❑ **Exercice 1** : *Programmer un algorithme de tri*
Trois algorithmes de tri on été vus :

— le tri par sélection,

— le tri par insertion,

— le tri à bulles.

Pour *un seul* de ces algorithmes (celui de *de votre choix*) :

1. Indiquer l'algorithme choisi et donner les étapes de son fonctionnement sur le tableau `{2, 8, 1, 5}`

On propose la correction pour les trois algorithmes (un seul était demandé). On note `n` la taille du tableau.

— **tri par sélection**
Le principe est de chercher pour `i=0 ... i=n-1` le minimum du sous tableau `tab[i] ... tab[n-1]` et de l'échanger avec l'élément d'indice `i`.
Sur l'exemple on obtient (en soulignant le minimum de chaque sous tableau)

| Etapes | Etat du tableau |
|---|---|
| Initialisation | {2, 8, 1, 5} |
| i=0 1er minimum | {1, 8, 2, 5} |
| i=1 2eme minimum | {1, 2, 8, 5} |
| i=2 3eme minimum | {1, 2, 5, 8} |

— **tri par insertion**
Le principe est d'insérer pour `i=1 ... i=n-1`, tab[i] dans le sous tableau `tab[0] ... tab[i-1]` en *considérant ce sous tableau déjà trié*. Sur l'exemple on obtient (en soulignant la partie déjà triée)

| Etapes | Etat du tableau |
|---|---|
| Initialisation | {2, 8, 1, 5} |
| Insertion de 8 | {2, 8, 1, 5} |
| Insertion de 1 | {1, 2, 8, 5} |
| Insertion de 5 | {1, 2, 5, 8} |

— **tri à bulles**
Le principe est pour `i=n-1 ... 0` de parcourir le sous tableau `tab[0], ... tab[i]` en échangeant un élément avec son voisin de droite s'il lui est supérieur. Sur l'exemple on obtient :

| Etapes | Etat du tableau |
|---|---|
| Initialisation | {2, 8, 1, 5} |
| 1er parcours | {2, 1, 5, 8} |
| 2eme parcours | {1, 2, 5, 8} |
| 3eme parcours | {1, 2, 5, 8} |

2. Ecrire son implémentation en C, en supposant *déjà écrite* une fonction de prototype :
`echange(int tab[], int i, int j)` qui prend en argument un tableau `tab`, sa taille `size` et deux indices `i` et `j` et échange les éléments situés aux indices `i` et `j` de `tab` en supposant vérifiées les préconditions sur `i` et`j`.

```c
// Renvoie l'indice du minimum des éléments de tab depuis l'indice ind
int min_depuis(int tab[], int taille, int ind)
{
    assert(ind < taille && ind >= 0);
    int vmin = tab[ind];
    int imin = ind;
    for (int i = ind; i < taille; i++)
    {
        if (tab[i] < vmin)
        {
            imin = i;
            vmin = tab[i];
        }
    }
    return imin;
}

// Tri en place un tableau par sélection
void tri_selection(int tab[], int size)
{
    int im;
    for (int i = 0; i < size; i++)
    {
        im = min_depuis(tab, size, i);
        echange(tab, i, im, size);
    }
}
```

```c
// Insère tab[i] dans le sous tableau tab[0]...tab[i-1]
// en supposant que ce sous tableau est déjà trié
void insere(int tab[],int i,int size)
{
    while (i-1>=0 && tab[i]<tab[i-1])
    {
        echange(tab,i,i-1,size);
    }
}

// Tri par insertion le tableau tab
void tri_insertion(int tab[], int size)
{
    for (int i=1;i<size;i++)
    {
        insere(tab,i,size);
    }
}
```

```c
// Parcours le tableau tab jusqu'à l'indice limite
// en échangeant un élément avec son voisin s'il lui est supérieur
void parcours(int tab[], int limite, int taille)
{
    assert (limite<taille);
    for (int i=0;i<limite;i++)
    {
        if (tab[i]>tab[i+1])
        {
            echange(tab,i,i+1,taille);
        }
    }
}

// Tri à bulles du tableau tab
void tri_bulles(int tab[], int taille)
{
    for (int lim=taille-1;lim>=1;lim--)
    {
        parcours(tab,lim,taille);
    }
}
```

3. Créer un tableau `test` de 5000 entiers et initialiser ce tableau de façon à ce que `tab[i] = i*i - 564*i + 77760` pour `i=0 ... 4999`. Trier ce tableau à l'aide de l'implémentation de tri précédente et donner la valeur de `tab[2024]`.

```c
int main()
{
    int test[5000];
    for (int i = 0; i < 5000; i++)
    {
        test[i] = i * i - 564 * i + 77760;
    }
    tri_selection(test, 5000);
    printf("Réponse = %d\n", test[2024]);
}
```

On obtient la valeur 3032800