Treball de Recerca: Algoritmes d'ordenació

Joan Coma Bages

Curs 2020/2021

Índex

1. Introducció

2. Què és ordenar un vector?

Un vector és un seguit d'elements en un ordre qualsevol. Quan n'ordenem un estem canviant de posició els elements que conté per tal que estiguin ordenats d'una manera determinada.

En aquest treball els elements a ordenar són números, que ordenarem de forma ascendent.

2.1. Què podem ordenar?

No podem ordenar elements sense una referència que ens indiqui quin va primer. Això ho veiem en números i lletres: primer va l'1 i el següeix el 2, comença la A i a continuació la B.

Si volem ordenar hortalisses no existeix cap sistema com amb els números o les lletres i se'ns plantegen dues opcions:

a) inventar-nos un sistema de referència: primer les cols, després les pastanagues, i finalment els espinacs o

b) buscar una propietat comuna en totes les nostres verdures i ordenar-les en funció d'aquesta (alfabèticament pel nom, de menor a major pes, etc.)

Ordenem el que ordenem el procediment és el mateix, i els mètodes aqui utilitzats per ordenar números també són vàlids per ordenar paraules o hortalisses.

2.2. Per què ordenem?

Ordenem números pel mateix motiu que ordenem l'armari, l'habitació o la casa sencera: el temps que dediquem ara a ordenar el recuperem a l'hora de buscar.

Si guardem totes les garanties dels electrodomèstics en una carpeta no patirem (tant) quan se n'espatlli un.

L'índex d'aquest document permet navegar-lo amb facilitat, però no faria servei si les pàgines no estiguéssin ordenades. Si aquesta pàgina, la 5 es trobés entre la 11 i la 3 la numeració dels fulls faria més nosa que servei.

De la mateixa manera agraïm que els diccionaris (en paper) ordenin les paraules alfabèticament, si ho féssin aleatòriament vendrien pocs exemplars.

3. Selection sort

3.1. Com funciona?

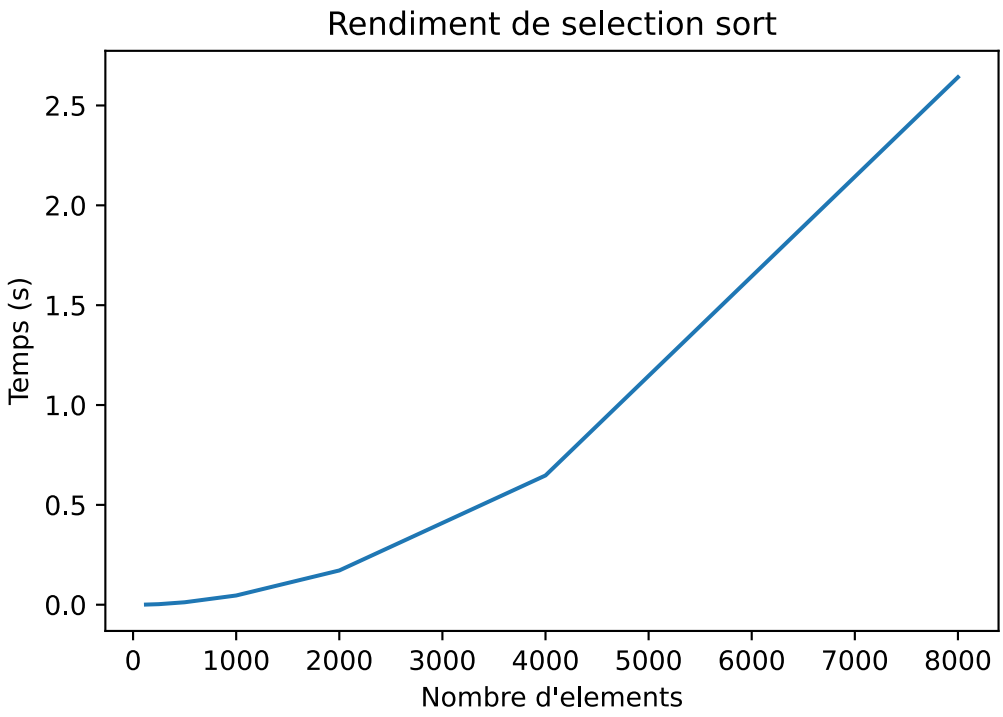
Aquest algoritme itera sobre la llista buscant el nombre més baix, que col·loca al principi. A continuació repeteix el mateix procés buscant el segon nombre més baix, però no des de l'inici sinó des de la segona posició (sabem que en la primera hi trobem el nombre més baix). Aquest procediment es repeteix fins a ordenar tota la llista.

