1. Сортування обміном

```cpp
/* a - масив для сортування , n - розмір масиву
ідея сортування : кожен раз знаходимо найменший елемент із ще не відсортованих
і ставимо його на відповідну позицію в масиві
Складність : O(n^2)
*/
void selectionSort(int a[], int n)
{
    for(int i = 0; i < n - 1; i++) {
        // знаходимо мінімальний елемент із ще не відсортованих
        int m = i;
        for(int j = i + 1; j < n; j++) {
            if(a[j] < a[m]) {
                m = j;
            }
        }
        /// ставимо мінімальний елемент на потрібне місце
        swap(a[i], a[m]);
    }
}
```

2. Сортування бульбашкою

```cpp
/* a - масив для сортування , n - розмір масиву
Ідея : кожного разу за допомогою порівнянь сусідніх елементів
підтягуємо на місце найбільший елемент, який не на своєму місці
Оскільки в найгіршому випадку знадобиться n-1 підятгування,
 то потрібно повторити цю операцію n-1 раз
Складність : O(n^2)
*/
void stupidBubbleSort(int a[], int n)
{
    for(int i = 0; i < n - 1; i++) {
        // на кожному кроці підтягуємо елемент послідовними порівняннями
        for(int j = 1; j < n - i; j++) {
            if(a[j - 1] > a[j]) {
                swap(a[j - 1], a[j]);
            }
        }
    }
}

/*
Ідея оптимізації в тому, щоб завершувати роботу функції одразу,
як ми відсортували масив. Критерій відсортованості :
якщо жоден елемент не потрібно було обмінювати місцями,
то всі елементи стоять на своїх місцях, і масив відсортований.
*/

void smartBubbleSort(int a[], int n)
{
    for(int i = 0; i < n - 1; i++) {
        int swapped = 0;
        for(int j = 1; j < n - i; j++) {
            if(a[j - 1] > a[j]) {
                swap(a[j - 1], a[j]);
                swapped++;
            }
        }
        if(!swapped) break;
    }
}
```

3. Сортування включенням

```cpp
/*a - масив для сортування, n - розмір масиву
Ідея : приймаємо по черзі елементи масиву і
вставлємо їх на потрібні позиції в масиві
Складність : O(n^2) в гіршому випадку або O(n + d), де d - кількість інверсій
*/

void insertionSort(int a[], int n)
{
    for(int i = 0; i < n; i++) {
        int j = i - 1, x = a[i]; // витягуємо елемент з масива і запам'ятовуємо значення в x
        while(j >= 0 && a[j] > x) { // поки не закінчились позиції і цей елемент більший за той, який вставляємо
            a[j + 1] = a[j];//зміщуємо цей елемент вперед на одну позицію
            j--;
        }
        a[j + 1] = x;//ставимо елемет після першого елемента, не більшого за наший;
    }
}
```

4. Сортування злиттям

```cpp
const int MAXN = 1e5 + 10;
int b[MAXN];// Допоміжний масив, потрібний для об'єднання двох відсортованих відрізків
/*
Процедура об'єднання двох відсортованих відрізків в один
a - масив, l, r - ліва і права межа відрізка відповідно

Ідея : давайте формувати наш список так : розглядаємо
два чергові елементи відрізків(спочатку це два перші елементи),
очевидно, один із них стоятиме на першій позиції,
тож обираємо найменший з них, ставимо його в початок нашої послідовності
і вважаємо черговим елементом наступний
*/
void mergeRanges(int a[], int l, int r) {
    int sizeOfTheRange = r - l + 1;//розмір відрізка-результату
    int m = (l + r) / 2;//середина відрізка
    for(int i = l, j = m + 1, k = 0; k < sizeOfTheRange; k++) {
    /* i - черговий елемент першої половини,
    j- другої,
    k - номер елемента, який ми додаємо в послідовність-відповідь*/
        if(i > m || (j <= r && a[i] > a[j])) {
            /* виконується якщо перша половина вичерпалась,
            або теперішній мінімум занходиться в іншій половині,
            яка себе не вичерпала також
            */
            b[k] = a[j];
            j++;
        }
        else {
            b[k] = a[i];
            i++;
        }
    }
    for(int i = 0; i < sizeOfTheRange; i++) {
        a[l + i] = b[i];
    }
}
/*
Процедура сортування злиттям
a - масив, l і r - межі відрізка, що сортується

Ідея : якщо масив довжини один, то він автоматично відсортований,
в іншому випадку ми можемо розділити масив на 2 половини і
рекурсивно запустити сортування,
після цього об'єднати дві половини масива
*/
void mergeSort(int a[], int l, int r) {
    if(l == r) return;// в цьому випадку відрізок відсортований
    int m = (l + r) / 2;
    //визначили середину і запустили рекурсивне сортування для обох половин масиву
    mergeSort(a, l, m);
    mergeSort(a, m + 1, r);
    //об'єднали два відрізки в один
    mergeRanges(a, l, r);
}
```