# 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

del

def

else:

return None

recherche\_dico\_recur(t, v):

return recher c he(t, v, 0, ken (t) - 1)

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(11, 12):

””” Renvoie un tableau trié contenant les valeurs de 11 et 12 rangés dans l'ordre croissant ”””

13 []

n1, n2 = len(11), len(12) i, j = 0, 0

while i ‹ n1 and j < n2: if 11[i] ‹= 12[j]:

13.append(11[i]) i *+--* 1

else:

13.append(12[j]) j += 1

while i ‹ n1:

13.append(11[i]) i += 1

while j ‹ n2:

13.append(12[j]) j += 1

return 13

def fusion\_V2(11, 12):

””” Renvoie un tableau trié contenant les valeurs de 11 et 12, supposés trié dans l'ordre croissant à l'aide de sentinelles "””

n = len(11) + len(12) 13 = [0] \* n

i, j = 0, 0

1. append(float(”inf"))
2. append(float(”inf")) for l‹ in range(n) :

if 11[i] <= 12[j]:

13[k] = 11[i] i += 1

else:

13[k] = 12[j] j += 1

return 13

def fusion\_recur(11, 12): if 11 == []:

return 12 if 12 == []:

return 11

if 11[0] <= 12[0]:

return [11[0]] + fusion\_recur(11[1:], 12) else:

return [12[0]] + fusion recur(11, 12[1:])

# 5. Arbres (POO) class Node:

def init (self, v, g=None, d=None) : self.val = v

self.fg = g self.fd = d

t ree1 = No de (7, No de ( 3, No de ( 2, No de (1) , Node ( 5) ) ) , No de (4, No de ( 6) , Node (8) ) )

def taille(arbre): if arbre is None:

return 0

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

del hauteur (arbre):

if- a rbre 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(treel)) print(hauteur(treel)) print(parcours prefixe(tree1)) print() print(parcours\_infixe(treel)) print()

print(parcours postfixe(tree1))

# 6. Arbres ( Li stes)

treel [7,[3,[2,[1,[],[]],[5,[],[]]],[]],[4,[6,[],[]],[8,[],[]]]]

def taille(arbre): if arbre == []:

retur

t

U ’ O aille(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])

deb parcours\_post f-:txe (arbre):

if arbre == []: return None

parcours postfixe(arbre[1])

parcours postfixe(arbre[2]) print(arbre[0], end=” - ”)

print(taille(treel)) print(hauteur(treel)) print(parcours prefixe(treel)) print() print(parcours\_infixe(treel)) print()

print(parcours postfixe(treel))