C9 Les Dictionnaires - Révision

Contenus	Capacités attendues	Commentaires
Dictionnaires, index et clé.	Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.	

Ref1, Ref2

1. Révision

- Un dictionnaire est une structure de donnée gardant en mémoire des informations sous la forme (clé, valeur).
- Le but d'un dictionnaire est d'être capable d'accéder rapidement à une valeur à partir de sa clé.

1.1. Créer un dictionnaire

Il y a deux manière de créer un dictionnaire vide.

🐍 **Script Python**

```
diconul = {}
print(diconul)
```

```
{}
```

ou

🐍 **Script Python**

```
diconul = dict()
print(diconul)
```

```
{}
```

On peut créer un dictionnaire

- directement

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
print("dico1: ", dico1)
```

dico1: {'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}

- champ par champ

🐍 **Script Python**

```
dico2 = {} # Un dictionnaire vide
dico2["nom"] = "simpson"
dico2["prénom"] = "Homer"
dico2["année"] = 1989
print("dico2: ", dico2)
```

dico2: {'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}

- avec la méthode **`update()`**

🐍 **Script Python**

```
dico3 = {}
dico3.update({"nom": "simpson",'prénom': 'Homer','année': 1989})
print("dico3: ", dico3)
```

dico3: {'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}

- À partir d'une liste de liste ou d'une liste de tuple

🐍 **Script Python**

```
#liste = [("nom", "simpson"), (["prénom", "Homer"]), (["année", 1989])]
liste = [["nom", "simpson"], ["prénom", "Homer"], ["année", 1989]]
dico4 = dict(liste)
print("dico4: ", dico4)
```

dico4: {'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}

- À partir de deux listes (ou tuple)

🐍 **Script Python**

```
liste_keys = ["nom", "prénom", "année"]
liste_values = ["simpson", "Homer", 1989]
# On utilise une compréhension de liste
dico5 = {key: value for key, value in zip(liste_keys,liste_values)}
print("dico5: ", dico5)
```

dico5: {'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}

1.2. Lire et modifier des valeurs des valeurs

🐍 **Script Python**

```
print(dico1["nom"])
```

simpson

🐍 Script Python

```
"surnom" in dico1.keys() # teste si "surnom" est une clé
# identique à "surnom" in dico1 # teste si "surnom" est une clé
```

False

🐍 Script Python

```
"Homer" in dico1.values() # teste si Homer est une valeur
True
```

True

🐍 Script Python

```
dico1["année"]=1990    # On change l'année de naissance
dico1
```

{'nom': 'simpson', 'prénom': 'Homer', 'année': 1990}

1.3. Parcourir un dictionnaire

- Par clés

🐍 Script Python

```
for cle in dico1.keys():
    # ou plus simplement for key in dico1:
    print(f"la clé est {cle} et sa valeur est {dico1[cle]}.")


la clé est nom et sa valeur est simpson.
la clé est prénom et sa valeur est Homer.
la clé est année et sa valeur est 1990.
```

- par valeurs

🐍 Script Python

```
for valeur in dico1.values():
    print(valeur)

simpson
Homer
1990
```

- Par couple (clé, valeurs)

🐍 Script Python

```
for key,value in dico1.items():
    print(f"La clé est {key} et sa valeur {value}")

La clé est nom et sa valeur simpson
La clé est prénom et sa valeur Homer
La clé est année et sa valeur 1990
```

1.4. Suppression d'un élément

- Méthode **pop()**
  Cette méthode retourne la valeur

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
val=dico1.pop('nom')
print(dico1)
print(val)

{'prénom': 'Homer', 'année': 1989}
simpson
```

- Méthode **popitem()**
  Cette méthode retourne le tuple (clé, valeur).

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
print(dico1)
val=dico1.popitem()
print(dico1)
val=dico1.popitem()
print(val)
print(dico1)


{'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}
{'nom': 'simpson', 'prénom': 'Homer'}
('prénom', 'Homer')
{'nom': 'simpson'}
```

- Instruction **del**

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
del dico1["prénom"]
dico1

{'nom': 'simpson', 'année': 1989}
```

1.5. Les méthodes

- **`.copy()`** : Copie d'un dictionnaire car une variable contient l'adresse d'un dictionnaire et non le dictionnaire lui même.

