Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы» (ИТАС)

Лабораторная работа

На тему

«Сложные сортировки»

Выполнил

Студент группы ИВТ-23-1б

Адаев Даниил Дмитриевич

Проверил

Доцент кафедры ИТАС

Яруллин Д. В.

г. Пермь, 2024

Задание

Выполнить сортировку массива 6 методами сортировки:

1. Блочная сортировка
2. Сортировка подсчетом
3. Быстрая сортировка
4. Сортировка слиянием
5. Сортировка Шелла
6. Сортировка Хоара
7. Сортировка естественным слиянием
8. Сортировка многофазным слиянием

Создать меню, в котором пользователь выбирает необходимый метод сортировки.

Блочная сортировка

Алгоритм блочной сортировки:

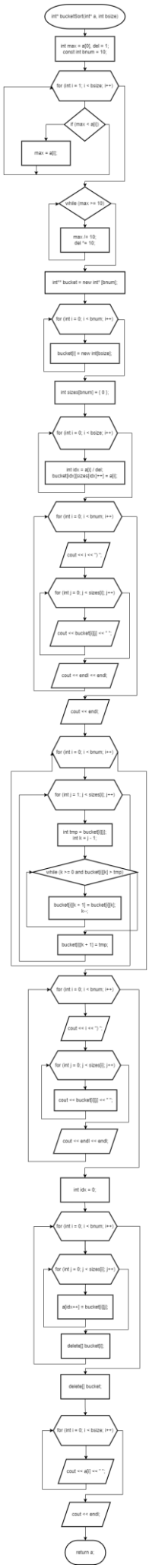
1. Создадим массив, где каждый слот будет ведром.
2. Распределим элементы по ведрам на основании их значений.
   - берем элементы по очереди
   - умножаем элемент на размер массива с ведрами
   - получаем целое число, которое и даст индекс ведер
   - элемент помещается в ведро с этим индексом
   - повторяем для всех остальных элементов
3. Сортируем каждое отдельно взятое ведро, берем любой алгоритм простой сортировки
4. Собираем по очереди из каждого ведра элементы и помещаем их в исходный массив
   - идем по ведрам по очереди
   - идем по элементам ведер по очереди
   - помещаем каждый элемент в том же порядке в исходный массив
   - как только все элементы помещены в исходный массив, ведро считается пустым
   - переходим к следующему ведру
5. Массив отсортирован

Код на C++

```
int* bucketSort(int* a, int bsize)
{
	int max = a[0], del = 1;
	const int bnum = 10;

	for (int i = 1; i < bsize; i++)
	{
		if (max < a[i])
		{
			max = a[i];
		}
	}

	while (max >= 10)
	{
		max /= 10;
		del *= 10;
	}

	int** bucket = new int* [bnum];

	for (int i = 0; i < bnum; i++)
	{
		bucket[i] = new int[bsize];
	}

	int sizes[bnum] = { 0 };

	for (int i = 0; i < bsize; i++)
	{
		int idx = a[i] / del;
		bucket[idx][sizes[idx]++] = a[i];
	}

	for (int i = 0; i < bnum; i++)
	{
		cout << i << ") ";

		for (int j = 0; j < sizes[i]; j++)
		{
			cout << bucket[i][j] << " ";
		}

		cout << endl << endl;
	}

	cout << endl;

	for (int i = 0; i < bnum; i++)
	{
		for (int j = 1; j < sizes[i]; j++)
		{
			int tmp = bucket[i][j];
			int k = j - 1;

			while (k >= 0 and bucket[i][k] > tmp)
			{
				bucket[i][k + 1] = bucket[i][k];
				k--;
			}

			bucket[i][k + 1] = tmp;
		}
	}
```

```
	for (int i = 0; i < bnum; i++)
	{
		cout << i << ") ";

		for (int j = 0; j < sizes[i]; j++)
		{
			cout << bucket[i][j] << " ";
		}

		cout << endl << endl;
	}

	int idx = 0;

	for (int i = 0; i < bnum; i++)
	{
		for (int j = 0; j < sizes[i]; j++)
		{
			a[idx++] = bucket[i][j];
		}

		delete[] bucket[i];
	}

	delete[] bucket;

	for (int i = 0; i < bsize; i++)
	{
		cout << a[i] << " ";
	}

	cout << endl;

	return a;
}
```

Блок-схема


int* bucketSort(int* a, int bsize)
int max = a[0], del = 1;
const int bnum = 10;
for (int i = 1; i < bsize; i++)
if (max < a[i])
max = a[i];
while (max >= 10)
max /= 10;
del *= 10;
int** bucket = new int* [bnum];
for (int i = 0; i < bnum; i++)
bucket[i] = new int[bsize];
int sizes[bnum] = { 0 };
for (int i = 0; i < bsize; i++)
int idx = a[i] / del;
bucket[idx][sizes[idx]++] = a[i];
for (int i = 0; i < bnum; i++)
cout << i << ": ";
for (int j = 0; j < sizes[i]; j++)
cout << bucket[i][j] << " ";
cout << endl << endl;
cout << endl;
for (int i = 0; i < bnum; i++)
for (int j = 1; j < sizes[i]; j++)
int tmp = bucket[i][j];
int k = j - 1;
while (k >= 0 and bucket[i][k] > tmp)
bucket[i][k + 1] = bucket[i][k];
k--;
bucket[i][k + 1] = tmp;
for (int i = 0; i < bnum; i++)
cout << i << ": ";
for (int j = 0; j < sizes[i]; j++)
cout << bucket[i][j] << " ";
cout << endl << endl;
int idx = 0;
for (int i = 0; i < bnum; i++)
for (int j = 0; j < sizes[i]; j++)
a[idx++] = bucket[i][j];
delete[] bucket[i];
delete[] bucket;
for (int i = 0; i < bsize; i++)
cout << a[i] << " ";
cout << endl;
return a;

Сортировка подсчетом

Алгоритм сортировки подсчетом:

