Algorithmes au programme du cycle terminal

1. Préliminaires

```python
import random

# création d'un tableau de n entiers aléatoires compris entre 0 et N

def creation_tableau(N,n):
    tableau = []
    for i in range(n):
        tableau.append(random.randint(0,N))
    return tableau


T=creation_tableau(100,15)
```

2. Recherche d'une occurrence dans un tableau

a. en impératif

```python
def recherche_occurrence(tableau, element):
    """
    recherche l'indice d'une occurrence d'un élément dans un tableau
    :param tableau: (list) un tableau
    :param element:(any)
    :return: (int)
    :CU: elt appartient à tableau
    """
    indice = 0
    while indice != len(tableau) and tableau[indice] != element:
        indice += 1
    return indice
```

b. en récursif

```python
def recherche_occurrence_recursive(tableau, element):
    """
    recherche l'indice d'une occurrence d'un élément dans un tableau
    :param tableau: (list) un tableau
    :param element:(any)
    :return: (int)
    :CU: elt appartient à tableau
    """
    if tableau[0] == element:
        return 0
    else:
        return 1 + recherche_occurrence_recursive(tableau[1:], element)
```

c. en récursif terminal

```python
def recherche_occurrence_recursive_terminale(tableau, element, indice = 0):
    """
    recherche l'indice d'une occurrence d'un élément dans un tableau
    :param tableau: (list) un tableau
    :param element:(any)
    :return: (int)
    :CU: elt appartient à tableau
    """
    if tableau[0] == element:
        return indice
    else:
        return recherche_occurrence_recursive_terminale(tableau[1:], element, indice+1
```

3. Recherche d'un maximum

a. en impératif

```python
def recherche_max(tableau):
    """
    recherche le maximum parmi les valeurs d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (int)
    """
    maxi = tableau[0]
    for i in range(1,len(tableau)):
        if tableau[i] > maxi:
            maxi = tableau[i]
    return maxi
```

b. en récursif

```python
def recherche_max_recursive(tableau):
    """
    recherche le maximum parmi les valeurs d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (int)
    """
    if len(tableau)==1:
        return tableau[0]
    else:
        if tableau[0] > recherche_max_recursive(tableau[1:]):
            return tableau[0]
        else:
            return recherche_max_recursive(tableau[1:])
```

c. en récursif terminal

```python
def recherche_max_recursive_terminale(tableau, maxi = None):
    """
    recherche le maximum parmi les valeurs d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (int)
    """
    if maxi == None:
        maxi = tableau[0]
    if len(tableau) == 0:
        return maxi
    else:
        if tableau[0] > maxi:
            maxi = tableau[0]
        return recherche_max_recursive_terminale(tableau[1:], maxi)
```

4. Recherche d'un minimum

a. en impératif

```python
def recherche_min(tableau):
    """
    recherche le minimum parmi les valeurs d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (int)
    """
    mini = tableau[0]
    for i in range(1,len(tableau)):
        if tableau[i] < mini:
            mini = tableau[i]
    return mini
```

b. en récursif

```python
def recherche_min_recursive(tableau):
    """
    recherche le minimum parmi les valeurs d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (int)
    """
    if len(tableau)==1:
        return tableau[0]
    else:
        if tableau[0] < recherche_min_recursive(tableau[1:]):
            return tableau[0]
        else:
            return recherche_min_recursive(tableau[1:])
```

c. en récursif terminal

```python
def recherche_min_recursive_terminale(tableau, mini = None):
    """
    recherche le minimum parmi les valeurs d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (int)
    """
    if mini == None:
        mini = tableau[0]
    if len(tableau) == 0:
        return mini
    else:
        if tableau[0] < mini:
            mini = tableau[0]
        return recherche_min_recursive_terminale(tableau[1:], mini)
```

