# 1. Tri sélection

# 2. Tri insertion

# 3. Recherche dicotomique

# 4. Tri fusion

# 5. Arbres (POO)

# 6. Arbres (Listes)

# ------------------------------------------

# 1. Tri sélection

def tri\_selection(tab):

n = len(tab)

for i in range(n - 1):

ind\_min = i

for j in range(i + 1, n):

if tab[j] < tab[ind\_min]:

ind\_min = j

tab[i], tab[ind\_min] = tab[ind\_min], tab[i]

# 2. Tri insertion

def tri\_insertion(tab):

n = len(tab)

for i in range(1, n):

cle = tab[i]

j = i - 1

while j >= 0 and cle < tab[j]:

tab[j + 1] = tab[j]

j = j - 1

tab[j + 1] = cle

# 3. Recherche dicotomique

def recherche(t, v, g ,d):

# recherche(t, v, g=0, d=len(t) - 1)

"""Renvoie un indice de la valeur v dans le tableau de nombre t[g--d] (?)

None sinon

t supoosé trié dans l'ordre croissant"""

if g <= d:

m = (g + d) // 2

if t[m] > v:

return recherche(t, v, g, m - 1)

elif t[m] < v:

return recherche(t, v, m + 1, d)

else:

return m

else:

return None

def recherche\_dico\_recur(t, v):

return recherche(t, v, 0, len(t)-1)

def recherche\_dico\_iteratif(t, v):

g = 0

d = len(t) - 1

while g <= d:

m = (g + d) // 2

if t[m] > v:

d = m - 1

elif t[m] < v:

g = m + 1

else:

return m

return None

# 4. Tri fusion

def fusion(l1, l2):

""" Renvoie un tableau trié contenant les valeurs de l1 et l2

rangés dans l'ordre croissant """

l3 = []

n1, n2 = len(l1), len(l2)

i, j = 0, 0

while i < n1 and j < n2:

if l1[i] <= l2[j]:

l3.append(l1[i])

i += 1

else:

l3.append(l2[j])

j += 1

while i < n1:

l3.append(l1[i])

i += 1

while j < n2:

l3.append(l2[j])

j += 1

return l3

def fusion\_V2(l1, l2):

""" Renvoie un tableau trié contenant les valeurs de l1 et l2,

supposés trié dans l'ordre croissant à l'aide de sentinelles """

n = len(l1) + len(l2)

l3 = [0] \* n

i, j = 0, 0

l1.append(float("inf"))

l2.append(float("inf"))

for k in range(n):

if l1[i] <= l2[j]:

l3[k] = l1[i]

i += 1

else:

l3[k] = l2[j]

j += 1

return l3

def fusion\_recur(l1, l2):

if l1 == []:

return l2

if l2 == []:

return l1

if l1[0] <= l2[0]:

return [l1[0]] + fusion\_recur(l1[1:], l2)

else:

return [l2[0]] + fusion\_recur(l1, l2[1:])

# 5. Arbres (POO)

class Node:

def \_\_init\_\_(self, v, g=None, d=None):

self.val = v

self.fg = g

self.fd = d

tree1 = Node(7, Node(3, Node(2, Node(1), Node(5))), Node(4, Node(6), Node(8)))

def taille(arbre):

if arbre is None:

return 0

return 1 + taille(arbre.fg) + taille(arbre.fd)

def hauteur(arbre):

if arbre is None:

return 0

return 1 + max(hauteur(arbre.fg), hauteur(arbre.fd))

def parcours\_prefixe(arbre):

if arbre is None:

return None

print(arbre.val, end=" - ")

parcours\_prefixe(arbre.fg)

parcours\_prefixe(arbre.fd)

def parcours\_infixe(arbre):

if arbre is None:

return None

parcours\_infixe(arbre.fg)

print(arbre.val, end=" - ")

parcours\_infixe(arbre.fd)

def parcours\_postfixe(arbre):

if arbre is None:

return None

parcours\_postfixe(arbre.fg)

parcours\_postfixe(arbre.fd)

print(arbre.val, end=" - ")

print(taille(tree1))

print(hauteur(tree1))

print(parcours\_prefixe(tree1))

print()

print(parcours\_infixe(tree1))

print()

print(parcours\_postfixe(tree1))

# 6. Arbres (Listes)

tree1 = [7,[3,[2,[1,[],[]],[5,[],[]]],[]],[4,[6,[],[]],[8,[],[]]]]

def taille(arbre):

if arbre == []:

return 0

return 1 + taille(arbre[1]) + taille(arbre[2])

def hauteur(arbre):

if arbre == []:

return 0

return 1 + max(hauteur(arbre[1]), hauteur(arbre[2]))

def parcours\_prefixe(arbre):

if arbre == []:

return None

print(arbre[0], end=" - ")

parcours\_prefixe(arbre[1])

parcours\_prefixe(arbre[2])

def parcours\_infixe(arbre):

if arbre == []:

return None

parcours\_infixe(arbre[1])

print(arbre[0], end=" - ")

parcours\_infixe(arbre[2])

def parcours\_postfixe(arbre):

if arbre == []:

return None

parcours\_postfixe(arbre[1])

parcours\_postfixe(arbre[2])

print(arbre[0], end=" - ")

print(taille(tree1))

print(hauteur(tree1))

print(parcours\_prefixe(tree1))

print()

print(parcours\_infixe(tree1))

print()

print(parcours\_postfixe(tree1))