Búsqueda y ordenamiento

Informática I - 2547100

Departamento de Ingeniería Electrónica
y de Telecomunicaciones
Facultad de Ingeniería
2016-2

Search algorithms

```
def linSearch(L, e):
    for k in range(len(L)):
        if L[k] == e:
            return True
    return False
```

Búsqueda lineal
O(len(L))

```
def ordSearch(L, e):
    for k in range(len(L)):
        if L[k] == e:
            return True
        elif L[k] > e:
            return False
    return False
```

Optimización para listas ordenadas
Peor caso -> O(len(L))
Caso promedio -> O(len(L))/2

```
def binSearch(L, e):
    low = 0
    high = len(L)-1
    while low <= high:
        mid = (low + high)//2
        if e>L[mid]:
            low=mid+1
        elif e<L[mid]:
            high=mid-1
        else:
            return True
    return False
```

Búsqueda binaria
Optimización para listas ordenadas
Peor caso -> O(log(len(L)))

¡Puedo buscar muy rápido si mis datos están ordenados!

Sort before search?

Sea ***O(complOrd(L))*** la complejidad de ordenar una lista. Como sabemos que cualquier lista se puede buscar en ***O(len(L))***, ¿será entonces que la siguiente inecuación es cierta?

$$\overset{\substack{\text{ordenar}\\ \downarrow}}{\mathit{complOrd}(L)} + \overset{\substack{\text{buscar rápido}\\ \downarrow}}{\log(\mathit{len}(L))} < \overset{\substack{\text{buscar lento}\\ \downarrow}}{\mathit{len}(L)}$$

NO, porque el ordenamiento es como mínimo lineal: es necesario acceder a todos los elementos. Pero, ¿qué tal si ordenamos una vez y hacemos ***k*** búsquedas?

$$\mathit{complOrd}(L) + k*\log(\mathit{len}(L)) < k*\mathit{len}(L)$$

Para valores grandes de ***k***, *complOrd(L) es despreciable*.

Selection sort

```python
def selSort(L):
    '''Asume que L es una lista de elementos que pueden ser comparados con el operador >
    Ordena L ascendentemente'''
    sufStart = 0
    while sufStart < len(L):
        for i in range(sufStart+1, len(L)):
            if L[i]<L[sufStart]:
                temp = L[i]
                L[i] = L[sufStart]
                L[sufStart] = temp
        sufStart += 1
```

O(len(L))

O(len(L))

O(len(L)^2)

L	len(L)	i	L[i]	sufStart	L[sufStart]
[5, 2, 10, 1, 9]	5	1	2	0	5
[2, 5, 10, 1, 9]	5	2	10	0	2
[2, 5, 10, 1, 9]	5	3	1	0	2
[1, 5, 10, 2, 9]	5	4	9	0	1
[1, 5, 10, 2, 9]	5	2	10	1	5
[1, 5, 10, 2, 9]	5	3	2	1	5
[1, 2, 10, 5, 9]	5	4	9	1	2
[1, 2, 10, 5, 9]	5	3	5	2	10
[1, 2, 5, 10, 9]	5	4	9	2	5
[1, 2, 5, 10, 9]	5	4	9	3	10
[1, 2, 5, 9, 10]	5			4	

Merge sort

Es un algoritmo del tipo *divide y vencerás*, inventado por John von Neumann en 1945.

El algoritmo parte de la observación de que dos listas ordenas pueden mezclarse en una sola lista ordenada en tiempo lineal.

Lista 1	Lista 2	Lista mezcla
[5, 22, 31, 35]	[1, 34, 46]	[]
[5, 22, 31, 35]	[34, 46]	[1]
[22, 31, 35]	[34, 46]	[1, 5]
[31, 35]	[34, 46]	[1, 5, 22]
[35]	[34, 46]	[1, 5, 22, 31]
[35]	[46]	[1, 5, 22, 31, 34]
[]	[46]	[1, 5, 22, 31, 34, 35]
[]	[]	[1, 5, 22, 31, 34, 35, 46]