🐍 Script Python

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
dico2 = dico1
dico1['prénom']='lisa'
print(dico1)
print(dico2)
```

```
{'nom': 'simpson', 'prénom': 'lisa', 'année': 1989}
{'nom': 'simpson', 'prénom': 'lisa', 'année': 1989}
```

Attention comme pour les liste quand on écrit `dico2 = dico1`, c'est l'emplacement de la mémoire du dictionnaire dico1 qui est copié et non le dictionnaire.

🐍 Script Python

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
dico2 = dico1.copy()
dico1['prénom']='lisa'
print(dico1)
print(dico2)
```

```
{'nom': 'simpson', 'prénom': 'lisa', 'année': 1989}
{'nom': 'simpson', 'prénom': 'Homer', 'année': 1989}
```

- **`.pop(key[, default])`** : Supprime key si key est dans le dictionnaire sinon renvoie default.
- **`.popitem()`** : Supprime et renvoie une paire (key, value) du dictionnaire.

🐍 Script Python

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
var=dico1.pop('nom')
print(var)
print(dico1)
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
var=dico1.popitem()
print(var)
print(dico1)
```

```
simpson
{'prénom': 'Homer', 'année': 1989}
('année', 1989)
{'nom': 'simpson', 'prénom': 'Homer'}
```

🐍 Script Python

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
var=dico1.pop('surnom')
```

```
print(var)


Traceback (most recent call last):
  File "<input>", line 2, in <module>
KeyError: 'surnom'
```

La clé **surnom** n'existe pas et il y a une erreur.
Pour gérer les erreurs on utilise la condition **try ... except**

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
try:
  var=dico1.pop('surnom')
  print(var)
except:
  print("Il n'y a pas de clé surnom")


Il n'y a pas de clé surnom
```

On peut aussi utiliser le deuxième argument de la méthode **.pop()**

🐍 **Script Python**

```
dico1 = {"nom": "simpson", "prénom": "Homer", "année": 1989}
var=dico1.pop('surnom',"Il n'y a pas surnom")
print(var)

var=dico1.pop('nom',"Il n'y a pas nom")
print(var)


Il n'y a pas surnom
simpson
```

Dans tous les cas utilisez la fonction **help()** pour en savoir plus

🐍 **Script Python**

```
dico= {"nom": "simpson", "prénom": "Homer", "année": 1989}
help(dico.pop)
```

🐍 **Script Python**

```
  Help on built-in function pop:

  pop(...) method of builtins.dict instance
    D.pop(k[,d]) -> v, remove specified key and return the corresponding value.

    If the key is not found, return the default if given; otherwise,
    raise a KeyError.
```

- **.keys()** Les clés.

- **`.values()`** Les valeurs.
- **`.items()`** (clés, valeurs).

2. TP

Exo

Enoncé Solution

Ecrire une fonction `dicoCarre(n)` programme Python qui permet de créer à partir d'un entier n , un dictionnaire formé des entiers de 1 à n et de leurs carrées.

Exemple pour n = 7 le dictionnaire sera de la forme:

Script Python

```
>>>dicoCarre(7)
{1: 1, 2: 4, 3: 9, 4: 16,
5: 25 , 6: 36 , 7: 49}
```

Script Python

```
def creer_dico(n:int)->dict:
    dico={}
    for k in range(1,n+1):
        dico[k]=k**2
    return dico

dcarre=creer_dico(10)
print(dcarre)
```

Script Python

```
assert  creer_dico(7) == {1: 1, 2: 4, 3: 9, 4: 16, 5: 25 , 6: 36 , 7: 49}
```

Exo

Enoncé **Solution**

Écrire une fonction `dicoPaireImpaire(L)` en Python qui prends en paramètre une liste de nombres entiers et qui renvoie un dictionnaire dont les clés sont les entiers de la liste et dont les valeurs sont `'pair'` ou `'impair'` selon la parité du nombre.

🐍 **Script Python**