1. Находим максимальный элемент массива
2. Создаем массив для подсчета длиной max+1, заполняем его 0
3. В массиве подсчета храним число вхождений каждого элемента (на соответствующих индексах)
4. Вычисляем префиксную сумму для элементов в массиве подсчета (это сумма элементов, стоящих до заданного и самого этого элемента)
5. Начинаем идти с конца исходного массива и обновляем выходной массив при помощи алгоритма:

outArr[countArr[inpArr[i]]-1] = inpArr[i];

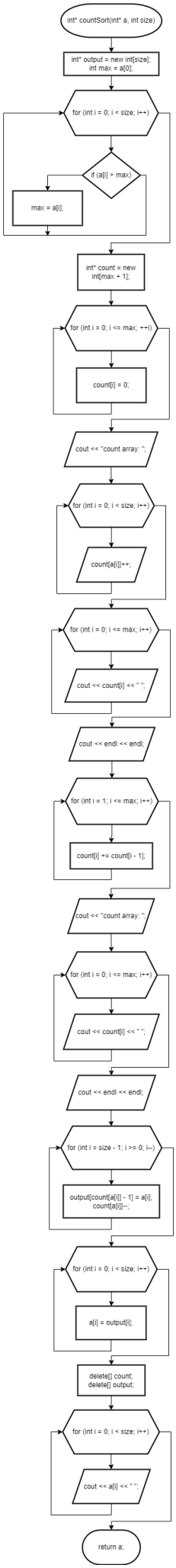
countArr[inpArr[i]] = countArr[inpArr[i]]--;

Код на C++

```
int* countSort(int* a, int size)
{
	int* output = new int[size];
	int max = a[0];

	for (int i = 0; i < size; i++)
	{
		if (a[i] > max)
		{
			max = a[i];
		}
	}

	int* count = new int[max + 1];

	for (int i = 0; i <= max; ++i)
	{
		count[i] = 0;
	}

	cout << "count array: ";

	for (int i = 0; i < size; i++)
	{
		count[a[i]]++;
	}

	for (int i = 0; i <= max; i++)
	{
		cout << count[i] << " ";
	}

	cout << endl << endl;

	for (int i = 1; i <= max; i++)
	{
```

```
        count[i] += count[i - 1];
    }

    cout << "count array: ";

    for (int i = 0; i <= max; i++)
    {
        cout << count[i] << " ";
    }

    cout << endl << endl;

    for (int i = size - 1; i >= 0; i--)
    {
        output[count[a[i]] - 1] = a[i];
        count[a[i]]--;
    }

    for (int i = 0; i < size; i++)
    {
        a[i] = output[i];
    }

    delete[] count;
    delete[] output;

    for (int i = 0; i < size; i++)
    {
        cout << a[i] << " ";
    }

    return a;
}
```

Блок-схема


int* countSort(int* a, int size)
int* output = new int[size];
int max = a[0];
for (int i = 0; i < size; i++)
if (a[i] > max)
max = a[i];
int* count = new int[max + 1];
for (int i = 0; i <= max; ++i)
count[i] = 0;
cout << "count array: ";
for (int i = 0; i < size; i++)
count[a[i]]++;
for (int i = 0; i <= max; i++)
cout << count[i] << " ";
cout << endl << endl;
for (int i = 1; i <= max; i++)
count[i] += count[i - 1];
cout << "count array: ";
for (int i = 0; i <= max; i++)
cout << count[i] << " ";
cout << endl << endl;
for (int i = size - 1; i >= 0; i--)
output[count[a[i]] - 1] = a[i];
count[a[i]]--;
for (int i = 0; i < size; i++)
a[i] = output[i];
delete[] count;
delete[] output;
for (int i = 0; i < size; i++)
cout << a[i] << " ";
return a;

Сортировка слиянием

Алгоритм сортировки слиянием:

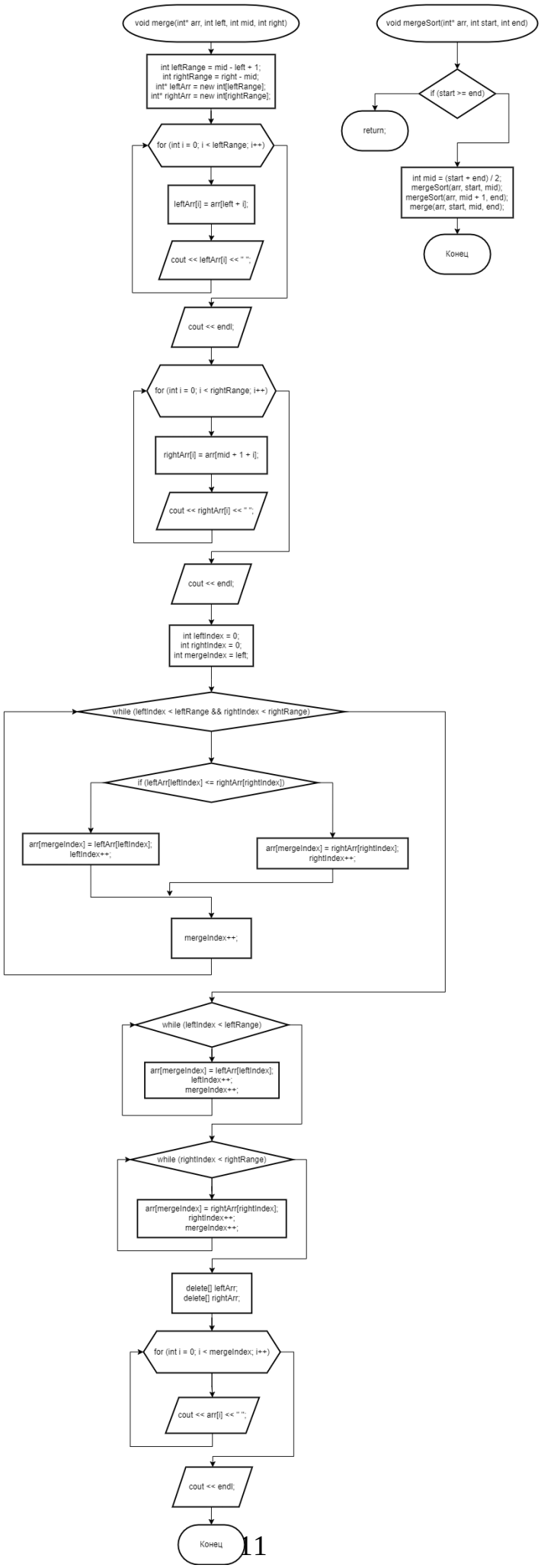
1. Принимаем на вход массив
2. Определяем его левую (l) и правую (r) границы, а также середину массива (округляя в большую сторону, если нужно)
3. Выполняем рекурсивное слияние.

