

Corrigé du TP n°6 - Algorithmes dichotomiques - Corrigé

1. Recherche dichotomique

Programmons l'algorithme de recherche dichotomique.

Dans la suite, E est un ensemble totalement ordonné, L est un tableau **trié** d'éléments de E et x est un élément de E .

Q1 : (*Rappel*) Écrire une fonction `recherche(L,x)` qui opère une recherche de complexité linéaire de x dans L .

Q2 : *Version récursive de la recherche dichotomique :*

En vous basant sur la description du cours, écrire une fonction `dicho_rec(L,x)` récursive de la recherche dichotomique de x dans L .

Q3 : *Estimation empirique de la complexité :*

a) Importer le module `random` et écrire une fonction `liste_triee(n)` qui renvoie une liste triée de nombres entiers tirés aléatoirement et uniformément entre 0 et 100. *En attendant l'étude des algorithmes de tri, on pourra se servir de la méthode `L.sort()` pour trier la liste L*

b) Modifier les fonctions `recherche(L,x)` et `dicho_rec(L,x)` de telle sorte qu'elle renvoie un compteur c qui compte le nombre de comparaisons effectuées dans chacune des fonctions. On appellera `recherche_complexite(L,x)` et `dicho_rec_complexite(L,x)` ces nouvelles fonctions.

c) Comparer les résultats de `recherche_complexite(liste_triee(n),101)` et `dicho_rec_complexite(liste_triee(n),101)` pour $n = 2^k$ où $k = 5, 10, 15, 20, 100, \dots$. Qu'en pensez-vous ?
Proposer un nom de complexité pour l'algorithme de recherche dichotomique.

Q4 : Proposer une version itérative de la recherche dichotomique.

Correction.

Q1 :

```
1 def recherche(L,x):
2     for e in L:
3         if x == e:
4             return True
5     return False
```

Q2 :

```

1 ### 1ere version avec slicing, peu économique en mémoire
2 def dichorec(L,x):
3     if len(L)==0:
4         return False
5     else:
6         m = len(L)//2
7         if x == L[m]:
8             return True
9         elif x < L[m]:
10            return dichorec(L[:m],x)
11        else:
12            return dichorec(L[m+1:],x)
13
14 #2eme version, plus économe en mémoire
15 def dichorec_ind(L,x,deb,fin):
16     if deb>=fin:
17         return False
18     else:
19         m = (deb+fin)//2
20         if x == L[m]:
21             return True
22         elif x < L[m]:
23             return dichorec_ind(L,x,deb,m)
24         else:
25             return dichorec_ind(L,x,m+1,fin)
26
27 def dichorec2(L,x):
28     return dichorec_ind(L,x,0,len(L))

```

Q3 : Estimation empirique de la complexité :

a)

```

1 import random as rd
2 def listetrie(n):
3     L=[rd.randint(0,100) for i in range(n)]
4     L.sort()
5     return L

```

b)

```

1 def recherche_compteur(L,x):
2     c=0
3     for e in L:
4         c+=1
5         if x == e:
6             return c
7     return c
8
9 def dichorec_c(L,x,c):
10    if len(L)==0:
11        return c
12    else:
13        m = len(L)//2
14        c+=1
15        if x == L[m]:
16            return c
17        elif x < L[m]:
18            return dichorec(L[:m],x,c)
19        else:
20            return dichorec(L[m+1:],x,c)
21
22 def dichorec_compteur(L,x,c):
23    return dichorec_c(L,x,0)

```

Q4 :

```

1 def dichoit(L,x):
2     deb = 0
3     fin = len(L)
4     while deb<fin:
5         m = (deb+fin)//2
6         if x == L[m]:
7             return True
8         elif x < L[m]:
9             fin = m
10        else:
11            deb = m+1
12    return False

```

2. Exponentiation rapide

Programmons l'algorithme d'exponentiation rapide.

Dans la suite, x est un nombre et n est un entier positif. On n'aura bien-sûr jamais recours à l'opérateur ****** dans les questions qui suivent !

Q1 : (*Échauffement*) Écrire une fonction `puissance(x,n)` qui renvoie la valeur de x à la puissance n de complexité linéaire.

Q2 : *Version récursive de l'exponentiation rapide :*
En vous basant sur la description du cours, écrire une fonction `expo_rec(x,n)` récursive de l'exponentiation rapide.

Q3 : *Estimation empirique de la complexité :*

a) Modifier les fonctions `puissance(x,n)` et `expo_rec(x,n)` de telle sorte qu'elle renvoie un compteur `c` qui compte le nombre de multiplication de chaque fonction. On appellera `puissance_complexite(x,n)` et `expo_rec_complexite(x,n)` ces nouvelles fonctions.

b) Comparer les résultats de `puissance_complexite(2,n)` et `expo_rec_complexite(2,n)` pour $n = 2^k$ où $k = 5, 10, 15, 20, 100, \dots$. Que pensez-vous de la complexité de l'exponentiation rapide ?

Q4 : Proposer une version itérative de l'exponentiation rapide.

Correction.

Q1 :

```
1 def puissance(x,n):
2     p=1
3     for i in range(1,n+1):
4         p = p*x #ou p*=x
5     return p
```

Q2 :

```
1 def expo_rec(x,n):
2     if n == 0:
3         return 1
4     else:
5         p = expo_rec(x,n//2)
6         if n%2 == 0:
7             return p*p
8         else:
9             return p*p*x
```

Q3 : *Estimation empirique de la complexité :*

a)

```

1 def puissance(x,n):
2     c=0
3     p=1
4     for i in range(1,n+1):
5         c+=1
6         p = p*x
7     return c
8
9 def expo_rec_c(x,n):
10    if n == 0:
11        return 0
12    else:
13        p = expo_rec(x,n//2)
14        if n%2 == 0:
15            return p+1
16        else:
17            return p+1

```

Q4 :

```

1 def expo_it(x,n):
2     N = n
3     X = x
4     p=1
5     while N > 0:
6         if N%2 == 1:
7             p = p*X
8             X = X*X
9             N = N//2
10    return p

```