Exo1:

On vous donne une liste d'entiers non trié L ainsi qu'un entier e. Ecrivez un programme retournant la valeur dans L la plus proche de e en utilisant une recherche séquentielle (sequential search).Exemple : L = [1, 2, 5, 8, 12, 16, 24, 56, 58, 63] et e=41. Résultat attendu : 56

Exo2:

On vous donne une liste d'entiers non trié L ainsi qu'un entier e. Ecrivez un programme retournant la valeur dans L la plus proche de e en utilisant une recherche binaire (binary search). Exemple : L = [1, 2, 5, 8, 12, 16, 24, 56, 58, 63] et e=41. Résultat attendu : 56

Exo3:

On vous donne une matrix ordrée m (sorted) ainsi qu'un élément l. Une matrix ordrée répond aux critères suivants :

- m[i][j]<=m[i+1][j] (une ligne va du plus petit au plus grand)
- m[i][j]<=m[i][j+1] (une colonne va du plus petit au plus grand)

Ecrivez un algorithme qui retourne la position de l'élément l dans m. Si l n'est pas présent dans m alors il faut retourner (-1, -1). Quelle est la complexité de votre algorithme ?
Exemple1 : si m=[[1,2,3,4],[4,5,7,8],[5,6,8,10],[6,7,9,11]] et que l=7. Nous souhaitons avoir la réponse (1,2) OU (3,1) (l'une des deux, pas besoin de retourner les deux résultats).
Exemple2 : si m=[[1,2],[3,4]] et que l=7. Nous souhaitons avoir la réponse (-1,-1) car 7 n'est pas dans la matrix m.
Astuce : nous avons vu comment parcourir une matrix dans une série précédente (pour trouver la valeur max et min).

Exo3c (difficile): Pouvez faire un algorithme qui a une complexité O(log n) ?
Astuce : vous n'êtes pas obligé de commencer aux coordonnées (0,0), prenez avantage du fait que les données sont triées et explorez la diagonale...

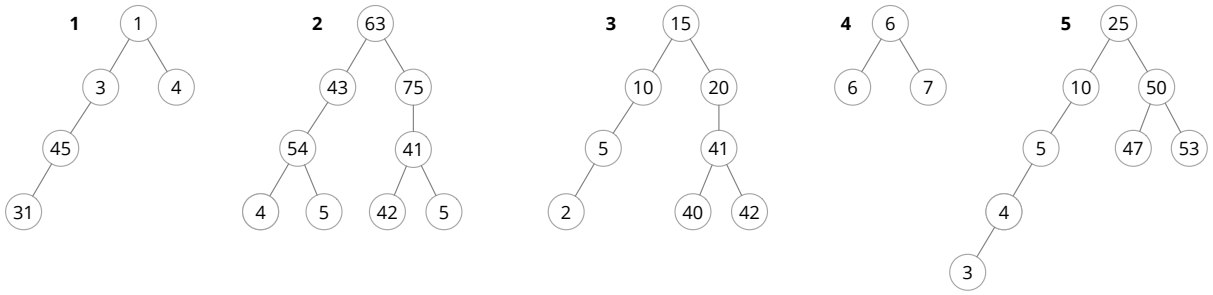
Exo4:

En tant que biologiste, on vous demande souvent de contrôler qu'un poisson existe dans une liste contenant tous les poissons existants. Cette liste est mise à jour chaque année. On vous aidant de la complexité des opérations de base des « Linked list » et des « Array list » disponible ci-dessous, utiliseriez-vous plutôt une « linked list » ou une « array list » ? Pourquoi ?

	Linked List	**Array List**
get(int index)	*O(n)*	*O(1)*
add(E element)	*O(1)*	*O(1)*
remove(int index)	*O(n)*	*O(n)*
Iterator.remove()	*O(1)*	*O(n)*
Iterator.add(E element)	*O(1)*	*O(n)*

Exo5:

Lesquels (ou lequel) de ces arbres est un arbre binaire (binary tree) ? Donnez leur hauteur (height).



Exo6:

Dessinez, à la main, un arbre binaire contenant les éléments suivants: [4 6 8 10 15 18 21]

Exo7:

On vous donne une liste d'entiers triés L ainsi qu'un entier e. Ecrivez un programme qui fait une recherche séquentielle pour retourner l'index de l'élément e dans L. Si e n'est pas dans L, retournez -1.
Exemple avec L = [1231321,3213125,3284016,4729273,5492710] et e = 3284016. Résultat attendu: 2

Exo8:

Même donnée que l'exercice précédant mais en utilisant une recherche binaire (binary search) .

Exo9:

On vous donne une liste d'entiers A et une liste d'entiers B de taille c. Insérez les éléments de B dans A, sachant que les c derniers éléments de A ont la valeurs None qui devront être remplacé par les éléments de B. Ecrivez un programme qui n'utilise que les opérations **pop** et **insert** pour modifier les listes.
Exemple:
a = [33, 44, 55, 66, 88, 99, None, None, None]
b = [11, 22, 77]
Résultat attendu: a = [33, 44, 55, 66, 88, 99, 11, 22, 77]

Exo10:

On souhaite former "des pics de montagne" et des "vallées" avec une liste d'entiers. Un pic est un élément qui est plus grand ou égale aux deux entiers adjacents dans la liste et une vallée est un élément qui est plus petit ou égale aux deux entiers adjacents. La liste d'entiers [5, 8, 7, 2, 3, 4, 6], 8 et 6 sont des pics et 5 et 2 sont des vallées. Ecrivez un programme qui forme des pics et des vallées en alternance à partir d'une liste d'entiers. Exemple avec [5, 3, 1, 2, 3] en entrée. Résultat attendu: [3, 5, 1, 3, 2]