```
def dicoPaireImpaire(L):
    dico={}
    for elt in L:
        if elt%2==0:
            dico[elt]='pair'
        else:
            dico[elt]='impair'
    return dico

dicoPaireImpaire([1,5,7,8,9,12,15,456,87])
```

Exo

On considère trois dictionnaires Pythons qui regroupe la totalité du matériels informatiques. Écrire une fonction `concaDico(dico1,dico2,dico3)` Python qui regroupe en concaténant ces trois dictionnaires en un seul et le renvoie.

Exemple:

🐍 **Script Python**

```
dicPC={"HP": 11 , "Acer": 7 , "Lenovo": 17 ,
"Del": 23}
dicPhone={"Sumsung": 22 , "Iphone": 9 ,
"Other": 13 }
dicTablette = {"Sumsung": 15 , "Other": 13}
>>>concaDico(dicPC,dicPhone,dicTablette)
{"PC HP": 11 , "PC Acer": 7 , "PC Lenovo": 17 ,
"PC Del": 23,"Phone Sumsung": 22 , "Phone
Iphone": 9 , "Phone Other": 13, "Tablette
Sumsung": 15 , "Tablette Other": 13 }
```

🐍 **Script Python**

```
#Version en une fonction

def concaDico(dico1,dico2,dico3):
    dicTotal={}
    for cle,val in dico1.items():
        if cle in dicTotal:
            dicTotal["PC"+" "+cle]+=val
        else:
            dicTotal["PC"+" "+cle]=val
    for cle,val in dico2.items():
        if cle in dicTotal:
            dicTotal["Phone"+" "+cle]+=val
        else:
            dicTotal["Phone"+" "+cle]=val
    for cle,val in dico3.items():
        if cle in dicTotal:
            dicTotal["Tablette"+" "+cle]+=val
        else:
            dicTotal["Tablette"+" "+cle]=val

    return dicTotal

dicPC={"HP": 11 , "Acer": 7 , "Lenovo": 17 ,
"Del": 23}
dicPhone={"Sumsung": 22 , "Iphone": 9 ,
"Other": 13 }
```

```
dicTablette = {"Sumsung": 15 , "Other": 13}

dicTotal=concaDico(dicPC,dicPhone,dicTablette)

print(dicTotal)
```

`{'PC HP': 11, 'PC Acer': 7, 'PC Lenovo': 17, 'PC Del': 23, 'Phone Sumsung': 22, 'Phone Iphone': 9, 'Phone Other': 13, 'Tablette Sumsung': 15, 'Tablette Other': 13}`

🐍 **Script Python**

```
#Version en deux fonctions

def ajout(dico,dicTotal,nom):
    for cle,val in dico.items():
        if cle in dicTotal:
            dicTotal[nom+" "+cle]+=val
        else:
            dicTotal[nom+" "+cle]=val
    return dicTotal


def concaDico(dico1,dico2,dico3):
    dicTotal={}
    ajout(dico1,dicTotal,"PC")
    ajout(dico2,dicTotal,'Phone')
    ajout(dico3,dicTotal,"Tablette")
    return dicTotal

dicPC={"HP": 11 , "Acer": 7 , "Lenovo": 17 ,
"Del": 23}
dicPhone={"Sumsung": 22 , "Iphone": 9 ,
"Other": 13 }
dicTablette = {"Sumsung": 15 , "Other": 13}

dicTotal=concaDico(dicPC,dicPhone,dicTablette)
```

Ecrire une fonction qui regroupe et calcule la somme des apareils par marque.

🐍 **Script Python**

```
dicPC = {"HP": 11 , "Acer": 7 , "Lenovo": 17 ,
"Del": 23}
dicPhone = {"Sumsung": 22 , "Iphone": 9 ,
"Other": 13 }
dicTablette = {"Sumsung": 15 , "Other": 13}
```