3.2. Pseudocodi

3.3. Implementació

```
#!/usr/bin/env python3
import utils

def sort(array):
    for i in range(len(array[:-1])):
        low = i
        for j in range(i, len(array)):
            if array[low] > array[j]:
                low = j

        array[i], array[low] = array[low], array[i]
    return array

if __name__ == "__main__":
    array = utils.numbers()
    print(sort(array))
```

3.4. Rendiment



4. Insertion sort

4.1. Com funciona?

L'insertion sort itera sobre tots els elements de la llista i els va comparant amb els anteriors de manera que els elements a la dreta de l'actual quedin ordenats.

Pas a pas això vol dir agafar d'entrada un sol element, el primer, que compararem amb els anteriors. Al ser el primer element no n'hi ha d'anteriors, el considerarem ordenat. En la seguent iteració agafem el segon element, aquest el compararem progressivament amb els anteriors. El nostre element, que es troba encara en la segona posició, el comparem amb l'anterior. Si l'anterior és major el nostre passarà davant d'aquest, si el nostre és menor quedarà darrere del primer. En les següents iteracions repetim: anem comparant amb els anteriors fins que trobem un element menor al nostre. Quan el trobem el nostre element el col·locarem a continuació del menor. Si després de comparar-lo amb tota la resta en trobéssim un de menor el deixariem al principi de la llista.

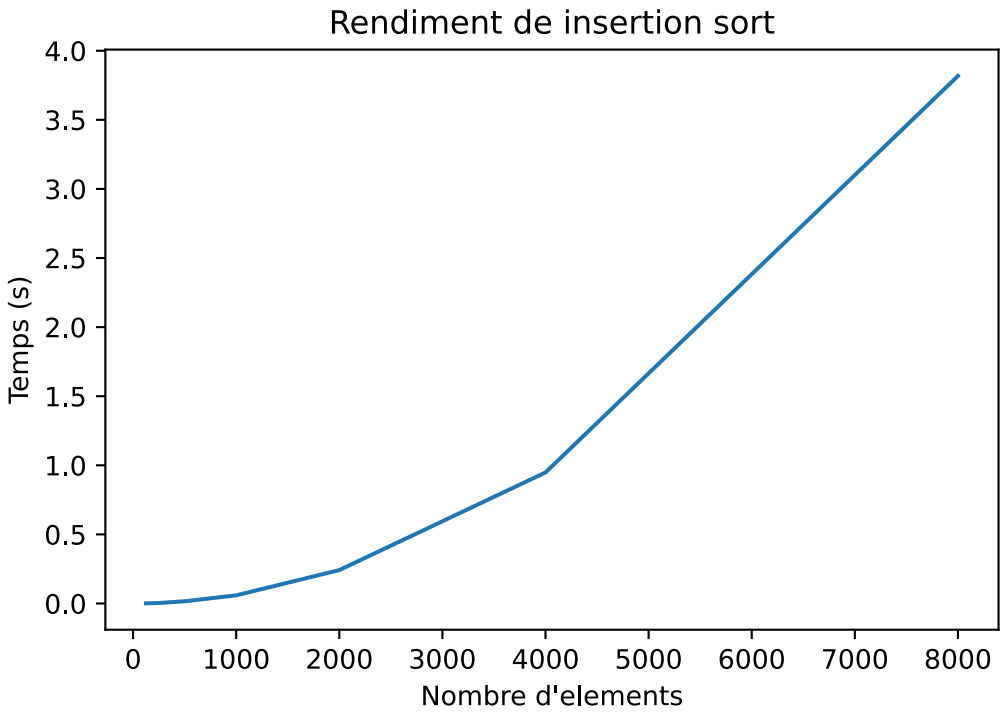
4.2. Pseudocodi

4.3. Implementació

```
1  #!/usr/bin/env python3
2  import utils
3
4  def sort(array):
5      for i in range(len(array)):
6          comp = array[i]
7          j = i - 1
8          while j >= 0 and array[j] > comp:
9              array[j + 1] = array[j]
10             j -= 1
11         array[j + 1] = comp
```

```
12      return array
13
14  if __name__ == "__main__":
15      array = utils.numbers()
16      print(sort(array))
```