Код на C++

```
void merge(int* arr, int left, int mid, int right)
{
	int leftRange = mid - left + 1;
	int rightRange = right - mid;

	int* leftArr = new int[leftRange];
	int* rightArr = new int[rightRange];

	for (int i = 0; i < leftRange; i++)
	{
		leftArr[i] = arr[left + i];

		cout << leftArr[i] << " ";
	}

	cout << endl;

	for (int i = 0; i < rightRange; i++)
	{
		rightArr[i] = arr[mid + 1 + i];

		cout << rightArr[i] << " ";
	}

	cout << endl;

	int leftIndex = 0;
	int rightIndex = 0;
	int mergeIndex = left;

	while (leftIndex < leftRange && rightIndex < rightRange)
	{
		if (leftArr[leftIndex] <= rightArr[rightIndex])
		{
			arr[mergeIndex] = leftArr[leftIndex];
			leftIndex++;
		}
		else
		{
			arr[mergeIndex] = rightArr[rightIndex];
			rightIndex++;
		}
		mergeIndex++;
	}

	while (leftIndex < leftRange)
	{
		arr[mergeIndex] = leftArr[leftIndex];
		leftIndex++;
		mergeIndex++;
```

```
	}

	while (rightIndex < rightRange)
	{
		arr[mergeIndex] = rightArr[rightIndex];
		rightIndex++;
		mergeIndex++;
	}

	delete[] leftArr;
	delete[] rightArr;

	for (int i = 0; i < mergeIndex; i++)
	{
		cout << arr[i] << " ";
	}

	cout << endl;
}


void mergeSort(int* arr, int start, int end)
{
	if (start >= end)
	{
		return;
	}
	int mid = (start + end) / 2;
	mergeSort(arr, start, mid);
	mergeSort(arr, mid + 1, end);
	merge(arr, start, mid, end);
}
```

Блок-схема


void merge(int* arr, int left, int mid, int right)
int leftRange = mid - left + 1;
int rightRange = right - mid;
int* leftArr = new int[leftRange];
int* rightArr = new int[rightRange];
for (int i = 0; i < leftRange; i++)
leftArr[i] = arr[left + i];
cout << leftArr[i] << " ";
cout << endl;
for (int i = 0; i < rightRange; i++)
rightArr[i] = arr[mid + 1 + i];
cout << rightArr[i] << " ";
cout << endl;
int leftIndex = 0;
int rightIndex = 0;
int mergeIndex = left;
while (leftIndex < leftRange && rightIndex < rightRange)
if (leftArr[leftIndex] <= rightArr[rightIndex])
arr[mergeIndex] = leftArr[leftIndex];
leftIndex++;
arr[mergeIndex] = rightArr[rightIndex];
rightIndex++;
mergeIndex++;
while (leftIndex < leftRange)
arr[mergeIndex] = leftArr[leftIndex];
leftIndex++;
mergeIndex++;
while (rightIndex < rightRange)
arr[mergeIndex] = rightArr[rightIndex];
rightIndex++;
mergeIndex++;
delete[] leftArr;
delete[] rightArr;
for (int i = 0; i < mergeIndex; i++)
cout << arr[i] << " ";
cout << endl;
Конец
void mergeSort(int* arr, int start, int end)
if (start >= end)
return;
int mid = (start + end) / 2;
mergeSort(arr, start, mid);
mergeSort(arr, mid + 1, end);
merge(arr, start, mid, end);
Конец

Сортировка Шелла

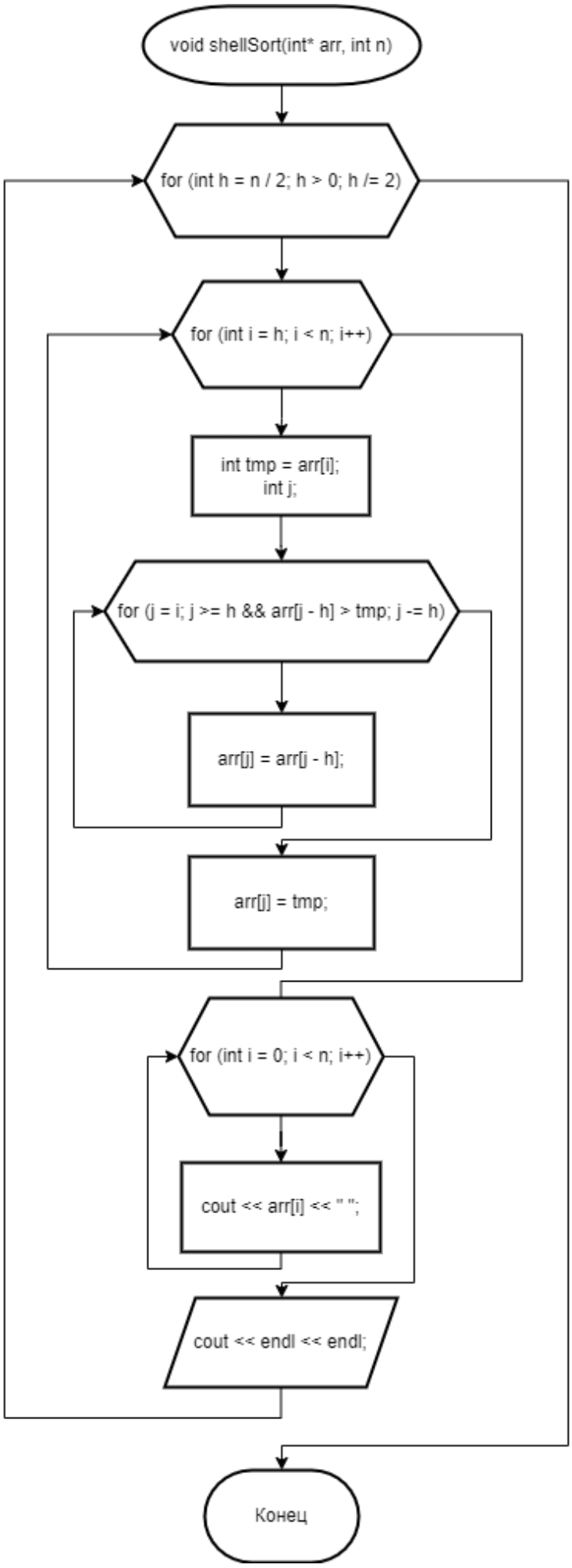
Алгоритм сортировки Шелла:

