0x7f81fe12e6d8>]

Out[381]: [<matplotlib.lines.Line2D at 0x7f81fe12ea58>]

Out[381]: [<matplotlib.lines.Line2D at 0x7f81fe12edd8>]




```
In [391]: def plus_bas(points):
              """ Trouve le point (xa, ya) le plus en bas à gauche, en temps linéaire."""
              n = len(points)
              xa, ya = points[0]
              for j in range(1, n):
                  xj, yj = points[j]
                  if (ya > yj) or (ya == yj and xa > xj):
                      xa, ya = xj, yj
              return xa, ya
```

```
In [392]: plus_bas(exemple_points)  # (12, 2)
```

Out[392]: (12, 2)