4.4. Rendiment



5. Bubble sort

5.1. Com funciona?

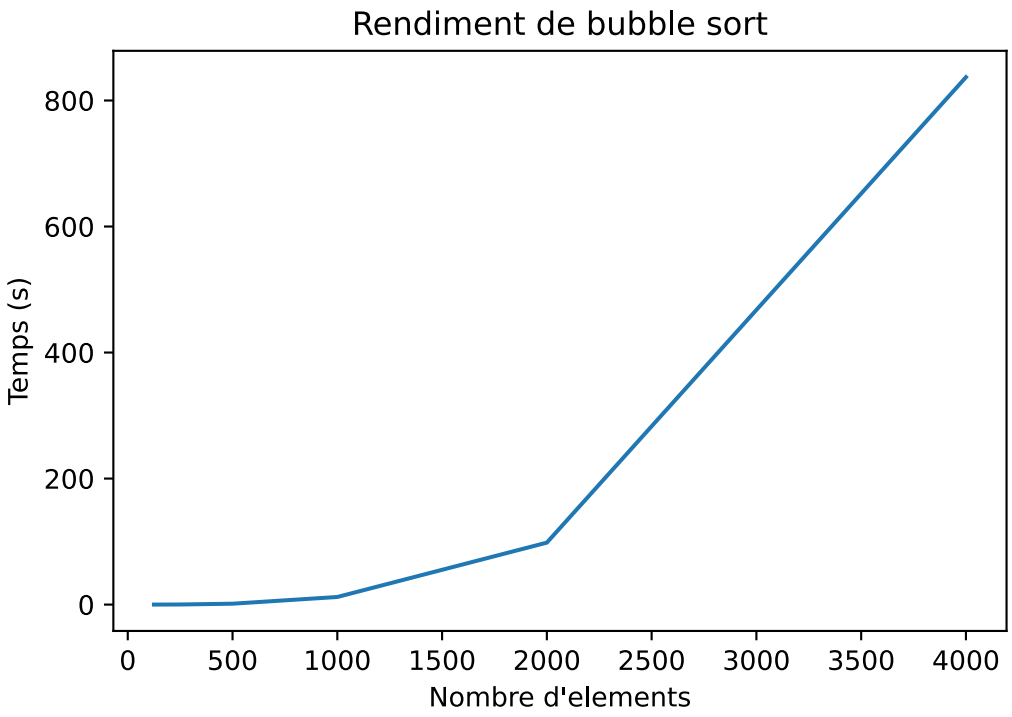
El bubble sort compara cada element de la llista amb el següent, canviant-ne la posició si el segon és més gran que el primer. Aquest procés es repeteix tantes vegades com faci falta per ordenar la llista.

5.2. Pseudocodi

5.3. Implementació

```
#!/usr/bin/env python3
import utils

def sort(array):
    while True:
        for i in range(len(array[:-1])):
            if array[i] > array[i+1]:
                array[i], array[i+1] = array[i+1], array[i]
                break

        # aquest else s'executa si el bloc anterior s'ha executat sense cap break, és a dir, si mai s'ha complert la condició array[i] > array[i+1]
        else:
            break
    return array

if __name__ == "__main__":
    array = utils.numbers()
    print(sort(array))
```

5.4. Rendiment



6. Shell sort

6.1. Com funciona?

L'ordenació shell és un algoritme que abans de comparar els elements que es troben un al costat de l'altre per tal de anar creant una llista ordenada progressivament el que fa és compartimentar la tasca agrupant nombres similars.

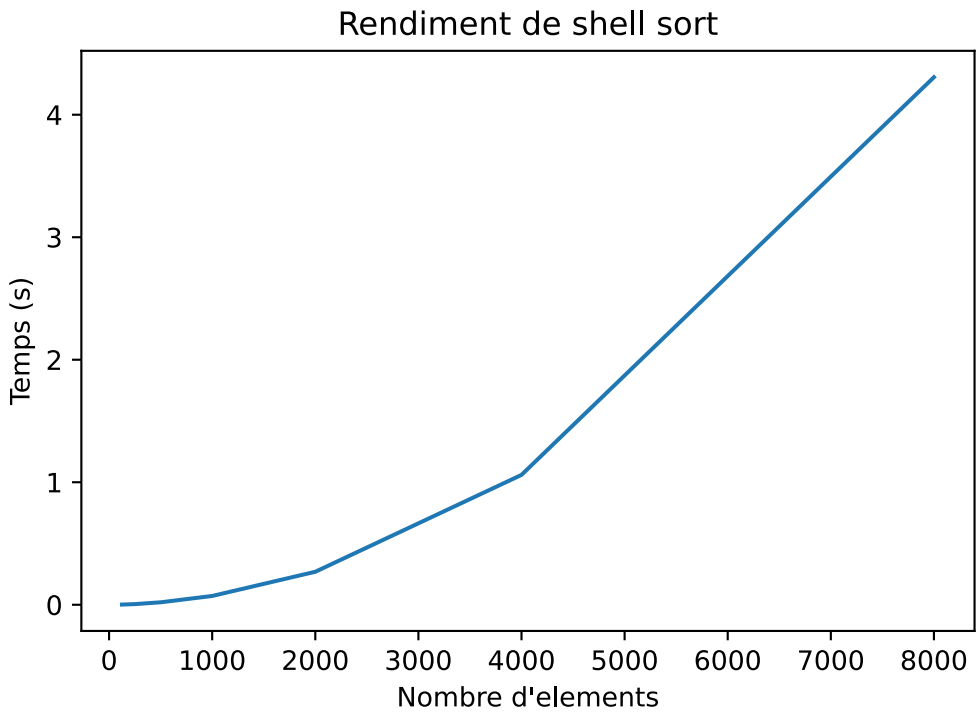
Comença agafant el primer element, i no el compara amb el segon sinó amb l'enèsim, el que sigui més petit ocupa la posició del primer. Continua així fins que arriba al final de la llista, quan torna a començar però fent salts (entre el primer i l'enèsim element) més petits. La última iteració es fa amb passos d'u.

