

```

4 def tri_insertion(tab):
5     # parcours de 1 à longueur du tableau
6
7     for i in range(1, len(tab)):
8         1           6
9         k = tab[i]
10        22
11        k = tab[i]
12        22
13        j = i-1
14        0
15
16        while j >= 0 and k < tab[j] : Vraie
17            0       22  98
18
19            tab[j+1] = tab[j]
20            98       98
21
22            j = j-1
23            -1      0
24
25            tab[j+1] = k
26

```

0	1	2	3	4	5
98	22	15	32	2	74
tab[0]	tab[1]	tab[2]	tab[3]	tab[4]	tab[5]

0	1	2	3	4	5
98	98	15	32	2	74
tab[0]	tab[1]	tab[2]	tab[3]	tab[4]	tab[5]

```

4 def tri_insertion(tab):
5     # parcours de 1 à longueur du tableau
6
7     for i in range(1, len(tab)):
8         1           6
9         k = tab[i]
10        22
11        k = tab[i]
12        22
13        j = i-1
14        0
15
16        while j >= 0 and k < tab[j] : Faux
17            -1      22  -1
18
19            tab[j+1] = tab[j] pas exécuté
20
21            j = j-1 pas exécuté
22            on sort de la boucle
23
24            tab[j+1] = k
25            0       22
26

```

Compléter le tableau tab chaque fois qu'il est modifié dans le programme.

On notera que `tab[-1]` ne plante pas en python et n'engendre pas ici d'effet de bord mais ce n'est pas le cas dans d'autres langages qui interdisent les indices négatifs ...

0	1	2	3	4	5
22	98	15	32	2	74
tab[0]	tab[1]	tab[2]	tab[3]	tab[4]	tab[5]

```

4 def tri_insertion(tab):
5     # parcours de 1 à longueur du tableau
6
7     for i in range(1, len(tab)):
8         2
9         k = tab[i]
10        15
11        k = tab[i]
12        15
13        j = i-1
14        1  2
15
16        while j >= 0 and k < tab[j] : Vraie
17            1       1       1
18
19            tab[j+1] = tab[j]
20            2       98
21
22            j = j-1
23            0       1
24
25            tab[j+1] = k
26

```

0	1	2	3	4	5
[	,	,	,	,	]
tab[0]	tab[1]	tab[2]	tab[3]	tab[4]	tab[5]

```

4 def tri_insertion(tab):
5     # parcours de 1 à longueur du tableau
6
7     for i in range(1, len(tab)):
8
9         k = tab[i]
10
11        j = i-1
12
13        while j >= 0 and k < tab[j] :
14
15            tab[j+1] = tab[j]
16
17            j = j-1
18
19            tab[j+1] = k
20
21
22
23
24
25
26

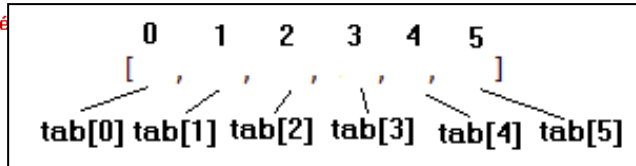
```

0	1	2	3	4	5
[	,	,	,	,	]
tab[0]	tab[1]	tab[2]	tab[3]	tab[4]	tab[5]

```

4 def tri_insertion(tab):
5     # parcours de 1 à longueur du tableau
6
7     for i in range(1, len(tab)):
8
9         k = tab[i]
10
11         j = i-1
12
13         while j >= 0 and k < tab[j] : Faux
14             -1
15
16             tab[j+1] = tab[j] pas exé
17
18             j = j-1 pas exécuté
19
20         tab[j+1] = k

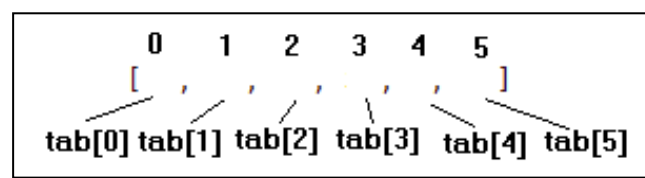
```



```

4 def tri_insertion(tab):
5     # parcours de 1 à longueur du tableau
6
7     for i in range(1, len(tab)):
8
9         k = tab[i]
10
11         j = i-1
12
13         while j >= 0 and k < tab[j] :
14
15             tab[j+1] = tab[j]
16
17             j = j-1
18
19         tab[j+1] = k

```

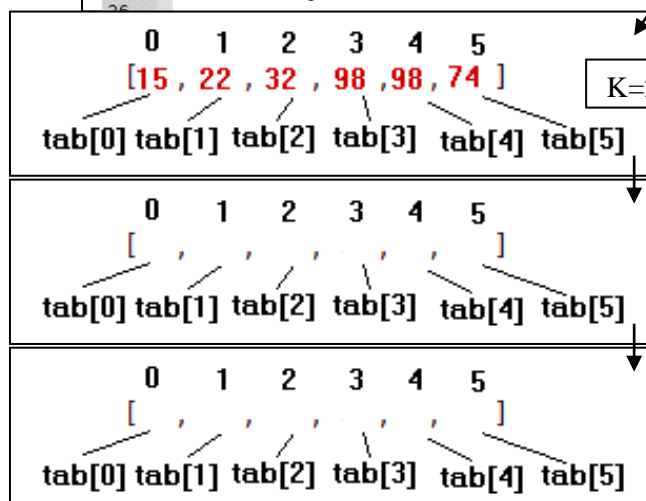
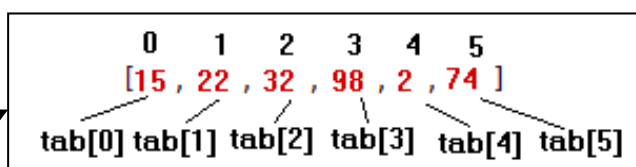


```

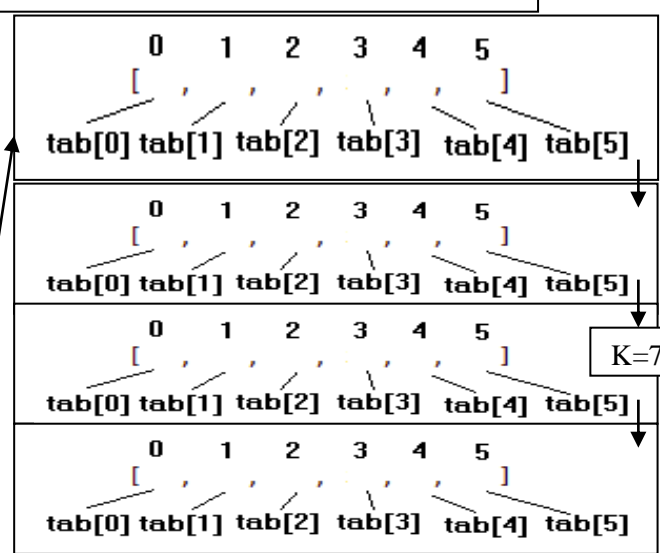
4 def tri_insertion(tab):
5     # parcours de 1 à longueur du tableau
6
7     for i in range(1, len(tab)):
8
9         k = tab[i]
10
11         j = i-1
12
13         while j >= 0 and k < tab[j] :
14
15             tab[j+1] = tab[j]
16
17             j = j-1
18
19         tab[j+1] = k

```

A ce stade vous devriez avoir compris le déroulement de l'algorithme, compléter les modifications du tableau jusqu'à la fin du tri



K=2



K=74