Ce document recense des fonctions du cours et des TP que vous devez savoir écrire rapidement et sans erreur. Lorsque plusieurs versions d'une même fonction sont données, toutes les versions sont à connaître. Pensez à revoir également les fiches de révisions précédentes.

1 Listes

⋆ Indique si toutes les sous-listes de « `L: list[list[int]]` » sont de taille `n`.

```
def test_taille(L, n):
    for l in L:
        if len(l) != n:
            return False
    return True
```

⋆ Renvoie `True` si et seulement si tout `i` $\in [\![0\,;\,n-1]\!]$ apparaît au moins une fois dans `L`.
Complexité en $\mathcal{O}$(`max(len(L),n)`).

```
def tous_presents(L, n):
    B = [False] * n
    for e in L:
        if 0 <= e < n:
            B[e] = True
    for i in range(n):
        if not B[i]:
            return False
    return True
```

2 Boucles imbriquées

⋆ Renvoie $\displaystyle\sum_{i=1}^{n}\sum_{j=1}^{2n}\sqrt{i+j}$:

```
def somme_sqrt(n):
    s = 0
    for i in range(1, n+1):
        for j in range(1, 2*n+1):
            s += (i+j)**0.5
    return s
```

⋆ Indique si tous les entiers présents dans « `L: list[list[int]]` » appartiennent à $[\![1\,;\,n]\!]$.

```
def test_val1(L, n):
    for l in L:
        for e in l:
            if e < 1 or e > n:
                return False
    return True
```

⋆ Renvoie une liste « `L: list[int,int]` » contenant tous les couples `(i, j)` $\in [\![1, n]\!]^2$ tels que `i < j`.

```
def liste_couples(n):
    L = []
    for i in range(1, n+1):
        for j in range(i+1, n+1):
            L.append((i,j))
    return L
```

3 Algorithmes dichotomiques

⋆ Indique si `e` appartient à `L` à l'aide d'une recherche dichotomique

```
# Hypothèse: L est triée par ordre croissant
def recherche_dicho(L, e):
    i = 0
    j = len(L)-1
    while i <= j:
        m = (i+j)//2
        if L[m] == e:
            return True
        elif L[m] < e:
            i = m+1
        else:
            j = m-1
    return False
```

⋆ Calcule x^n à l'aide d'une exponentiation rapide.

```
# Version itérative
# Hypothèse: n >= 0
def expo_rapide(x, n):
    p = 1; y = x; m = n
    while m != 0:
        if m % 2 == 1:
            p = p*y
        y = y*y
        m = m//2
    return p
```

```
# Version récursive
# Hypothèse: n >= 0
def expo_rapide(x, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return expo_rapide(x*x, n//2)
    else:
        return x*expo_rapide(x*x, n//2)
```