1. Выбрать размер окна
2. Разделить массив на несколько меньших частей, каждая должна нацело делиться на размер окна
3. Отсортировать каждую из частей при помощи простой сортировки вставкой
4. Продолжать с шага 1 до тех пор, пока не отсортируется весь массив

Код на C++

```
void shellSort(int* arr, int n)
{
	for (int h = n / 2; h > 0; h /= 2)
	{
		for (int i = h; i < n; i++)
		{
			int tmp = arr[i];
			int j;

			for (j = i; j >= h && arr[j - h] > tmp; j -= h)
			{
				arr[j] = arr[j - h];
			}

			arr[j] = tmp;
		}

		for (int i = 0; i < n; i++)
		{
			cout << arr[i] << " ";
		}

		cout << endl << endl;
	}
}
```

Блок-схема


void shellSort(int* arr, int n)
for (int h = n / 2; h > 0; h /= 2)
for (int i = h; i < n; i++)
int tmp = arr[i];
int j;
for (j = i; j >= h && arr[j - h] > tmp; j -= h)
arr[j] = arr[j - h];
arr[j] = tmp;
for (int i = 0; i < n; i++)
cout << arr[i] << " ";
cout << endl << endl;
Конец

Быстрая сортировка Ломуто

Общий алгоритм быстрых сортировок:

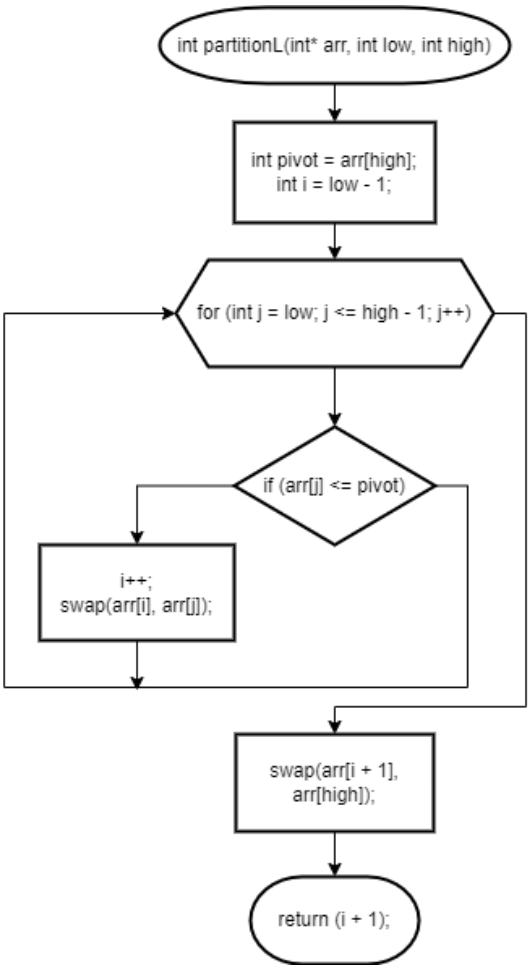
1. Если диапазон менее 2-х элементов, сразу же прекращаем действия, т.к. сортировать нечего.
2. Выбираем точку (pivot, значение), которая встречается в диапазоне. Конкретный принцип выбора такого значения зависит от выбранного извода и может включать элементы случайности (рандомизации).
3. Разбиваем диапазон на части: меняем порядок элементов, одновременно определяя точку деления таким образом, чтобы элементы со значением меньшим, чем pivot, шли до точки деления, тогда как элементы с большим значением — после точки деления. Элементы, равные значению pivot, могут записываться в любую часть (зависит от извода). Т.к. минимум одно значение равно pivot, большинство изводов стремятся точку деления приравнять к pivot.
4. Рекурсивно применяем быструю сортировку к каждой из полученных частей (до и после точки деления), пока алгоритм это позволяет (т.е. пока он не остановится на шаге 1)

