Complexité du tri

Pour évaluer la complexité d'un algorithme de tri, nous allons mesurer

- le nombre de comparaisons
- le nombre d'écritures dans le tableau. (ou d'échanges si c'est plus pertinent).

In [1]:
```
def tri_a_bulles(T):
    nb_comparaisons = nb_echanges = 0

    N = len(T)
    for k in range(N,1,-1):
        for i in range(0,k-1):
            nb_comparaisons += 1
            if T[i] > T[i+1]:
                nb_echanges += 1
                T[i],T[i+1] = T[i+1],T[i]

    return nb_comparaisons, nb_echanges
```

Enfin, il convient d'effectuer les mesures pour diverses tailles de tableau et éventuellement divers contenus.

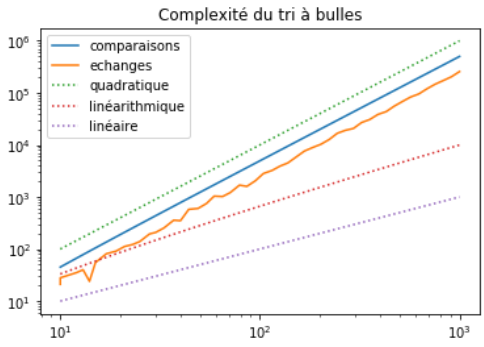
La fonction qui suit teste 50 tailles allant de 10 à 1000 selon une progression géométrique (facteur multiplicatif constant).

Générer le contenu du tableau est sous-traité à la fonction genere_tab passée en paramètre.

In [3]:
```
def evalue_complexite(algorithme, genere_tab, nom):

    C1 = []
    C2 = []
    X = [ int(x) for x in np.logspace(1,3,50) ]

    for n in X:
        T = genere_tab(n)
        comp, ech = algorithme(T)
        C1.append(comp)
        C2.append(ech)

    affiche_complexite(X,C1,C2,nom)
```

Evaluons d'abord la complexité du tri d'un tableau au contenu généré aléatoirement.

In [4]:
```
def tab_aleatoire(n): return np.random.uniform(0,1,n)
evalue_complexite(tri_a_bulles, tab_aleatoire, "tri à bulles")
```



N	Comp.	Ech.
10	45	21
19	171	91
37	666	358
71	2485	1212
138	9453	4502
268	35778	19041
517	133386	67990
1000	499500	256151

Nous voyons que tant le nombre de comparaisons que le nombre d'échanges ont une **complexité quadratique $\Theta(n^2)$** pour trier n éléments.

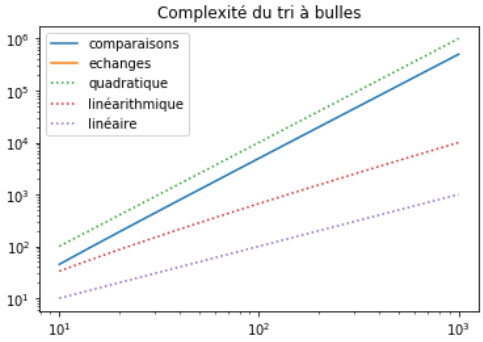
Pour être plus précis

- le nombre de comparaisons est exactement $n(n-1)/2$
- le nombre d'échanges varie selon le contenu mais est de l'ordre de $n^2/4$

Il sera souvent pertinent d'effectuer deux autres tests pour évaluer la complexité de nos algorithmes.

Le premier test consiste à trier un tableau déjà parfaitement trié.

In [6]:
```
def tableau_trie(n): return range(n)
evalue_complexite(tri_a_bulles, tableau_trie, "tri à bulles")
```



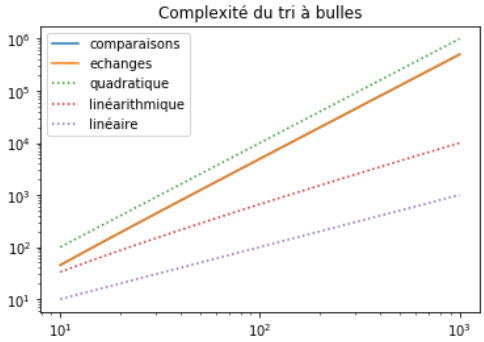
N	Comp.	Ech.
10	45	0
19	171	0
37	666	0
71	2485	0
138	9453	0
268	35778	0
517	133386	0
1000	499500	0

Le nombre de comparaisons ne dépend pas du contenu du tableau.

Le nombre d'échanges est uniformément nul.

Le deuxième test utilise un tableau en ordre décroissant.

In [7]:
```
def tableau_inverse(n): return list(range(n,0,-1))
evalue_complexite(tri_a_bulles, tableau_inverse, "tri à bulles"
)
```



N	Comp.	Ech.
10	45	45
19	171	171
37	666	666
71	2485	2485
138	9453	9453
268	35778	35778
517	133386	133386
1000	499500	499500

Le nombre de comparaisons est inchangé mais le nombre d'échanges est égal au nombre de comparaisons.

Tous les tests de comparaisons renvoient `True`, il y a échange à chaque fois.

Notons que cela ne change rien à la complexité du tri à bulles qui est toujours dominée par celle du nombre de comparaisons.

La complexité du tri à bulles est donc **quadratique, indépendemment du contenu à trier.**

Complexité spatiale

L'analyse précédente ne s'intéresse qu'à l'utilisation d'une ressource: le **temps de calcul**.

Dans certains cas, il est aussi pertinent de s'intéresser à une autre ressource importante: la **quantité de mémoire** utilisée.

Pour le **tri à bulles, cette complexité spatiale est constante,** $\Theta(1)$. En effet la mémoire supplémentaire nécessaire pour trier le tableau inclut

- les deux compteurs de boucle
- un élément temporaire utilisé lors des échanges

la quantité de mémoire est donc indépendante de la taille du tableau à trier.

Ce sera aussi le cas le tri par sélection, le tri par insertion et le tri de Shell que nous verrons par la suite. Nous renviendrons sur ce problème pour les **tri par fusion et tri rapide**.

heig-vd

ASD1 Notebooks on GitHub.io (https://ocuisenaire.github.io/ASD1-notebooks/)