Université de Bordeaux
Semestre 1

2021-2022

TD 2 : Complexité d'un algorithme

Compétences

- Lecture d'un algorithme informatique écrit en Python.
- Évaluer le nombre d'affections d'une variable dans un algorithme
- Évaluer la complexité en nombre d'opérations élémentaires

* **Ex. 1** — Soit la fonction `indexSort(t, n)` où `t` est un tableau de `n` entiers :

```python
def indexSort(t, n) :
    i = 0
    while ((i + 1) < n) and (t[i] <= t[i + 1]) :
        i += 1
    return i
```

1. Donner le nombre d'affectations effectuées sur la variable `i` dans la fonction `indexSort(t, n)` en fonction de la valeur de retour `i`
2. Déduire les complexités en nombre d'affectations, dans le meilleur et pire cas, de la fonction `indexSort(t, n)` en fonction de `n` (taille du tableau `t`) ?

** **Ex. 2** — Soit la fonction `sort(t, g, d, n)` où `t` est un tableau de `n` entiers et `g` et `d` deux entiers compris entre 0 et $n-1$. La fonction `sort(t, g, d, n)` utilise la fonction du TD1 `swap(t, i, j)`.

```python
def sort(t, g, d, n):
    if g >= 0 and g < d and d <= (n -1):
        for i in range(g, d):
            for j in range(i+1, d+1):
                if t[i] > t[j]:
                    swap(t, i, j)
```

1. Donner et justifier les complexités, dans le meilleur cas et pire cas, en nombre d'affectations sur le tableau `t` de la fonction `sort(t, g, d, n)` en fonction des paramètres `g` et `d`.

*** **Ex. 3** — Soit la fonction `invalidSort(t, n)` où `t` est un tableau de `n` entiers :

```python
def invalidSort(t, n):
    i = indexSort(t, n)
    sort(t, i+1, n-1, n)
```

1. Donner le nombre d'affectations effectuées sur la variable `i` et sur les variables `t[k]` du tableau lors d'un appel à la fonction `invalidSort(t, n)` en fonction de la variable `i` et de `n` (taille du tableau `t`)
2. En Déduire les complexités en nombre d'affectations de la fonction `invalidSort(t, n)` en fonction de `n` (taille du tableau `t`)
3. Expliquer pourquoi la fonction `invalidSort(t, n)` peut ne pas trier le tableau `t`.

*** **Ex. 4** — Utiliser la dichotomie dans un algorithme s'inspire du principe **diviser pour mieux régner**. Cela signifie qu'en divisant un intervalle, on se retrouve avec deux parties plus petites, donc plus faciles à analyser. D'autant que rien n'interdit ensuite de continuer la division.

Le problème : Soit t un tableau de n entiers, trié dans l'ordre croissant. On cherche à savoir si un entier x est présent ou pas dans le tableau.
Le principe : La méthode de recherche dichotomique est la suivante : on compare x à l'élément médian du tableau (élément d'index noté $m = (n-1)//2$)

- Si $t[m]$ est égal à x alors x est présent dans le tableau.
- sinon
  - soit $t[m] > x$ et on cherche x dans la première moitié du tableau (indices strictement inférieurs à m).
  - soit $t[m] < x$ et on cherche x dans la deuxième moitié du tableau (indices strictement supérieurs à m).

Puis on recommence avec le même principe sur la moitié de tableau retenue.
La terminaison : La recherche s'arrête quand $t[m] = x$ ou que x n'a pas été trouvé.

Questions :

1. Comment est calculé l'index de l'élément médian quand on utilise les index de début et de fin tableau notés respectivement d et g ?
2. Soit $t = [1,4,13,14,20,25,31,40,43,49]$ un tableau d'entiers triés. Écrire la suite des couples (d,g) délimitant les intervalles successifs de recherche de l'élément x pour les valeurs suivantes
   (a) $x = 20$
   (b) $x = 4$
   (c) $x = -1$
   (d) $x = 49$
3. À partir des couples (g,d) proposer un variant de boucle permettant d'être sûr que la recherche dichotomique se termine que x soit dans le tableau ou pas.
4. Dans le cas où la valeur cherchée n'est pas présente, comment évolue la taille du tableau dans lequel se fait la recherche lorsque la taille du tableau de départ est $N = 2^k - 1$?
5. Si $N = 2^k - 1$, combien de comparaisons entre x et des valeurs du tableau sont-elles nécessaires, dans le cas le pire (quel est-il ?), en fonction de k ? En fonction de N ?
6. Dans le cas général (N n'est pas nécessairement de la forme $2^k - 1$), évaluer le nombre de comparaisons (dans le pire des cas) à effectuer lors de la recherche dichotomique, en fonction de la taille N du tableau.
7. Écrire une fonction Python iterative `rechercheDichotomique(x, t, n)` qui effectuera la recherche dichotomique de la valeur s dans le tableau t contenant n éléments. Cette fonction renvoie l'indice de position de x dans le tableau ou -1 si x n'est pas dans le tableau.
8. À l'aide d'un tableau permettant de visualiser les variables de boucle à chaque itération, tester votre algorithme pour les valeurs de x de la question 2.