

Algorithmes et structures de données avancées : TD 2Listes 1D - Temps d'exécution d'un algorithme $T(n)$ - recherche linéaire - recherche dichotomique**Exercice 2.1** *Boucles*

1. Ecrire un algorithme qui calcule la somme suivante et qui utilise une boucle :

$$\sum_{i=1}^n i$$

```
somme=0
i=1
while i<=n:
    somme=somme+i
    i=i+1
```

2. Déterminer le temps d'exécution $T(n)$ de votre algorithme.

Affectations	2+2n
Additions	2n
comparaisons	n+1
Total	5n+3

Le temps d'exécution est $T_{2.1.2}(n) = 5n + 3$

3. Ecrire maintenant un (très court) algorithme sans boucles qui calcule cette somme (notez que la somme est égale à $\frac{n(n+1)}{2}$).

```
somme=(n*(n+1))/2
```

4. Déterminer de nouveau le temps d'exécution $T(n)$ de votre algorithme.

Affectations	1
Additions	1
Multiplications	1
Divisions	3
Total	4

Le temps d'exécution est $T_{2.1.4}(n) = 4$

5. Prouvez par récurrence que les résultats sont identiques.

Une solution compréhensible se trouve par exemple sur

http://www.bmlo.ca/~gdube/preuves/preuves_eleves_g/lecon8/lecon.html

6. A votre avis quel algorithme est plus efficace (ou, autrement dit, a une meilleure complexité).

Evidemment, pour $\forall n \geq 1$, l'algorithme 2.1.2 est plus efficace car $T_{2.1.4}(n) = 4 \in O(1)$, comparé à $T_{2.1.1}(n) = 5n + 3 \in O(n)$.

Exercice 2.2 *Recherche linéaire*

1. Ecrire un algorithme qui initialise une liste qui s'appelle `liste` avec `n` éléments aléatoires compris entre 1 et 100.

```

import random
list = [0] * n
i=0
while i<n:
    x=random.randint(1,100)
    list[i] = x
    i=i+1
# plus court :
# list=[random.randint(1,100) for i in range(n)]

```

2. Ecrire un algorithme qui détermine s'il y a un élément x dans la liste.

```

resultat=False
i=0
while i<n && resultat==False:
    if list[i]==x:
        resultat=True
    i=i+1

```

3. Déterminer le temps d'exécution $T(n)$ de votre algorithme qui détermine s'il y a un élément x dans la liste.

Affectations	$n+3$
Additions	n
Comparaisons	$2n+2$
Total	$5n+5$

Le temps d'exécution est $T_{2.2.3}(n) = 4$

Exercice 2.3 Recherche dichotomique

Maintenant, vous pouvez supposer que la liste est triée par ordre croissant en utilisant la fonction suivante :

```

liste.sort()

```

1. Ecrire de nouveau un algorithme qui détermine s'il y a un élément x dans la liste, **en bénéficiant que la liste est triée**.

```

inf=0
sup=n-1
resultat=False
i=0
while inf<=sup and resultat==False:
    milieu = (inf+sup)/2
    if x<list[milieu]:
        sup=milieu-1
    elif x>list[milieu]:
        inf=milieu+1
    else:
        resultat=True

```

2. Déterminer le temps d'exécution $T(n)$ de votre algorithme.

C'est un algorithme de recherche dichotomique. En algorithmique, la dichotomie (du grec << couper en deux >>) est un processus itératif de recherche où à chaque étape l'espace de recherche est restreint à l'une de deux parties. Pour des grands valeurs de longueur de données n , cet algorithme est mieux que celui dessus.

Pour déterminer son temps maximal d'exécution, il faut tout d'abord déterminer le nombre d'itérations de la boucle dans le pire des cas.

Affectations	$2 \log_2 n + 4$
Additions/Soustractions/Divisions	$3 \log_2 n + 1$
Comparaisons	$4 \log_2 n + 2$
Total	$9 \log_2 n + 7$

Son temps maximal d'exécution est $T(n) = 7 \log_2 n + 5 \in O(\log n)$, sa complexité asymptotique est donc $O(\log n)$.

$$T_{2.3.2} = 7 \log_2 n + 7$$

3. Est-ce que votre algorithme est plus efficace que l'algorithme de l'exercice précédent ?

Oui, il est plus efficace car le $T_{2.3.2} \in O(\log n)$, comparé au $T_{2.2.3} \in O(n)$