```python
def ajout(dico,dicTotal):
    for cle,val in dico.items():
        if cle in dicTotal:
            dicTotal[cle] += val
        else:
            dicTotal[cle] = val
    return dicTotal


def sommeDico(dico1,dico2,dico3):
    dicTotal={}
    ajout(dico1,dicTotal)
    ajout(dico2,dicTotal)
    ajout(dico3,dicTotal)
    return dicTotal

dicTotal=sommeDico(dicPC,dicPhone,dicTablette)
```

🐍 **Script Python**

```python
assert dicTotal ==  {'HP': 11, 'Acer': 7, 'Lenovo': 17, 'Del': 23, 'Sumsung': 37, 'Other': 26, 'Iphone': 9}
```

Exo

Enoncé **Solution**

1. Ecrire une fonction `diviseur(a)` qui prend un entrier `a` en argument et renvoie la liste de ses diviseurs.

Exemple:

🐍 Script Python

```
>>>diviseur(14)
[1,2,7,14]
```

1. Ecrire une fonction `dicoDiviseur(L)` en Python qui prend une liste de nombres entiers et renvoe un dictionnaire dont les clés sont les entiers saisis et dont les valeurs sont les listes des diviseurs des nombres saisis.

Exemple :

🐍 Script Python

```
>>>dicoDiviseur([2 , 7 , 11 , 5 , 3 , 19 , 14 , 9 , 1 , 4])
d = {2 : [1,2] , 7: [1,7] , 14: [1,2,7,14] , 9: [1,3,9] , 11: [1,11] , 5: [1,5]  , 3: [1,3]  , 19: [1,19]  , 1: [1] , 4: [1,2,4] }
```

🐍 Script Python

```
def diviseur(a):
    tab=[]
    for k in range(1,a+1):
        if a%k==0:
            tab.append(k)
    return tab


def dicoDiviseur(L):
    dico={}
    for elt in L:
        rep=diviseur(elt)
        dico[elt]=rep
    return dico

dicoDiviseur([2 , 7 , 11 ,
5 , 3 , 19 , 14 , 9 , 1 , 4])
```

Exo

Enoncé **Solution**

Écrire une fonction `dicoInverse(texte)` en Python qui prend un texte en argument et renvoye un dictionnaire dont les clés sont les mots du texte saisi et les valeurs sont les inverses des mots qui composent le texte.

Exemple :

🐍 **Script Python**

```
>>>dicoInverse("Python est un facile" )
{'Python': 'nohtyp' , 'est': 'tse' , 'facile': 'elicaf'}
```

🐍 **Script Python**

```
def decomposPhrase(phrase):
    mot=[phrase[0]]
    pos=0
    for i in range(1,len(phrase)):
        if phrase[i]==" ":
            mot.append(phrase[i])
            pos+=1
        else:
            mot[pos]+=phrase[i]
    return mot

print(decomposPhrase("Python est un facile" ))

def inverse_mot(mot):
    mot_inv=""
    for lettre in mot:
        mot_inv=lettre.lower()+mot_inv
    return mot_inv

print(inverse_mot("Python"))

def dicoInverse(phrase):
    dico={}
    for mot in decomposPhrase(phrase):
```

```
        dico[mot]=inverse_mot(mot)
    return dico

dicoInverse("Python est un facile" )
```

Exo

Enoncé Solution

Vous trouverez sur https://www.liste-de-mots.com/ liste des points de chaque lettres de l'alphabet.

Le module ***string*** permet d'obtenir les lettres de l'alphabet.

🐍 **Script Python**

```
import string
print(string.ascii_uppercase)
print(list(string.ascii_uppercase))
```

📋 **Texte**

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
'U', 'V', 'W', 'X', 'Y', 'Z']
```

1. Créer un dictionnaire `dico_scrabble` dont les clés sont lettres de l'alphabet et les valeurs sont 1
2. Copier et modifier alors le dictionnaire `dico_scrabble` dont les clés sont lettres de l'alphabet et les valeurs sont les points de ces lettres au scrabble.
3. Créer une fonction **`points_mot(mot:str)->int`** qui renvoie le nombre de points d'un mot

1.