6.2. Pseudocodi

6.3. Implementació

```
#!/usr/bin/env python3
import utils
import insertion

def sort(array):
    step = len(array) // 2
    while step != 0:
        for offset in range(step):
            sorted = insertion.sort(array[offset::step])
            for i in range(len(sorted)):
                array[step*i + offset] = sorted[i]
        step //= 2
    return sorted

if __name__ == "__main__":
    array = utils.numbers()
    print(sort(array))
```

6.4. Rendiment



7. Quick sort

7.1. Com funciona?

El quick sort és un algoritme recursiu, que divideix la llista inicial en dos i ordena cada meitat repetint els mateixos passos.

Es pren un element qualsevol que fa de pivot i separa els elements restants en funció de si són més grans o més petits que el pivot. Aquestes dues llistes de nombres menors i majors s'orednaran de la mateixa manera.

Una vegada les llistes contenen un sol element es comencen a ajuntar: la llista amb els nombres menors passa al davant, seguida del pivot i la dels nombres majors.

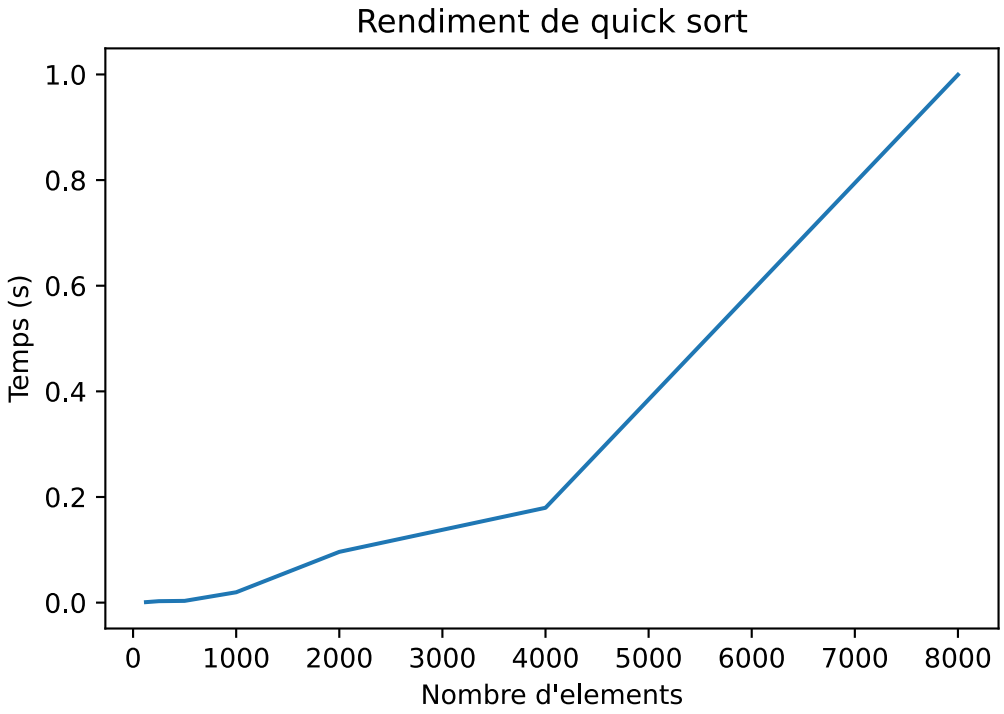
7.2. Pseudocodi

7.3. Implementació

```
#!/usr/bin/env python3
import utils

def sort(array):
    if len(array) < 2:
        return array

    pivot = array[0] # agafo el primer element com a pivot, pot ser qualsevol
    lower, higher = [], []

    for i in array[:-1]:
        if i < pivot:
            lower.append(i)
        else:
            higher.append(i)

    lower = sort(lower)
```

```
18      higher = sort(higher)
19
20      return lower + [pivot] + higher
21
22  if __name__ == "__main__":
23      array = utils.numbers()
24      print(sort(array))
```