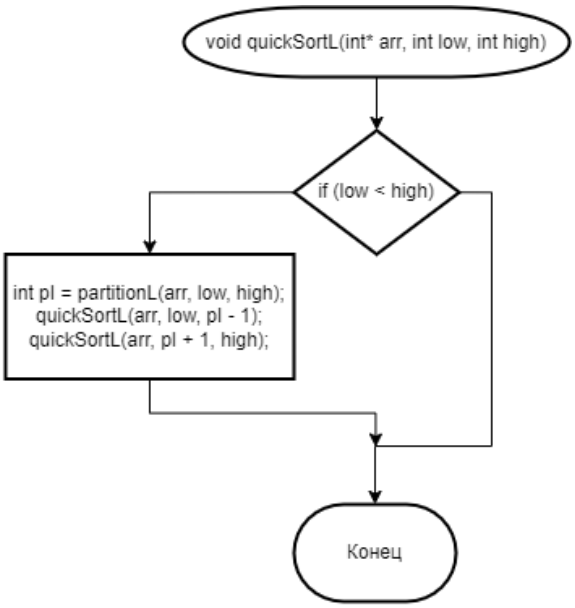
Код на C++

```
int partitionL(int* arr, int low, int high)
{
	int pivot = arr[high];
	int i = low - 1;

	for (int j = low; j <= high - 1; j++)
	{
		if (arr[j] <= pivot)
		{
			i++;
			swap(arr[i], arr[j]);
		}
	}

	swap(arr[i + 1], arr[high]);

	return (i + 1);
}

void quickSortL(int* arr, int low, int high)
{
	if (low < high)
	{
```

```
        int pI = partitionL(arr, low, high);
        quickSortL(arr, low, pI - 1);
        quickSortL(arr, pI + 1, high);
    }
                                        }
```

Блок-схема





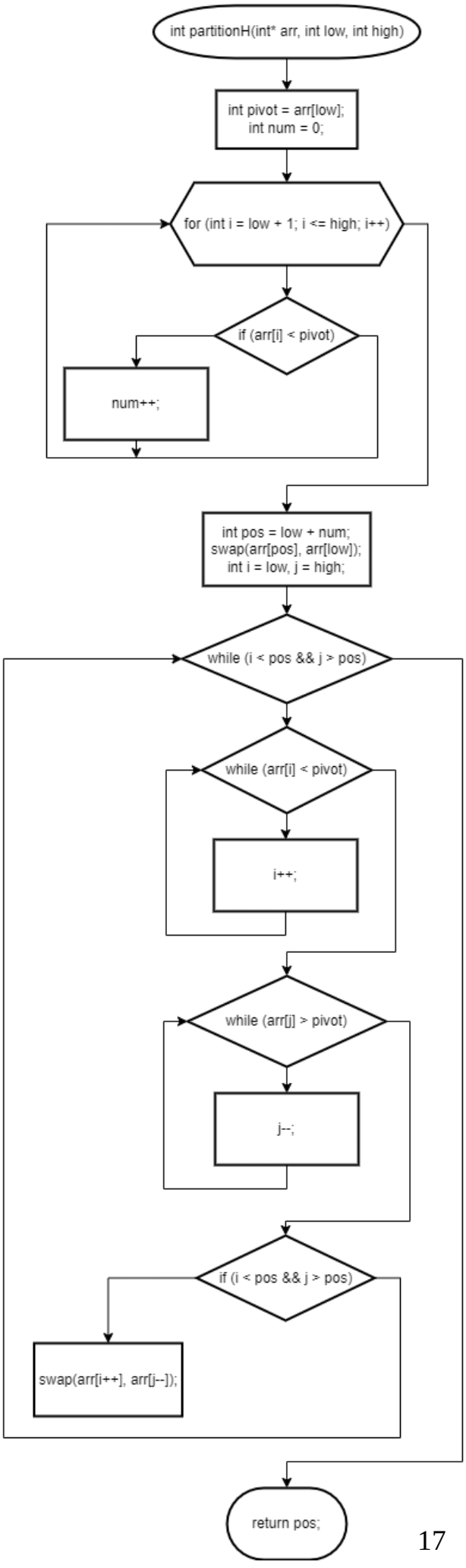
Быстрая сортировка Хоара

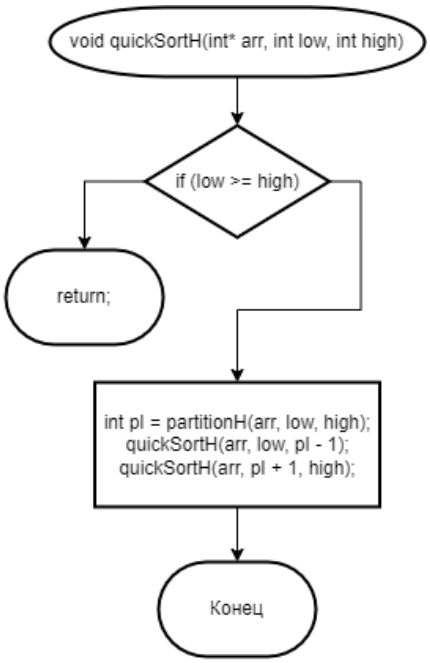
Код на C++

```
int partitionH(int* arr, int low, int high)
{
    int pivot = arr[low];
    int num = 0;

    for (int i = low + 1; i <= high; i++)
    {
        if (arr[i] < pivot)
        {
            num++;
        }
    }

    int pos = low + num;

    swap(arr[pos], arr[low]);
```

```
	int i = low, j = high;
	while (i < pos && j > pos)
	{
		while (arr[i] < pivot)
		{
			i++;
		}

		while (arr[j] > pivot)
		{
			j--;
		}

		if (i < pos && j > pos)
		{
			swap(arr[i++], arr[j--]);
		}
	}

	return pos;
}

void quickSortH(int* arr, int low, int high)
{
	if (low >= high)
	{
		return;
	}

	int pI = partitionH(arr, low, high);
	quickSortH(arr, low, pI - 1);
	quickSortH(arr, pI + 1, high);
}
```

Блок-схема


int partitionH(int* arr, int low, int high)
int pivot = arr[low];
int num = 0;
for (int i = low + 1; i <= high; i++)
if (arr[i] < pivot)
num++;
int pos = low + num;
swap(arr[pos], arr[low]);
int i = low, j = high;
while (i < pos && j > pos)
while (arr[i] < pivot)
i++;
while (arr[j] > pivot)
j--;
if (i < pos && j > pos)
swap(arr[i++], arr[j--]);
return pos;


void quickSortH(int* arr, int low, int high)
if (low >= high)
return;
int pl = partitionH(arr, low, high);
quickSortH(arr, low, pl - 1);
quickSortH(arr, pl + 1, high);
Конец

