**Пузырьковая сортировка**

int arr[] = { … };

for (int i = 0; i < size - 1; i++)

{

bool swapped = false;

for (int j = 0; j < size - i - 1; j++)

{

if (arr[j] > arr[j + 1])

{

T temp = arr[j + 1];

arr[j + 1] = arr[j];

arr[j] = temp;

swapped = true;

}

}

if (!swapped) { break; }

}

**Сортировка вставками**

int arr[] = { … };

for (int i = 1; i < size; i++)

{

T temp = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > temp)

{

arr[j + 1] = arr[j];

j--;

}

arr[++j] = temp;

}

# Сортировка выбором

# for (int i = 0; i < size - 1; i++) {

int temp = i;

for (int j = i + 1; j < size; j++)

{

if (foo(arr[j],arr[temp]))

{

temp = j;

}

}

if (temp != i)

{

std::swap(arr[i], arr[temp]);

}

}

**Сортировка слиянием**

const int nmax **= …** - максимальное значение массива

void fill(int arr[], int begin, int end)

{

int i = begin,

mid = begin + (end - begin) / 2,

t = 0,

j = mid + 1,

d[nmax];

while (i <= mid && j <= end)

{

if (arr[i] <= arr[j])

{

d[t] = arr[i];

t++;

i++;

}

else

{

d[t] = arr[j];

j++;

t++;

}

}

while (i <= mid)

{

d[t] = arr[i];

i++; t++;

}

while (j <= end)

{

d[t] = arr[j];

j++; t++;

}

for (int i = 0; i < t; i++)

{

arr[begin + i] = d[i];

cout << arr[begin + i] << ", ";

}

cout << endl;

}

void sort(int\* arr, int left, int right)

{

if (right > left)

{

if (right - left == 1)

{

if (arr[right] < arr[left])

{

int temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

}

}

else

{

sort(arr, left, left + (right - left) / 2);

sort(arr, left + (right - left) / 2 + 1, right);

fill(arr, left, right);

}

}

}

**Bucket sort**

srand(time(0));

int N = 1000, bucket = 10, j;

double \*arr = new double[N];

double max = NULL, min = NULL, devider;

vector<double> \*B = new vector<double>[bucket];

//fill

for (int i = 0; i < N; i++)

{

arr[i] = rand();

if (arr[i] > max || max == NULL) {

max = arr[i];

}

if (arr[i] < min || min == NULL) {

min = arr[i];

}

}

//find devider

devider = ceil((max + 1) / bucket);

for (int i = 0; i < N; i++)

{

j = floor(arr[i] / devider);

B[j].push\_back(arr[i]);

}

//sorting

for (int i = 0; i < bucket; i++)

{

for (vector<double>::iterator it2 = B[i].begin(); it2 != B[i].end(); ++it2)

{

for (vector<double>::iterator it = B[i].begin(); it + 1 != B[i].end(); ++it)

{

if (\*it > \*(it + 1)) {

double temp = \*it;

\*it = \*(it + 1);

\*(it + 1) = temp;

}

}

}

}

//Filling main array

int m = 0;

for (int i = 0; i < N; i++)

{

for (vector<double>::iterator it = B[i].begin(); it != B[i].end() && m < N; ++it) {

if (m < N) {

arr[m++] = \*it;

}

}

}

//show arr

cout << "array: \n";

for (int i = 0; i < N; i++)

{

cout << arr[i] << ", ";

}

cout << endl;

**Piramid sort**

void heap(double \*arr, int i, int n) {

int max = i;

while (true)

{

int child = 2 \* i + 1; // left child

if (child < n && arr[child] > arr[max])

{

max = child;

}

child++; //right one

if (child < n && arr[child] > arr[max])

{

max = child;

}

if (max == i) {

break;

}else{

swap(arr[max], arr[i]);

i = max;

}

}

}

void heapSort() {

int Size = 10;

double\* arr = createArray(Size);

//////////////////////////////////////////////

for (int i = Size/2; i >= 0; i--)

{

heap(arr, i, Size);

}

for (int i = Size - 1; i >= 1; i--)

{

swap(arr[0], arr[i]);

heap(arr, 0, i);

}

cout << "\n\narray: ";

for (int i = 0; i < Size; i++)

{

cout << arr[i] << ", ";

}

cout << endl;

}