# TP2 ACT : Diviser pour régner

Question 1 :

* (2; 0)(2; 5)(4; 4)(4; 7)(5; 7)(5; 0)

C’est une ligne de toit. (pas sûre)

* (2; 0)(2; 5)(4; 5)(4; 7)(5; 7)(5; 0)

C’est une ligne de toit. (pas sûre)

* (2; 0)(2; 5)(4; 5)(4; 7)(5; 7)(6; 7)(5; 0)

Ce n’est pas une ligne de toit car les x ne sont pas triés.

* (2; 0)(2; 5)(4; 5)(4; 8)(4; 7)(5; 7)(5; 0)

C’est une ligne de toit.

Question 2 :

1er point : parcourir 1 fois la liste de tous les points 🡪 O(n)

2nd point : parcourir la table de taille n \* n pour la remplir 🡪 O(n²)

3rd point : parcourir la table pour le calcul de la ligne 🡪 O(n²)

Avec une telle approche nous serions approximativement en O(2n² + n)

Le désavantage est que l’on parcourt deux fois une table da taille qui pourrait être importante.

Question 3 :

Cette méthode correspond au tri par insertion.

On tri les immeubles suivant leur 1ère valeur(le 1er X).

Question 4 :

Question 5 :

Ligne de toit

Point 1

Point droit du bâtiment

Point gauche du bâtiment

On parcourt la liste des points qui composent la ligne de toit.

On cherche la position (l’indice) du point gauche du bâtiment dans cette liste.

Ensuite, on parcourt le reste de la liste à partir de l’indice que l’on a trouvé pour trouver le point droit du bâtiment et à chaque itération on compare les hauteurs pour savoir si on supprime ou insère un point dans cette liste.

Pour i de 1 à n

Recherche indice i qui correspond à l’emplacement dans la liste (ligne de toit) du point gauche de l’immeuble

Pour j de i + 1 à n

Recherche indice i qui correspond à l’emplacement dans la liste (ligne de toit) du point droit de l’immeuble

Comparaison des hauteurs entre l’immeuble et les points de la ligne de toit

La complexité est en O(n) car dans le pire des cas on parcourt une seule fois la totalité de la ligne de toit qui est de taille n.

Algorithme pour construire la ligne de toit :

Pour i de 1 à n (n = nombre immeubles)

ajouterImmeuble(immeuble)

Complexité : O(n) PAS SURE je pense que c’est O(n²) mais ça reviendrai à la complexité d’avant ça serait chelou

Question 6 :

Algorithme :

fusion(list1, list2) {

List result ;

**int** t1 = list1.size();

**int** t2 = list2.size();

**int** i\_l1 = 0;

**int** i\_l2 = 0;

**int** i;

**pour**(i = 0; i\_l1 < t1 && i\_l2 < t2; i++)

**si**(list1.get(i\_l1) <= list2.get(i\_l2)) {

result.add(i, list1.get(i\_l1));

i\_l1++;

}

**sinon** {

result.add(i, list2.get(i\_l2));

i\_l2++;

}

**Tant que**(i\_l1 < t1) {

result.add(i, list1.get(i\_l1));

i++;

i\_l1++;

}

**Tant que**(i\_l2 < t2) {

result.add(i, list2.get(i\_l2));

i++;

i\_l2++;

}

**retourner** result;

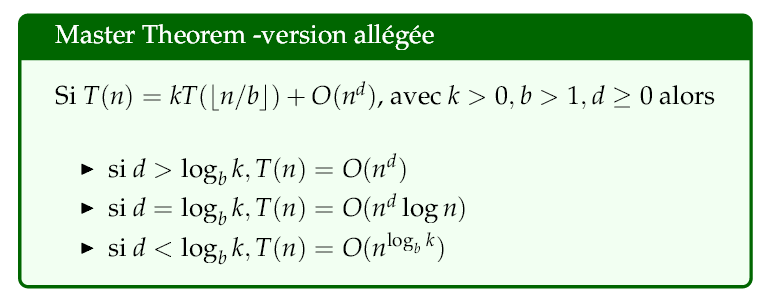
}

Le nombre de comparaisons pour fusionner 2 tableaux de taille k et m est en (k + m), puisque on accède une fois et une seule à chaque case des 2 tableaux et donc ici ( r - p + 1 ) = ( n à la 1ère itération).

On peut donc dire que la complexité de la fusion est de O(n).

Question 7 :

Rappel du Master Théorème :



Ce que l’on cherche à obtenir est une complexité en O (ndlogn) avec d = 1.

Ici nous divisons le problème en 2, car on divise la liste d’immeubles en 2 puis on génère la ligne de toit de ces 2 listes et enfin on la fusionne.

Complexité temporelle :

•T(n) = nombre de comparaisons avec n = r-p+1

•Tableau de taille n, partagé en 2 tableaux de taille n/2; on les trie récursivement et on les fusionne. On arrête lorsque le tableau est de taille 1 (p= r).

T ( 1 ) = 0

T ( n ) = T ( n/2 ) + T ( n/2 ) + g(n) n >1

Complexité de fusion (A, P, Q, R) = G(N)

Le nombre de comparaisons pour fusionner 2 tableaux de taille k et m est en ( k + m), puisque on accède une fois et une seule à chaque case des 2 tableaux et donc ici ( r - p + 1 ) = ( n à la 1èreitération).

* en fait n/2 dans le meilleur des cas
* n-1dans le pire des cas

Ainsi donc g(n) =θ( n ).

Complexité de la fusion:

•Fonction T(n) monotone croissante strictement.

On pose donc n =2 i

.

T (n ) = T ( 2i )= 2 T ( 2i-1 ) +α2i

Sommons, après simplification on obtient

T(n) = T (2i) = [α2i+ α2i+... + α2i] + 2i.T(1) = αi 2i

{i termes} {0}

Puisque i = log2n, T(n) =αn log2(n) et donc

Min (n) = Moy (n) = Max (n) =θ( n log2n )

Remarque :

Lorsque l’on génère une ligne de toit pour une liste d’immeubles, la ligne de toit qui est retournée est triée car dans la fonction qui insère un immeuble à la ligne de toit, on insère à l’endroit correspondant pour garder la ligne de toit triée par ordre croissant des valeurs en X.

Algorithme :

genererLigneToit(Liste immeubles) {

/\* on divise la liste des immeubles en 2 listes d’où le n/2 \*/

List list1 = immeubles [0, …, n/2] ;

List list2 = immeubles [n/2 + 1, …, n-1] ;

/\* pour chaque sous liste on génère la ligne de toit correspondante \*/

LigneToit ligne1 = genererLigneToit(list1) ;

LigneToit ligne2 = genererLigneToit(list2) ;

/\* on fusionne les 2 lignes de toit obtenues \*/

LigneToit result = fusion (ligne1, ligne2) ;

Retourner result ;

}