7.4. Rendiment



8. Merge sort

8.1. Com funciona?

Aquest algoritme també és recursiu que divideix la llista en dos i treballa per separat cada meitat. El que fa amb cada meitat és el mateix: separar-la en dos i això es repeteix fins que es troba amb elements individuals, que compara i ordena formant parelles.

Una vegada tenim les parelles (llistes de dos elements ordenats) podem reconstruir la llista següent: el que fa l'algoritme és agafar l'element més petit de una llista (el primer) i comparar-lo amb el més petit de l'altra. L'element més petit d'aquests dos es posa a la nova llista. El procés es va repetint fins que una llista es buida, aleshores els elements restants de l'altra es poden afegir al final (sempre respectant l'ordre entre ells). La següent llista aplica el mateix procediment, i així fins que es recupera la llista original, ara amb els elements ordenats.

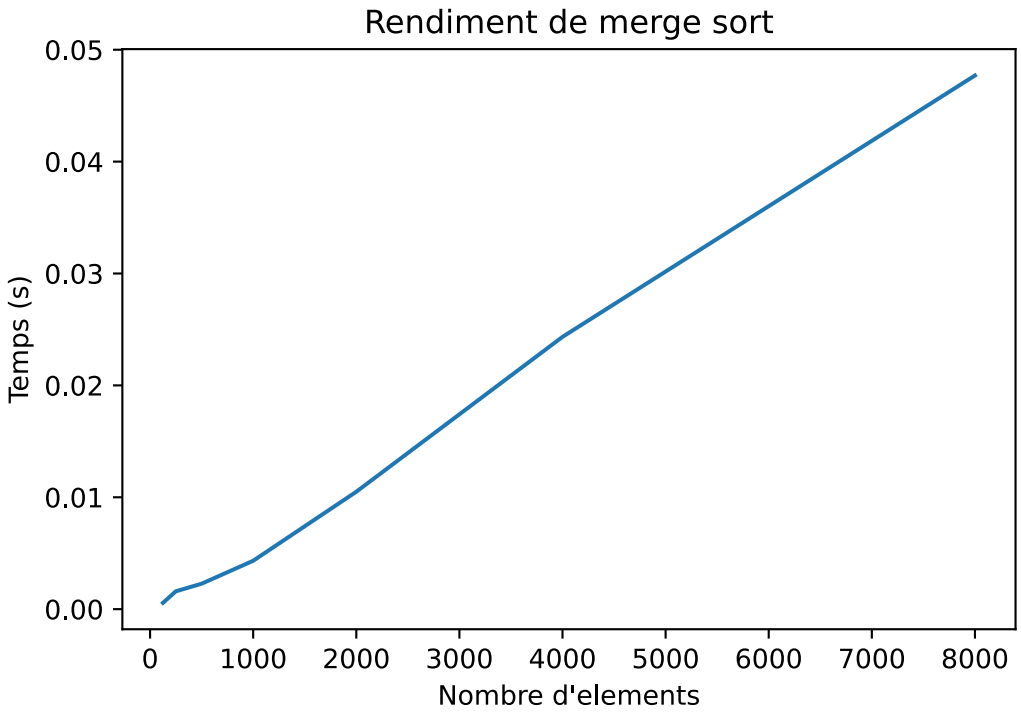
8.2. Pseudocodi

8.3. Implementació

```
#!/usr/bin/env python3
import utils

def sort(array):
    if len(array) < 2:
        return array
    mid = len(array)//2
    left = sort(array[:mid])
    right = sort(array[mid:])

    merged = []
    l = 0
    r = 0
```

```
14
15      while l < len(left) and r < len(right):
16          if left[l] < right[r]:
17              merged += [left[l]]
18              l += 1
19          else:
20              merged += [right[r]]
21              r += 1
22
23      if l < len(left):
24          merged += left[l:]
25      if r < len(right):
26          merged += right[r:]
27
28      return merged
29
30  if __name__ == "__main__":
31      array = utils.numbers()
32      print(sort(array))
```

8.4. Rendiment



9. Comparació dels algoritmes