🐍 **Script Python**

```
dico_scrabble = dict()  # ou {}
for i in range (65, 65+26):
    dico_scrabble[chr(i)] = 1
```

🐍 **Script Python**

```
assert dico_scrabble == {'A': 1, 'B':
1, 'C': 1, 'D': 1, 'E': 1, 'F': 1, 'G': 1, 'H':
1, 'I': 1, 'J': 1, 'K': 1, 'L': 1,
                    'M': 1, 'N': 1, 'O': 1,
'P': 1, 'Q': 1, 'R': 1, 'S': 1, 'T': 1, 'U': 1,
'V': 1, 'W': 1, 'X': 1, 'Y': 1, 'Z': 1}
```

1. 🐍 **Script Python**

   ```
   dico_scrabble = {'A': 1, 'B': 3,
   'C': 3, 'D': 2, 'E': 1, 'F': 4, 'G': 2,
   'H': 4, 'I': 1, 'J': 8, 'K': 10, 'L': 1,
   'M': 2,
                'N': 1, 'O': 1, 'P': 3, 'Q':
   8, 'R': 1, 'S': 1, 'T': 1, 'U': 1, 'V': 4,
   'W': 10, 'X': 10, 'Y': 10, 'Z': 10, '*':
   0}
   ```

🐍 **Script Python**

```
assert  dico_scrabble['A'] == 1
assert  dico_scrabble['Z'] == 10
assert  dico_scrabble['*'] == 0
```

1.

🐍 **Script Python**

```
def points_mot(mot:str)->int:
    pt = 0
    for i in mot:
        pt += dico_scrabble[i.upper()]
    return pt
```

🐍 **Script Python**

```
assert points_mot('YACK') == 24
assert points_mot('yack') == 24
assert points_mot('NERUDA') == 7
assert points_mot('Neruda') == 7
```

Exo

Enoncé **Solution**

On considère le dictionnaire suivant dont les clés sont les noms des élèves et les valeurs des clés sont les moyennes générales obtenues en passant l'examen final:

🐍 Script Python

```
lyceens =
{"lyceen_1": 13,
"lyceen_2": 17,
"lyceen_3": 9,
"lyceen_4": 15,
"lyceen_5": 8,
      "lyceen_6": 14,
"lyceen_7": 16,
"lyceen_8": 12,
"lyceen_9": 13,
"lyceen_10": 15,
      "lyceen_11": 14,
"lyceen_112": 9,
"lyceen_13": 10,
"lyceen_14": 12,
"lyceen_15": 13,
      "lyceen_16": 7,
"lyceen_17": 12,
"lyceen_18": 15,
"lyceen_19": 9,
"lyceen_20": 17}
```

Écrire un programme Python qui partitionne ce dictionnaire en deux sous dictionnaires:

1. `etudiantAdmis` dont les clés sont les étudiants admis et

les valeurs des clés sont les moyennes obtenues (moyenne supérieurs ou égales à 10).

2. `etudiantNonAdmis` dont les clés sont les étudiants non admis et les valeurs des clés sont les moyennes obtenues (moyenne strictement inférieur à 10).

🐍 Script Python

```
# on crée  deux dictionnaires vides un pour les admis et l'autre pour les non admis
# on crée  deux dictionnaires vides un pour les admis et l'autre pour les non admis
etudiantAdmis = dict()
etudiantNonAdmis = dict()
for k,v in lyceens.items():
    if v < 10:
etudiantNonAdmis[k] = v
    else:
etudiantAdmis[k] = v
```

🐍 Script Python

```
assert  etudiantAdmis == {'lyceen_1': 13, 'lyceen_2': 17,
```

```
    'lyceen_4': 15,
    'lyceen_6': 14,
    'lyceen_7': 16,
    'lyceen_8': 12,
    'lyceen_9': 13,
    'lyceen_10': 15,
    'lyceen_11': 14,
    'lyceen_13': 10,
    'lyceen_14': 12,
    'lyceen_15': 13,
    'lyceen_17': 12,
    'lyceen_18': 15,
    'lyceen_20': 17}
assert
etudiantNonAdmis
== {'lyceen_3': 9,
    'lyceen_5': 8,
    'lyceen_112': 9,
    'lyceen_16': 7,
    'lyceen_19': 9}
```

2.