Merge algorithm

```
def merge(left, right):
    '''Asume que left y right son listas ordenadas
       Retorna una lista ordenada con la unión de left y right'''
    result = []
    i = 0
    j = 0
    while i < len(left) and j < len(right):   ⟶   O(len(left)+len(right))
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    while(i < len(left)):   ⟶   O(len(left))
        result.append(left[i])
        i += 1
    while(j < len(right)):   ⟶   O(len(right))
        result.append(right[j])
        j += 1
    return result
```

$O(2*len(left) + 2*len(right)) \approx \mathbf{O(len(left)) \approx O(len(right))}$

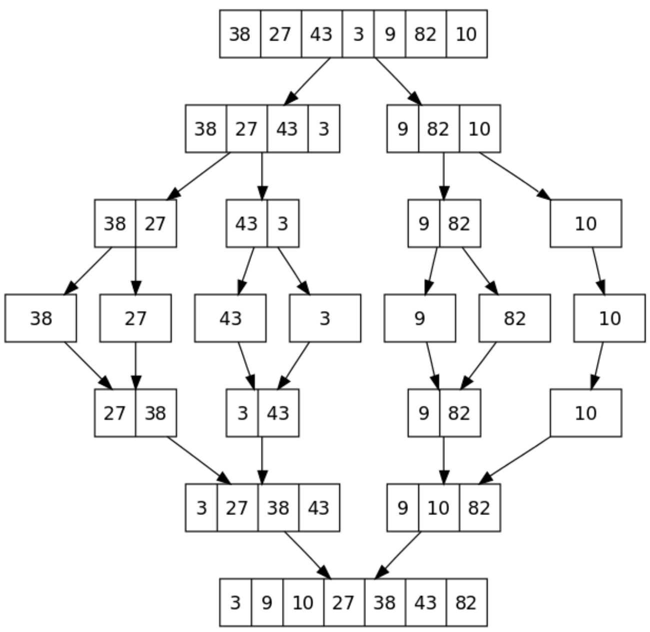
Merge sort algorithm

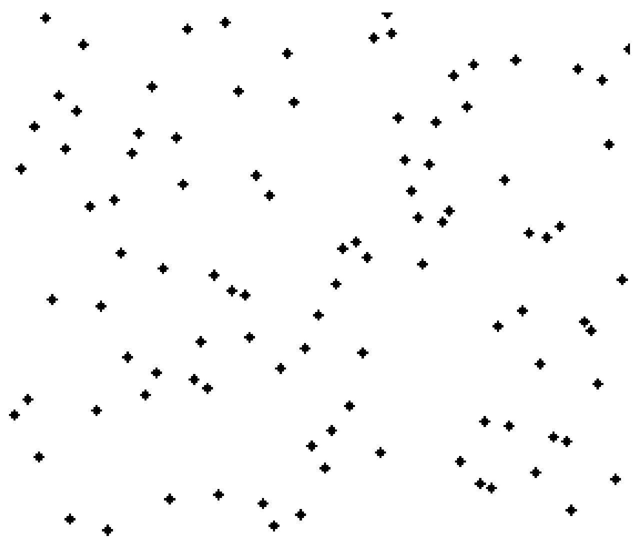
```
def mergeSort(L):
    '''Asume que L es una lista
       Retorna una nueva lista ordenada con los mismos elementos de L'''
    if len(L) < 2:
        return L[:]
    else:
        mid = len(L)//2
        left = mergeSort(L[:mid])
        right = mergeSort(L[mid:])
        return merge(left, right)
```

$O(log(len(L)))$

$O(len(L))$

$\mathbf{O(len(L)*log(len(L))}$





Sorting in Python

```
L=[3, 5, 2]
D = {'a':12, 'c':5, 'b':'dog'}
print(sorted(L))
print(L)
L.sort()
print(L)
print(sorted(D))
D.sort()
```

Python sort: Timsort
Peor caso: O(len(L)*log(len(L))
Caso promedio: < O(len(L)*log(len(L))

```
[2, 3, 5]
[3, 5, 2]
[2, 3, 5]
['a', 'b', 'c']

Traceback (most recent call last):
    ...
    D.sort()
AttributeError: 'dict' object has no attribute 'sort'
```