TD_complexite_parcours_sequentiel

April 26, 2021

1 Exercice 1

On rappelle que la complexité temporelle correspond au nombre d'opérations élémentaires effectuées pour accomplir la tâche

[ ]:
```
def recherche(liste, valeur):
    """
    Description de la fonction : Détermine si la valeur est présente dans la␣
 ↪liste
    paramètre liste (list)
    paramètre valeur (type quelconque)
    Return (bool)
    """
    for element in liste:
        if element == valeur:
            return True
    return False
```

1. Calculer la complexité temporelle de l'appel ci-dessous (meilleur cas)

[ ]:
```
liste = [12,6,3,8,14,5]
recherche(liste,12)
```

2. Calculer la complexité temporelle de l'appel ci-dessous (cas quelconque)

[ ]:
```
liste = [12,6,3,8,14,5]
recherche(liste,8)
```

3. Calculer la complexité temporelle de l'appel ci-dessous (pire cas)

[ ]:
```
liste = [12,6,3,8,14,5]
recherche(liste,20)
```

Généralisation dans le pire cas : Soit une liste contenant `n` éléments ne contenant pas l'élément `20`

4. Calculer la complexité temporelle de l'appel ci-dessous en fonction de `n`

[ ]:
```
recherche(liste,20)
```

2 Exercice 2

On donne la fonction `longueur` ci-dessous mettant en oeuvre 1 algorithme de **parcourt séquentiel** permettant de calculer le nombre des éléments dans une liste

[ ]:
```
def longueur(liste):
    compteur = 0
    for element in liste:
        compteur = compteur + 1
    return compteur
```

Soit une liste contenant `n` éléments. Calculer la complexité temporelle en fonction de `n` de cet algorithme