5. Recherche dichotomique dans un tableau trié

a. en impératif

Python

```python
def recherche_dicho(tableau, element):
    """
    recherche un élément dans un tableau d'entiers triés
    :param tableau: (list) un tableau
    :param element: (int) un entier
    :CU: tableau doit être trié
    :return: (int) -1 si l'element n'est pas dans le tableau, l'indice d'une occurrenc
    """
    indice_gauche = 0
    indice_droit = len(tableau)-1
    indice_milieu = (indice_gauche + indice_droit) // 2
    while element != tableau[indice_milieu] and indice_gauche < indice_droit:
        if element < tableau[indice_milieu]:
            indice_droit = indice_milieu -1
        else:
            indice_gauche = indice_milieu + 1
        indice_milieu = (indice_gauche + indice_droit) // 2
    if element == tableau[indice_milieu]:
        return indice_milieu
    else:
        return -1
```

b. en récursif

```python
def recherche_dicho_recursive(tableau, element, indiceGauche = 0,
                indiceDroit = None):
    """
    recherche un élément dans un tableau d'entiers triés
    :param tableau: (list) un tableau
    :param element: (int) un entier
    :return: (int) -1 si l'element n'est pas dans le tableau, l'indice d'une occurrenc
    :CU: tableau doit être trié, element doit appartenir à tableau
    """
    if indiceDroit == None:
        indiceDroit = len(tableau)-1
    indiceMilieu = (indiceGauche + indiceDroit) // 2
    if indiceDroit < indiceGauche :
        return -1
    elif element == tableau[indiceMilieu]:
        return indiceMilieu
    else:
        if element < tableau[indiceMilieu]:
            return recherche_dicho_recursive(tableau, element, indiceGauche,
                    indiceMilieu - 1)
        else:
            return recherche_dicho_recursive(tableau, element, indiceMilieu + 1,
                    indiceDroit)
```

c. en récursif terminal

```python
def recherche_dicho_recursive_terminale(tableau, element, indiceGauche = 0,
            indiceDroit = None, acc = -1):
    """
    recherche un élément dans un tableau d'entiers triés
    :param tableau: (list) un tableau
    :param element: (int) un entier
    :return: (int) -1 si l'element n'est pas dans le tableau, l'indice d'une occurrenc
    :CU: tableau doit être trié
    """
    if indiceDroit == None:
        indiceDroit = len(tableau)-1
    indiceMilieu = (indiceGauche + indiceDroit) // 2
    if indiceDroit < indiceGauche :
        return acc
    elif element == tableau[indiceMilieu]:
        acc = indiceMilieu
        return acc
    else:
        if element < tableau[indiceMilieu]:
            return recherche_dicho_recursive_terminale(tableau, element, indiceGauche,
                    indiceMilieu - 1, acc)
        else:
            return recherche_dicho_recursive_terminale(tableau, element, indiceMilieu
                    indiceDroit, acc)
```

6. Calcul de la somme des éléments d'un tableau

a. en impératif

```python
def somme(tableau):
    """
    recherche la somme des elements d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (int)
    """
    somme = 0
    for elt in tableau:
        somme += elt
    return somme
```

b. en récursif

```python
def somme_recursive(tableau):
    """
    recherche la somme des elements d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (int)
    """
    if len(tableau)==0:
        return 0
    else:
        return tableau[0] + somme_recursive(tableau[1:])
```

c. en récursif terminal

```python
def somme_recursive_terminale(tableau, somme = 0):
    """
    recherche la somme des elements d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (int)
    """
    if len(tableau) == 0:
        return somme
    else:
        return somme_recursive_terminale(tableau[1:], somme + tableau[0])
```

7. Calcul de la moyenne des éléments d'un tableau

a. en impératif

```python
def moyenne(tableau):
    """
    recherche la moyenne des elements d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (float)
    """
    somme = 0
    for elt in tableau:
        somme += elt
    return somme/len(tableau)
```

b. en récursif

```python
def moyenne_recursive(tableau):
    """
    recherche la moyenne des elements d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (float)
    """
    if len(tableau) == 0:
        return 0
    else:
        return ((len(tableau)-1)*moyenne_recursive(tableau[1:])+tableau[0])/len(tablea
```

c. en récursif terminal

```python
def moyenne_recursive_terminale(tableau, moyenne=0, n=None):
    """
    recherche la moyenne des elements d'un tableau d'entiers
    :param tableau: (list) un tableau
    :return: (float)
    """
    if n == None:
        n = len(tableau)
    if len(tableau) == 0:
        return moyenne
    else:
        return moyenne_recursive_terminale(tableau[1:], (n*moyenne+tableau[0])/n,n)
```

