Point Méthode

Comment calculer la complexité d'un algorithme (incrémenter et compter)

rappel: le but est d'estimer les ressources nécessaires (nombre d'instruction pour le cas temporel). le cout dépend d'un paramètre n taille des données.

soit C(n) le nombre d'instructions exécutées

la notation O permet d'estimer la complexité asymptotique

exemple: si $C(n)=n^2+bn+c$ alors $C(n) \sim O(n^2)$;

si $C(n) = 2^n + bn^2 + c$ alors $C(n) \sim O(2^n)$

méthode de comptage:

principe: on compare C(n+1) avec C(n) on a différents cas possibles:

1. C(n+1)=C(n) la complexité est O(1)
2. C(n+1) = C(n) +1 O(n)
3. C(n+1) = C(n) + epsilon O(logn) ou bien C(2n)= C(n) +1
4. C(n+1) = C(n) + n $O(n^2)$
5. C(n+1) = 2* C(n) $O(2^n)$

exemple: préparation sportive à une compétition. le paramètre n est le nombre de jours qui restent jusqu'à cette compétition

Algorithme	Complexité
faire 100 pompes indépendemment de n	O(1)
n>2 m:=n faire 1 pompe; m:=m-1; jusqu'à m=1	1 jour de plus (n+1) une pompe de plus C(n+1) = C(n) +1 donc la complexité est **O(n)**

n>2
m:=n
faire 1 pompe; m:=[illegible] /2[illegible]; // partie entière
jusqu'à m=1
	n=3 --> 1 pompes
	n=4 --> 2 pompes
	n=5 --> 2 pompes
	n=6 --> 2 pompes
	complexité O(logn)

n>2 m:=n faire n pompes; m:=m-1; jusqu'à m=1	+1 jour n pompes de plus C(n+1) = C(n) +n Complexité $O(n^2)$

n>2
m:=n; m2 = 1
faire m2 pompes n pompes; m2 := 2*m2;
m:=m-1;
jusqu'à m=1
	1p; 2p; 4p; 8p...
	C(n+1) = 2* C(n)
	$O(2^n)$

```
void tri_selection(const int n, int Tab[])
{
// n = Taille du tableau
int i,j, indice_min, echange;

for(i = 0; i < (n-1); i++)
{
// Recherche de la valeur minimale entre i et n
indice_min=i;

for (j=i+1; j<n; j++)
if (Tab[j] < Tab[indice_min]) indice_min=j;
// Echange entre i et indice_min
echange = Tab[i];
Tab[i]=Tab[indice_min];
Tab[indice_min]=echange;
}
```

• Dans la boucle principale, nous exécutons d'abord l'instruction élémentaire d'initialisation : i=0. Cette étape coûte une (1) unité.

• Pour chaque itération i de la boucle principale, nous réalisons :
deux (2) opérations élémentaires :

- une comparaison $i < (n - 1)$; et une incrémentation $i++$;

• quatre (4) opérations élémentaires :

1 indice_min=i;
2 echange = Tab[i];
3 Tab[i]=Tab[indice_min];
4 Tab[indice_min]=echange;

la boucle intérieure :

```
for (j=i+1; j<n; j++)
if (Tab[j] < Tab[indice_min]) indice_min=j;
```

• Notons T(n) le nombre d'instructions nécessaires pour trier un tableau d'entiers de taille n en utilisant notre fonction "Tri sélection".

• Pour un entier i compris entre 0 et n-2, notons TBI(i) le nombre d'instructions réalisées par la boucle intérieure :

```
for (j=i+1; j<n; j++)
if (Tab[j] < Tab[indice_min]) indice_min=j;
```

$T(n) = 1 + [(6 + TBI(0)) + (6 + TBI(1)) + ::: + (6 + TBI(n-2))]$
$= 1 + 6 . (n - 1) + [TBI(0) + TBI(1) + ::: + TBI(n-2)]$

- Il nous reste maintenant à calculer TBI(i) pour i compris entre 0 et (n-2).

- La boucle intérieure :

```
1 for (j=i+1; j<n; j++)
2 if (Tab[j] < Tab[indice_min]) indice_min=j;
```

contient une instruction élémentaire d'initialisation : j=i+1. Cette étape coute une (1) unité.

- Pour chaque itération j (de i + 1 jusqu' à (n-2)), nous réalisons :

    -Deux (2) opérations élémentaires :

- une comparaison j < n; et une opération d'incrémentation j + +;.
- Une (1) opération élémentaire de comparaison : Tab[j] < Tab[indice min]

Si la condition est vraie, une (1) autre opération élémentaire d'affectation indice min=j; est réalisée.

- Dans le pire des cas, nous obtenons :

$TBI(i) = 1 + 4 . (n - 1 - i).$

Reprenons le calcul de la complexité de la fonction Tri sélection est de :

$T(n) = 1 + 6 . (n - 1) + [TBI(0) + TBI(1) + ::: + TBI(n-2)]$
$= 1 + 6*(n-1)+[(1+4*(n-1-0)+1+4*(n-1-1)+...+1+4*(n-1-(n-2))]$
$= 1 + 6*(n-1)+(n-1)+[(4*(n-1-0)+4*(n-1-1)+...+4*(n-1-(n-2))]$
$= 7n-6+[(4*(n-1-0)+4*(n-1-1)+...+4*(n-1-(n-2))]$
$= 7n-6+4*[(n-1)+(n-2)+...+1]$
$= 7n-6+4* [(n-1).n]/2$

$T(n) = 2n^2 + 5n - 6$