Vo es expériences ci-dessus : le temps de recherche dans le dictionnaire est
pr du nombre d'entrées dans ce dictionnaires, car en multipliant le nombre de
cc est resté pratiquement identique alors que dans le cas de la recherche dans
ur portionnel à la longueur du tableau.

Le ne structure de données optimisée pour la recherche sur les clés.

Le fonction de hachage.

3. Complément hash (hors programme)

La notion de Hachage est omniprésente en informatique et est au cœur du fonctionnement des dictionnaires. Le hachage est un mécanisme permettant de transformer la clé en un nombre unique permettant l'accès à la donnée, un peu à la manière d'un indice dans un tableau.

3.1. Définition d'une fonction de hachage

Une fonction de hachage est une fonction qui va calculer une empreinte unique à partir de la donnée fournie en entrée. Elle doit respecter les règles suivantes :

- La longueur de l'empreinte (valeur retournée par la fonction de hachage) doit être toujours la même, indépendamment de la donnée fournie en entrée.
- Connaissant l'empreinte, il ne doit pas être possible de reconstituer la donnée d'origine
- des données différentes doivent donner dans la mesure du possible des empreintes différentes.
- des données identiques doivent donner des empreintes identiques.

3.2. Quelques utilisations du hachage

L'utilisation la plus courante est le stockage des mots de passe dans un système informatique un peu sécurisé.
En effet, lorsqu'on crée un compte sur un service en ligne, le mot de passe ne doit pas être stocké en clair, une empreinte est générée afin que si le service est piraté et que les comptes sont dérobés, il ne soit pas possible de reconstituer le mot de passe à partir de l'empreinte. Voici un exemple de fonctionnement d'une fonction de hachage.

Nous utiliserons le hachage **sha256**.

🐍 **Script Python**

```
from hashlib import sha256

h1=sha256(b"Simpson Homer")   # b"Simpson Homer" est un objet de type  bytes
print (h1.hexdigest())
```

```
8d167108dff6b68905c7154c04152a1218511d6b84bc839395e7d653222912f3
```

🐍 **Script Python**

```
h2=sha256(b"Simpson Homer")
print(h2.hexdigest())
```

```
8d167108dff6b68905c7154c04152a1218511d6b84bc839395e7d653222912f3
```

🐍 **Script Python**

```
h3=sha256(b"Simpson  Homer")
print(h3.hexdigest())
```

```
5c9fe022bd1d6dd652e32e0cd871920f7212124cba31532442211599fe634adf
```

Ce n'est pas une valeur au hasard. Un hash donne une même valeur pour un même texte.

Voici un texte qui vous permettra de comprendre le principe des fonctions de hachages : c'est quoi le hachage.

Pour avoir quelques idées sur le principe des tables de hachages, je vous recommande le visionnage de ces deux vidéos:

On constate bien sur cet exemple que :

- un petit changement dans la valeur d'entrée fournit une empreinte totalement différentes
- toutes les empreintes ont la même longueur 64 caractères hexadécimal donc 32 octets et donc \(32\times8=256\) bit.

Video

Vous avez déjà compris que l'algorithme de recherche dans une table de hachage a une complexité \(O(1)\) (le temps de recherche ne dépend pas du nombre d'éléments présents dans la table de hachage), alors que la complexité de l'algorithme de recherche dans un tableau non trié est \(O(n)\).

Comme l'implémentation des dictionnaires s'appuie sur les tables de hachage, on peut dire que l'algorithme de recherche d'un élément dans un dictionnaire a une complexité $O(1)$ alors que l'algorithme de recherche d'un élément dans un tableau non trié a une complexité $O(n)$.