Algorithmique Avancée : Séance 5

Frédéric A. Hayek

Mercredi 5 octobre 2022

Tri par Sélection

Exercice 1 : Tri par Sélection

Le tri par insertion va chercher le minimum de la liste et l'échanger avec le premier élément, puis chercher le minimum dans le restant de la liste et l'échanger avec le second élément, et ainsi de suite.

1. Écrire le pseudocode d'un algorithme de tri par sélection.
2. Calculer la complexité au meilleur cas et au pire cas.
3. Trouver un invariant de boucle.
4. Prouver que l'algorithme est correct.

Tri par Sélection

Corrigé

```
1:  function tri_selection(tab)
2:      for i ∈ {0, ··· , len(tab) − 1} do
3:          min ← +∞
4:          for j ∈ {i, ··· , len(tab) − 1} do
5:              if tab[j] < min then
6:                  min ← tab[j]
7:                  argmin ← j
8:          temp ← tab[i]
9:          tab[i] ← tab[argmin]
10:         tab[argmin] ← temp
```

Tri par Sélection

Corrigé

```
1:  function tri_selection(tab)
2:      for i ∈ {0, ··· , len(tab) − 1} do
3:          min ← +∞
4:          for j ∈ {i, ··· , len(tab) − 1} do
5:              if tab[j] < min then
6:                  min ← tab[j]
7:                  argmin ← j
8:          temp ← tab[i]
9:          tab[i] ← tab[argmin]
10:         tab[argmin] ← temp
```

La complexité est de $\Theta(n^2)$ au meilleur cas et $\Theta(n^2)$ au pire cas.

Recherche

Exercice 2 : Recherche par dichotomie

On suppose que la liste est ordonnée par ordre croissant. La recherche par dichotomie va délimiter la zone à chercher par des variables `gauche` et `droite`, qui commencent à `0` et `len(tab)-1`. À chaque itération, on regarde si l'élément au milieu de cette zone est l'élément recherché. Si oui, on renvoie son indice. Si non, si l'élément au milieu est plus grand que l'élément recherché, alors on se restreint à la demi-zone à gauche; si l'élément au milieu est plus petit que l'élément recherché, alors on se restreint à la demi-zone à droite.

1. Écrire le pseudocode d'un algorithme de recherche par dichotomie.
2. Calculer la complexité au meilleur cas et au pire cas.
3. Prouver que l'algorithme est correct.

Recherche

Corrigé : Recherche par dichotomie

```
1:  function recherche_dichotomie(tab, e)
2:      gauche ← 0
3:      droite ← len(tab) − 1
4:      i ← ⌊(gauche+droite)/2⌋
5:      while gauche ≤ droite do
6:          if tab[i] == e then
7:              return i
8:          else if tab[i] < e then
9:              gauche ← i + 1
10:         else
11:             droite ← i − 1
12:         i ← ⌊(gauche+droite)/2⌋
13:     return −1
```

Recherche

Corrigé : Recherche par dichotomie

Au meilleur cas, on fait $\Theta(1)$ opérations.
Le contenu de la boucle est fait en $\Theta(1)$ opérations. Au pire cas, combien de fois fait-on ce nombre constant d'opérations ? Donc combien de fois parcourt-on la boucle au maximum?
Autant de fois qu'il faut diviser n par 2 pour obtenir 1. Soit T ce nombre de fois. On a donc $\frac{n}{2^T} = 1 \iff n = 2^T \iff \log(n) = T$

Recherche

Corrigé : Recherche par dichotomie

Au meilleur cas, on fait $\Theta(1)$ opérations.
Le contenu de la boucle est fait en $\Theta(1)$ opérations. Au pire cas, combien de fois fait-on ce nombre constant d'opérations ? Donc combien de fois parcourt-on la boucle au maximum?
Autant de fois qu'il faut diviser n par 2 pour obtenir 1. Soit T ce nombre de fois. On a donc $\frac{n}{2^T} = 1 \iff n = 2^T \iff \log(n) = T$

Rappel et définition

$\forall \beta \in \mathbb{R}^{+*}, \log_\beta(1) = 0, \log_\beta(\beta) = 1$
$\log(n) = \log_2(n)$
$\ln(n) = \log_e(n)$

Récursivité

la vache qui rit®

Récursivité

Exemple

```
1: function fact(n)
2:     if n == 0 then
3:         return 1
4:     else
5:         return n × fact(n − 1)
```

Récursivité

Exemple

```
1: function fact(n)
2:     if n == 0 then
3:         return 1
4:     else
5:         return n × fact(n − 1)
```

$$T(n) = \Theta(1) + T(n-1) \implies T(n) = \Theta(n)$$

Récursivité

Exemple

```
1: function fact(n)
2:     if n == 0 then
3:         return 1
4:     else
5:         return n × fact(n − 1)
```

$$T(n) = \Theta(1) + T(n-1) \implies T(n) = \Theta(n)$$

Construction type

```
1: function recurs(params)
2:     if conditions d'arret then
3:         return quelque chose
4:     else
5:         appeler recurs en changeant params
6:         return quelque chose d'autre
```

Récursivité

Exercice 3 : Suites

Soit $u_{n+1} = u_n + 3$ avec $u_0 = 1$.

1. Prouver que $u_n = 1 + 3n$.
2. Écrire un algorithme non récursif qui calcule u_n.
3. Écrire un algorithme récursif qui calcule u_n.
4. Exprimer sa complexité sous forme de récurrence.
5. Expliciter sa complexité

Exercice 4 : Somme liste

1. Écrire un algorithme non récursif qui calcule la somme des éléments d'une liste. Donner sa complexité.
2. Écrire un algorithme récursif qui calcule la somme des éléments d'une liste. Donner sa complexité.