8. Tri par sélection

a. la fonction échange, nécessaire au tri par sélection

```python
def echange(tableau, i, j):
    """
    echange les éléments d'indice i et j d'un tableau
    :param tableau: (list) un tableau
    :param i:(int) un entier
    :param j:(int) un entier
    :CU: 0 <= i < len(tableau)
        0 <= j < len(tableau)
    :return: (None)
    :Side-effects: Le tableau est modifié
    """
    tableau[i], tableau[j] = tableau[j], tableau[i]
```

b. en impératif

```python
def tri_selection(tableau):
    """
    trie un tableau
    :param tableau: (list) un tableau
    :return: (list) le tableau trié
    :Side-effects: Le tableau est modifié
    """
    for i in range(len(tableau)):
        indice_mini = i
        for j in range(i+1, len(tableau)):
            if tableau[indice_mini] > tableau[j]:
                indice_mini = j
        echange(tableau, i, indice_mini)
    return tableau
```

c. en récursif

```python
def tri_selection_recursive(tableau):
    """
    trie un tableau
    :param tableau: (list) un tableau
    :return: (list) le tableau trié
    :Side-effects: Le tableau est modifié
    """
    if len(tableau) <= 1:
        return tableau
    else:
        indice_mini = 0
        for i in range(1, len(tableau)):
            if tableau[indice_mini] > tableau[i]:
                indice_mini = i
        echange(tableau, 0, indice_mini)
        return [tableau[0]]+tri_selection_recursive(tableau[1:])
```

d. en récursif terminal

```python
def tri_selection_recursive_terminal(tableau, resultat=[]):
    """
    trie un tableau
    :param tableau: (list) un tableau
    :return: (list) le tableau trié
    :Side-effects: Le tableau est modifié
    """
    if len(tableau) == 0:
        return resultat
    else:
        indice_mini = 0
        for i in range(1, len(tableau)):
            if tableau[indice_mini] > tableau[i]:
                indice_mini = i
        echange(tableau, 0, indice_mini)
        return tri_selection_recursive_terminal(tableau[1:], resultat + [tableau[0]])
```

9. Tri par insertion

a. la fonction insertion, nécessaire au tri par insertion

```python
def insertion(tableau, element):
    """
    insère un élément dans un tableau d'entiers triés
    :param tableau: (list) un tableau
    :param element: (int) un entier
    :CU: tableau doit être trié
    :return: (None)
    :Side-effects: Le tableau est modifié
    """
    indice = 0
    while len(tableau) != 0 and indice != len(tableau) and tableau[indice] < element:
        indice += 1
    return tableau[:indice]+[element]+tableau[indice:]
```

b. en impératif

```python
def tri_insertion(tableau):
    """
    trie un tableau
    :param tableau: (list) un tableau
    :return: (list) le tableau trié
    """
    resultat = []
    for i in range(len(tableau)):
        resultat = insertion(resultat, tableau[i])
    return resultat
```

10. Tri fusion

a. en récursif - méthode diviser pour régner

```python
def tri_fusion(liste, debut=0, fin=None):
    """
    tri la liste mise en paramètre entre les indice debut et fin
    :param liste:(list) une liste
    :param debut:(int) un entier, par défaut 0
    :param fin:(int ou NoneType) un entier, par défaut None
    :CU: debut>=0, fin>=0
    """
    if fin == None:
        fin = len(liste)-1
    if debut<fin:
        #Diviser le problème
        milieu=(debut+fin)//2
        #Résoudre chaque sous-problème
        tri_fusion(liste,debut,milieu)
        tri_fusion(liste,milieu+1,fin)
        #combine les solutions des sous-problèmes
        fusion(liste,debut,milieu,fin)

def fusion(liste,debut,pivot,fin):
    """
    Replace les éléments de la liste compris entre les indices debut et fin dans l'ord
    :param liste:(list) une liste
    :param debut:(int) un entier
    :param pivot:(int) un entier
    :param fin:(int ou NoneType) un entier
    :CU: debut>=0, fin>=0, pivot>=0, debut<fin
    """
    gauche=[]
    droite=[]
```

```
    for indice in range(debut,pivot+1):
        gauche.append(liste[indice])
    for indice in range(pivot+1,fin+1):
        droite.append(liste[indice])
    i=0
    j=0
    for k in range(debut,fin+1):
        if i==len(gauche) and j<len(droite):
            liste[k]=droite[j]
            j+=1
        elif j==len(droite) and i<len(gauche):
            liste[k]=gauche[i]
            i+=1
        elif gauche[i]<=droite[j]:
            liste[k]=gauche[i]
            i+=1
        else:
            liste[k]=droite[j]
            j+=1
```