Diviser pour régner : Rappels de 1ere année

September 24, 2024

1 Recherche dichotomique

On considère une suite de nombres L triée par ordre croissant. On y cherche un nombre x.

Q1

Ecrire une fonction `dichotomie(L,x)` qui renvoie un bouléen indiquant si $x \in L$ (`True`) ou non (`False`).

Q2

Donner sa complexité.

[2]:
```
t = [1,10,20,25,101]
print(dichot(t,25), end =" ")
print(dichot(t,26),end =" ")
print(dichot(t,101),end =" ")
print(dichot(t,102),end =" ")
print(dichot(t,1),end =" ")
print(dichot(t,0),end =" ")
print(dichot(t,12),end =" ")
```

```
True False True False True False False
```

2 Tri rapide

On rappelle le principe du *tri rapide* :

Placer un élément du tableau (le *pivot* à sa place définitive par *partitionnement*.

- Partitionnement : Mettre les éléments plus petits que le pivot à sa gauche, les autres à sa droite.
- Pour chaque sous-tableau : définir un nouveau pivot et recommencer récursivement.

A la fin le tableau est trié.

Q1

Ecrire la fonction `partition(L,i)` qui prend en paramètres une liste de nombres et la position du pivot. Elle retourne deux listes : la première contient les éléments plus petits que le pivot; la

seconde les éléments plus grands.

Le pivot est l'élément `L[i]`.

Q2

Ecrire la fonction `quick(L)` qu prend en paramètre une liste `L` et renvoie une version triée de `L` par l'algorithme du tri rapide.

A chaque étape, le pivot est le premier élément de la liste

Q3

On note $C(n)$ la complexité pour une liste de taille n.

Etablir une relation de récurrence respectée par la complexité.

Q4

Calculer la complexité si la liste est triée.

Q5

On suppose qu'à tout instant le tableau est divisé en deux parties de tailles proches. Montrer qu'on peut alors poser :

$$C(n) \leq U(n)$$

où $U(n)$ est une suite *diviser pour régner*.

Etudier une relation de domination pour $U(n)$ en supposant $n = 2^p$. Extrapoler au cas général.

En déduire une relation de même nature pour $C(n)$.

3 Tri fusion

Basé sur le principe *divisier pour régner* qui est un cas particulier de *programmation dynamique*.

- algorithme de tri par comparaison *stable* (qui conserve l'ordre des éléments identiques).
- Opération principale : *la fusion*, c.a.d réunir deux listes triées en une seule.

Principe :

- On coupe en deux parties à peu près égales les données à trier;
- On trie récursivement les données de chaque partie;
- On fusionne les deux parties.

Q1

Ecrire la fonction *merge(A,B)* qui fusionne deux listes A, B **triées par ordre croissant**. La liste renvoyée est croissante. Un temps linéaire en la somme des longueurs de liste doit être respecté.

[2]:
```
A = [5,10,14]
B= [6,8,11,15]
merge(A,B)
```

[2]:
```
[5, 6, 8, 10, 11, 14, 15]
```

Q2

Etablir la terminaison puis la correction du programme de fusion.

Q3

Ecrire la fonction `mergesort(L)` qui retourne une version triée de la liste L. L'algorithme est celui du tri fusion.

[4]:
```
mergesort([10,-1,10,5,3,12,4])
```

[4]:
```
[-1, 3, 4, 5, 10, 10, 12]
```

Q4

Etudier la terminaison, la correction et la complexité.