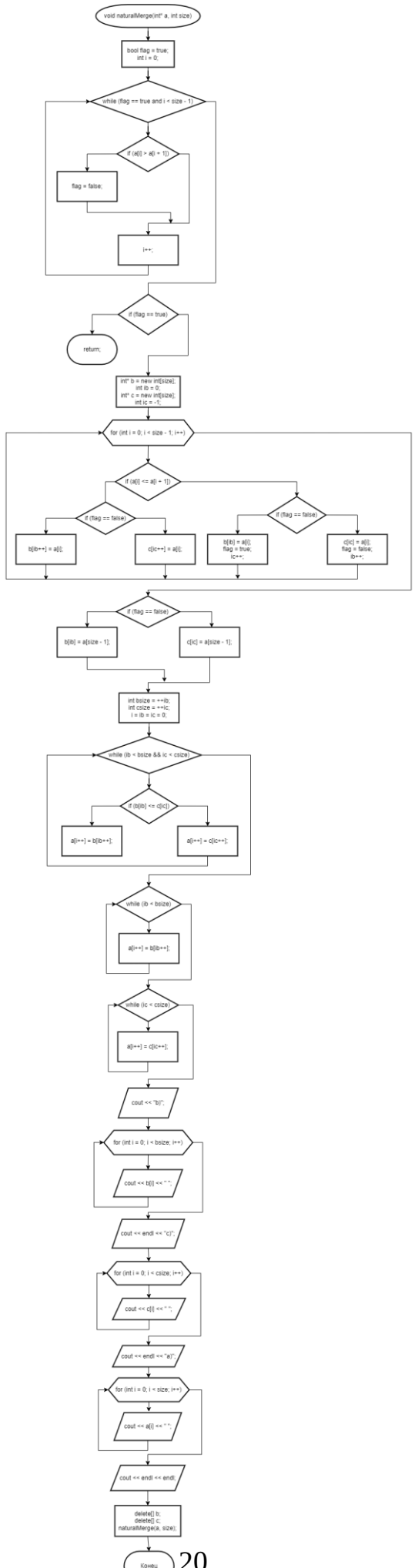
Сортировка естественным слиянием

Код на C++

```
void naturalMerge(int* a, int size)
{
	bool flag = true;
	int i = 0;

	while (flag == true and i < size - 1)
	{
		if (a[i] > a[i + 1])
		{
			flag = false;
		}

		i++;
	}

	if (flag == true)
	{
		return;
	}

	int* b = new int[size];
	int ib = 0;
	int* c = new int[size];
	int ic = -1;

	for (int i = 0; i < size - 1; i++)
	{
		if (a[i] <= a[i + 1])
		{
			if (flag == false)
			{
				b[ib++] = a[i];
			}
			else
			{
				c[ic++] = a[i];
			}
		}
		else
		{
			if (flag == false)
			{
				b[ib] = a[i];
				flag = true;
				ic++;
			}
			else
			{
				c[ic] = a[i];
				flag = false;
				ib++;
			}
		}
	}

	if (flag == false)
	{
		b[ib] = a[size - 1];
	}
	else
	{
		c[ic] = a[size - 1];
	}
```

```
	int bsize = ++ib;
	int csize = ++ic;
	i = ib = ic = 0;

	while (ib < bsize && ic < csize)
	{
		if (b[ib] <= c[ic])
		{
			a[i++] = b[ib++];
		}
		else
		{
			a[i++] = c[ic++];
		}
	}

	while (ib < bsize)
	{
		a[i++] = b[ib++];
	}

	while (ic < csize)
	{
		a[i++] = c[ic++];
	}

	cout << "b)";

	for (int i = 0; i < bsize; i++)
	{
		cout << b[i] << " ";
	}

	cout << endl << "c)";

	for (int i = 0; i < csize; i++)
	{
		cout << c[i] << " ";
	}

	cout << endl << "a)";

	for (int i = 0; i < size; i++)
	{
		cout << a[i] << " ";
	}

	cout << endl << endl;

	delete[] b;
	delete[] c;

	naturalMerge(a, size);
}
```

Блок-схема


void naturalMerge(int* a, int size)
bool flag = true;
int i = 0;
while (flag == true and i < size - 1)
if (a[i] > a[i + 1])
flag = false;
i++;
if (flag == true)
return;
int* b = new int[size];
int ib = 0;
int* c = new int[size];
int ic = -1;
for (int i = 0; i < size - 1; i++)
if (a[i] <= a[i + 1])
if (flag == false)
b[ib++] = a[i];
c[ic++] = a[i];
if (flag == false)
b[ib] = a[i];
flag = true;
ic++;
c[ic] = a[i];
flag = false;
ib++;
if (flag == false)
b[ib] = a[size - 1];
c[ic] = a[size - 1];
int bsize = ++ib;
int csize = ++ic;
i = ib = ic = 0;
while (ib < bsize && ic < csize)
if (b[ib] <= c[ic])
a[i++] = b[ib++];
a[i++] = c[ic++];
while (ib < bsize)
a[i++] = b[ib++];
while (ic < csize)
a[i++] = c[ic++];
cout << "b)";
for (int i = 0; i < bsize; i++)
cout << b[i] << " ";
cout << endl << "c)";
for (int i = 0; i < csize; i++)
cout << c[i] << " ";
cout << endl << "a)";
for (int i = 0; i < size; i++)
cout << a[i] << " ";
cout << endl << endl;
delete[] b;
delete[] c;
naturalMerge(a, size);
Конец

