Rapport
Projet : programmation fonctionnelle

Quentin Januel & Clément Defrétiére

5 avril 2018

1 Tests

1.1 Listes courtes

1.2 Listes longues

1.3 Listes déjà triées

2 Analyse de complexité

Nous allons à présent analyser le temps que prend chaque fonction de tri en fonction de la longueur de la liste qu'on lui passe en argument, en nous basant directement sur le code source de chacune des trois fonctions de tri.

Pour ce faire, nous allons d'abord considérer que ce temps est proportionnel au nombre de calculs que doit faire tel algorithme en fonction de la longueur de la liste.

Ainsi, pour une liste de taille N, nous recherchons l'ordre de grandeur en fonction de N du nombre d'opérations que va effectuer l'algorithme.

Par soucis de simplification, nous allons considérer que les fonctions comme `List.length` ou `@` sont instantannées ou au moins proportionnelles à la longeur des listes avec lesquelles on les utilise.

Afin de représenter la complexité d'un algorithme, nous allons utiliser la notation O.
Ainsi, un algorithme ayant une complexité $O(n^2)$ sera un algorithme qui, pour une liste de taille n, effectue $n \times n$ opérations (et prend donc un temps relativement proportionnel).

Pour finir, nous n'allons pas prendre en compte les opérations ne dépendant pas de n, ni les facteurs multiplicatifs de n puisque ces derniers n'auront aucun impact quand n sera suffisamment grand.
Une complexité $O(100 + 5 \times n)$ est donc la même que $O(n)$.

2.1 Tri par création du maximum

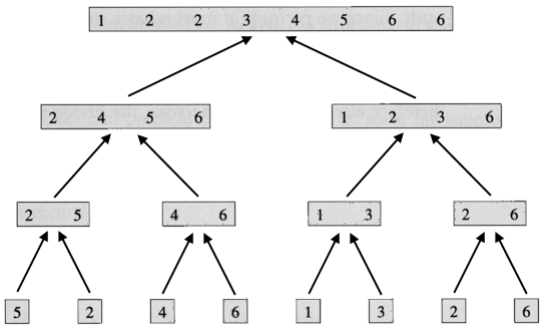
2.2 Tri par fusion

Voici le code source de la fonction de tri par fusion :

```
let rec tri_partition_fusion comp l =
    let l1, l2 = partitionne l in
        if (List.length l1)+(List.length l2) < 2 then
            l1@l2
        else
            let sorted1 = tri_partition_fusion comp l1
            and sorted2 = tri_partition_fusion comp l2 in
                fusionne comp sorted1 sorted2
;;
```

Nous pouvons constater que cette fonction fait elle même appel à deux fonctions ; `partitionne` et `fusionne`. Ces dernières ont toutes deux une complexité $O(n)$.

A chaque étape, cet algorithme coupe la liste en deux, pour ensuite les fusionner.
Nous pouvons le représenter par le diagramme ci dessous :



Appelons niveau i de l'algorithme la i^{eme} ligne du schéma.

A chaque niveau, la liste est coupé en p parties et la longueur de chaque sous liste est de n/p. Il y aura donc exactement n calculs par niveau (puisque, rappelons le, `partitionne` et `fusionne` sont de complexité $O(n)$).

La complexité de cet algorithme est donc $O(n\times < \#niveaux >)$.
Le nombre de niveaux est le nombre de fois qu'il faut couper une liste en

deux jusqu'à n'obtenir qu'un seul élément.
Supposons que n soit de la forme 2^a, il y aura donc a niveaux.
Plus généralement, pour une liste de longueur n, il y aura approximativement $log_2(n) = log(n)/log(2)$ niveaux.

Cet algorithme a donc une complexité $O(n \times ln(n))$.

2.3 Tri par arbre binaire de recherche

3 Comparaisons des courbes

4 Bilan