7 Algorithmes de tris

July 26, 2024

1 Tri par insertion

[2]:
```
def insert(val:int, li:[int])->[int]:
    if li == []:
        return [val]
    else:
        if val < li[0]:
            return [val] + li
        else:
            return [li[0]] + insert(val, li[1:])
```

[3]:
```
insert(3, [2, 4, 5])
```

[3]:
```
[2, 3, 4, 5]
```

[4]:
```
def insertionsort(li:[int])->[int]:
    if li == []:
        return []
    else:
        return insert(li[0], insertionsort(li[1:]))
```

[5]:
```
insertionsort([4, 6, 1, 3, 2, 5])
```

[5]:
```
[1, 2, 3, 4, 5, 6]
```

La complexité la fonction `insert` est $O(n)$ puisque dans le pire cas il faut parcourir toute la liste pour insérer.

2 Tri par sélection

[6]:
```
def min(li:[int])->int:
    if len(li) == 1:
        return li[0]
    else:
        m = min(li[1:])
        if li[0] < m:
            return li[0]
```

```
        else:
            return m
```

[7]:
```
min([4, 6, 1, 3, 2, 5])
```

[7]:
```
1
```

[8]:
```
def suppr(val:int, li:[int])->[int]:
    if li == []:
        return li
    else:
        if li[0] == val:
            return li[1:]
        else:
            return [li[0]] + suppr(val, li[1:])
```

[10]:
```
suppr(1, [4, 6, 1, 3, 2, 1, 5])
```

[10]:
```
[4, 6, 3, 2, 1, 5]
```

[11]:
```
def selectionsort(li:[int])->[int]:
    if li == []:
        return li
    else:
        m = min(li)
        return [m] + selectionsort(suppr(m, li))
```

[12]:
```
selectionsort([3, 1, 4, 2, 5])
```

[12]:
```
[1, 2, 3, 4, 5]
```

Les complexités des fonctions `min` et `suppr` sont $O(n)$, donc la complexité du tri est $O(n^2)$ (n appels à ces deux fonctions).

3 Tri fusion

[15]:
```
def dc(li:[int])->[int]:
    print(li)
    if len(li) > 1:
        m = len(li)//2
        dc(li[:m])
        dc(li[m:])
```

[14]:
```
dc([1, 2, 3, 4, 5])
```

```
[1, 2, 3, 4, 5]
[1, 2]
[1]
```

```
[2]
[3, 4, 5]
[3]
[4, 5]
[4]
[5]
```

[16]:
```
def sum(li:[int])->int:
    if li == []:
        return 0
    elif len(li) == 1:
        return li[0]
    else:
        m = len(li)//2
        return sum(li[:m]) + sum(li[m:])
```

[17]:
```
sum([1, 2, 3, 4])
```

[17]: 10

[29]:
```
def merge(li1:[int], li2:[int])->[int]:
    if li1 == [] or li2 == []:
        return li1 + li2
    else:
        if li1[0] < li2[0]:
            return [li1[0]] + merge(li1[1:], li2)
        else:
            return [li2[0]] + merge(li1, li2[1:])
```

[30]:
```
merge([1, 3, 5], [2, 4])
```

[35]:
```
def mergesort(li:[int])->[int]:
    if len(li) <= 1:
        return li
    else:
        m = len(li)//2
        return merge(mergesort(li[:m]), mergesort(li[m:]))
```

[36]:
```
mergesort([4, 6, 1, 3, 2, 5])
```

4 Tri rapide

[38]:
```
def quicksort(li:[int])->[int]:
    if len(li) <= 1:
        return li
    else:
        p = li[0]
```

```
        return quicksort([el for el in li if el < p]) + [p] + quicksort([el for␣
↪el in li if el > p])
```

[39]:
```
quicksort([4, 6, 1, 3, 2, 5])
```

[ ]:
```

```