Сортировка многофазным слиянием

Код на C++

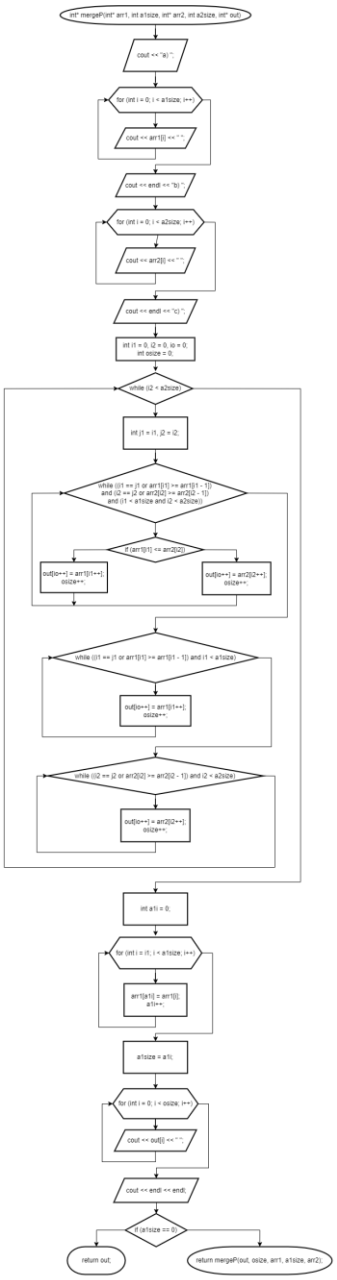
```
int* mergeP(int* arr1, int a1size, int* arr2, int a2size, int* out)
{
	cout << "a) ";

	for (int i = 0; i < a1size; i++)
	{
		cout << arr1[i] << " ";
	}

	cout << endl << "b) ";

	for (int i = 0; i < a2size; i++)
	{
		cout << arr2[i] << " ";
	}

	cout << endl << "c) ";

	int i1 = 0, i2 = 0, io = 0;
	int osize = 0;

	while (i2 < a2size)
	{
		int j1 = i1, j2 = i2;

		while ((i1 == j1 or arr1[i1] >= arr1[i1 - 1])
			and (i2 == j2 or arr2[i2] >= arr2[i2 - 1])
			and (i1 < a1size and i2 < a2size))
		{
			if (arr1[i1] <= arr2[i2])
			{
				out[io++] = arr1[i1++];
				osize++;
			}
			else
			{
				out[io++] = arr2[i2++];
				osize++;
			}
		}

		while ((i1 == j1 or arr1[i1] >= arr1[i1 - 1]) and i1 < a1size)
		{
			out[io++] = arr1[i1++];
			osize++;
		}

		while ((i2 == j2 or arr2[i2] >= arr2[i2 - 1]) and i2 < a2size)
		{
			out[io++] = arr2[i2++];
			osize++;
		}
	}

	int a1i = 0;

	for (int i = i1; i < a1size; i++)
	{
		arr1[a1i] = arr1[i];
		a1i++;
	}

	a1size = a1i;
```

```
	for (int i = 0; i < osize; i++)
	{
		cout << out[i] << " ";
	}

	cout << endl << endl;

	if (a1size == 0)
	{
		return out;
	}
	else
	{
		return mergeP(out, osize, arr1, a1size, arr2);
	}
}


int* polyphaseMerge(int* arr, int size)
{
	int count = 1;

	for (int i = 0; i < size - 1; i++)
	{
		if (arr[i] > arr[i + 1])
		{
			count++;
		}
	}

	if (count == 1)
	{
		return arr;
	}

	int fib0 = 0;
	int fib1 = 1;

	while (count > fib1)
	{
		int tmp = fib1;
		fib1 = fib0 + fib1;
		fib0 = tmp;
	}

	int* a = new int[size];
	int* b = new int[size];
	int* c = new int[size];
	int ia = 0, ib = 0, ic = 0, cnt = 0;
	int asize = 0, bsize = 0, csize = 0;

	for (int i = 0; i < size - 1; i++)
	{
		if (cnt < fib0)
		{
			a[ia++] = arr[i];
			asize++;
		}
		else
		{
			b[ib++] = arr[i];
			bsize++;
		}

		if (arr[i] > arr[i + 1])
		{
```

```
            cnt++;
      }
}


b[ib] = arr[size - 1];
bsize++;

cout << "a) ";

for (int i = 0; i < asize; i++)
{
      cout << a[i] << " ";
}

cout << endl << "b) ";

for (int i = 0; i < bsize; i++)
{
      cout << b[i] << " ";
}

cout << endl << "c) ";

ia = 0;
ib = 0;

for (int cnt = 0; cnt < fib1 - fib0; cnt++)
{
      int ja = ia, jb = ib;

      while ((ia == ja or a[ia] >= a[ia - 1])
            and (ib == jb or b[ib] >= b[ib - 1])
            and (ia < asize and ib < bsize))
      {
            if (a[ia] <= b[ib])
            {
                  c[ic++] = a[ia++];
                  csize++;
            }
            else
            {
                  c[ic++] = b[ib++];
                  csize++;
            }
      }

      while ((ia == ja or a[ia] >= a[ia - 1]) and ia < asize)
      {
            c[ic++] = a[ia++];
            csize++;
      }

      while ((ib == jb or b[ib] >= b[ib - 1]) and ib < bsize)
      {
            c[ic++] = b[ib++];
            csize++;
      }
}

int ai = 0;

for (int i = ia; i < asize; i++)
{
      a[ai] = a[i];
      ai++;
}
```

```
        asize = ai;

        for (int i = 0; i < csize; i++)
        {
              cout << c[i] << " ";
        }

        cout << endl << endl;

        if (asize == 0)
        {
              return c;
        }
        else
        {
              return mergeP(c, csize, a, asize, b);
        }

        delete[] a;
        delete[] b;
        delete[] c;
}
```

