TD 10 : AP.

-> sélect°

i	liste	
0	[6, 68, 69, 36, 89, 7, 76, 37, 33, 8]	-> 9 comp
1	[6, 7, 69, 36, 89, 68, 76, 37, 33, 8]	-> 4 comp
2	[6, 7, 8, 36, 89, 69, 76, 37, 33, 69]	-> 7 comp.
3	[6, 7, 8, 33, 89, 68, 76, 36, 69]	-> 5
4	[6, 7, 8, 33, 36, 68, 76, 89, 69]	-> 3
5	[6, 7, 8, 33, 36, 68, 76, 89, 69]	-> 1
6	[6, 7, 8, 33, 36, 68, 69, 89, 76]	-> 2
7	[6, 7, 8, 33, 36, 68, 69, 76, 89]	-> 1 .

-> 32 comp.

-> insert°.

i	liste .	
0	[6, 68, 69, 36, 89, 7, 76, 37, 33, 8]	-> 1 cmp.
1	[6, 68, 69, 36, 89, 7, 76, 37, 33, 8]	-> 2
3	[6, 36, 68, 69, 89, 7, 76, 37, 33, 8]	-> 3
4	[6, 36, 68, 69, 89, 7, 76, 37, 33, 8]	-> 1
5	[6, 7, 36, 68, 69, 89, 76, 37, 33, 8]	-> 5
6	[6, 7, 36, 68, 69, 76, 89, 37, 33, 8]	-> 1
7	[6, 7, 36, 37, 68, 69, 76, 89, 33, 8]	-> 5
8	[6, 7, 33, 36, 37, 68, 69, 76, 89, 8]	-> 7
9	[6, 7, 33, 36, 37, 68, 69, 76, 89] .	-> 9

-> 34 cmp.

5. La liste est triée donc on peut faire une recherche dichotomique.

6. [6, 19, 25, 30, 42, 49, 51, 60 ‖ 66, 67, 72, 76, 85, 92, 99]

[66, 67, 72 ‖ 76, 85, 92, 99]

[76, 85 ‖ 92, 99]

[76 ‖ 85]

[85]

Exo2 :

7, def compare_niveau(p1: Pochémon, p2: Pochémon) -> int:
    Return compare(p1.niveau, p2.niveau)

8. sorted(POCHEDEX, key = cmp_to_key(compare.niveau))

9 def compare_noms (p1: Pochémon, p2: Pochémon) -> int:
    Return compare(len(p1.nom), len(p2.nom))

sorted(POCHEDEX, key = cmp_to_key(compare_noms), Reverse = True)

10 def compare_att_def (p1: Pochémon p2: Pochémon) -> int:
    att = compare(p1.attaque, p2.attaque)
    if att == 0:
        return compare(p1.defense, p2.defense)
    Return att.

. . .

Exo3 :

On pourra utiliser une pile. - est_vide
- empile
- depile.

class TaskManagen:
    def __init__(self):
        self.do_tasks = Pile()
        self.done_tasks = Pile().

    def do(self, task):
        self.do_tasks.empile(task)

    def done(self)
        self.done_tasks.empile(self.do_tasks.depile())

    def undo(self):
        self.do_tasks.empile(self.do_tasks.depile())

Exo 4:

13.
```
def split(L: ApList, ind:int=0)-> tuple[Aplist, ApList]:
    if L.tail.est_vide():
        Return ApList(), ApList()
    else:
        if ind%2==0:
            Return Aplist(L.head, Aplist()) + split(L, ind=ind-1), Aplist()
        else
            Return Aplist(), Aplist(L.head, Aplist()) + split(L, ind=ind-1)
```

14.
```
def fusion(l1, l2):
    if not(l1.est_vide() and l2.est_vide):
        if l1.head() < l2.head():
            Return ApList(l1.head, fusion(l1.tail(), l2))
        else
            Return ApList(l2.head, fusion(l2.tail(), l1))
```

15.