Блок-схема


int* mergeP(int* arr1, int a1size, int* arr2, int a2size, int* out)
cout << "a) ";
for (int i = 0; i < a1size; i++)
cout << arr1[i] << " ";
cout << endl << "b) ";
for (int i = 0; i < a2size; i++)
cout << arr2[i] << " ";
cout << endl << "c) ";
int i1 = 0, i2 = 0, io = 0,
int osize = 0;
while (i2 < a2size)
int j1 = i1, j2 = i2;
while ((j1 == j1 or arr1[j1] >= arr1[j1 - 1])
and (j2 == j2 or arr2[j2] >= arr2[j2 - 1])
and (i1 < a1size and i2 < a2size))
if (arr1[i1] <= arr2[i2])
out[io++] = arr1[i1++];
osize++;
out[io++] = arr2[i2++];
osize++;
while ((j1 == j1 or arr1[j1] >= arr1[j1 - 1]) and (i1 < a1size)
out[io++] = arr1[i1++];
osize++;
while ((j2 == j2 or arr2[j2] >= arr2[j2 - 1]) and (i2 < a2size)
out[io++] = arr2[i2++];
osize++;
int a1i = 0;
for (int i = i1; i < a1size; i++)
arr1[a1i] = arr1[i];
a1i++;
a1size = a1i;
for (int i = 0; i < osize; i++)
cout << out[i] << " ";
cout << endl << endl;
if (a1size == 0)
return out;
return mergeP(out, osize, arr1, a1size, arr2);


int* polyphaseMerge(int* arr, int size)
int count = 1;
for (int i = 0; i < size - 1; i++)
if (arr[i] > arr[i + 1])
count++;
if (count == 1)
return arr;
int fib0 = 0;
int fib1 = 1;
while (count > fib1)
int tmp = fib1;
fib1 = fib0 + fib1;
fib0 = tmp;
int* a = new int[size];
int* b = new int[size];
int* c = new int[size];
int ia = 0, ib = 0, ic = 0, cnt = 0;
int asize = 0, bsize = 0, csize = 0;
for (int i = 0; i < size - 1; i++)
if (cnt < fib0)
a[ia++] = arr[i];
asize++;
b[ib++] = arr[i];
bsize++;
if (arr[i] > arr[i + 1])
cnt++;
b[ib] = arr[size - 1];
bsize++;
cout << "a) ";
for (int i = 0; i < asize; i++)
cout << a[i] << " ";
cout << endl << "b) ";
for (int i = 0; i < bsize; i++)
cout << b[i] << " ";
cout << endl << "c) ";
ia = 0;
ib = 0;
for (int cnt = 0; cnt < fib1 - fib0; cnt++)
int ja = ia, jb = ib;
while ((ia == ja or a[ia] >= a[ia - 1])
and (ib == jb or b[ib] >= b[ib - 1])
and (ia < asize and ib < bsize))
if (a[ia] <= b[ib])
c[ic++] = a[ia++];
csize++;
c[ic++] = b[ib++];
csize++;
while ((ia == ja or a[ia] >= a[ia - 1]) and ia < asize)
c[ic++] = a[ia++];
csize++;
while ((ib == jb or b[ib] >= b[ib - 1]) and ib < bsize)
c[ic++] = b[ib++];
csize++;
int ai = 0;
for (int i = ia; i < asize; i++)
a[ai] = a[i];
ai++;
asize = ai;
for (int i = 0; i < csize; i++)
cout << c[i] << " ";
cout << endl << endl;
if (asize == 0)
return c;
return mergeP(c, csize, a, asize, b);
delete[] a;
delete[] b;
delete[] c;
Конец

Меню

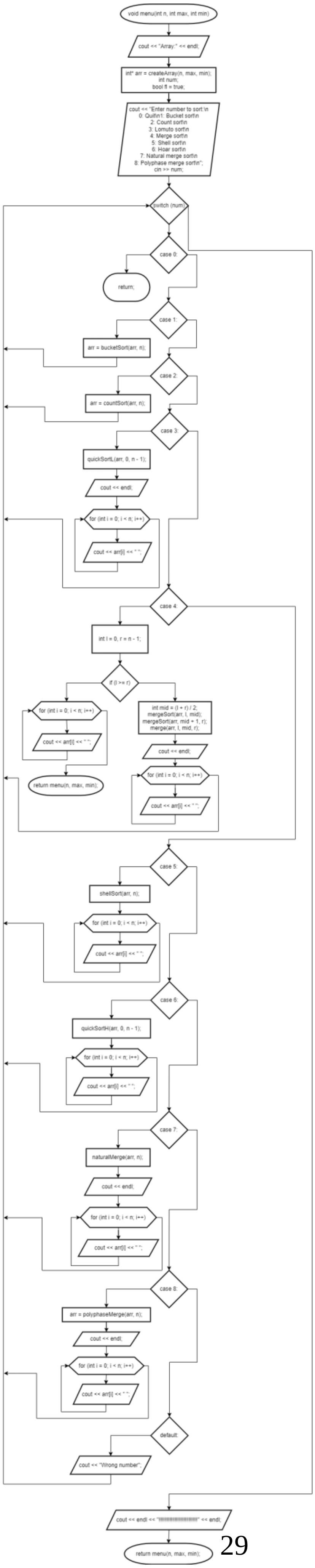
Код на C++

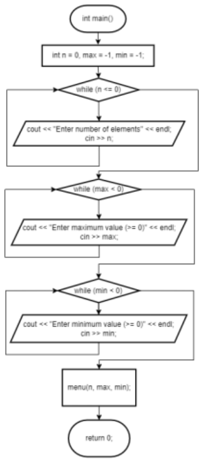
```
void menu(int n, int max, int min)
{
	cout << "Array:" << endl;

	int* arr = createArray(n, max, min);
	int num;
	bool fl = true;

	cout << "Enter number to sort:\n0: Quit\n1: Bucket sort\n2: Count sort\n3: Lomuto sort\n4: Merge sort\n5: Shell sort\n6: Hoar sort\n7: Natural merge sort\n8: Polyphase merge sort\n";
	cin >> num;

	switch (num)
	{
	case 0: return; break;
	case 1: arr = bucketSort(arr, n); break;
	case 2: arr = countSort(arr, n); break;
	case 3:
	{
		quickSortL(arr, 0, n - 1);

		cout << endl;

		for (int i = 0; i < n; i++)
		{
			cout << arr[i] << " ";
		}

		break;
	}
	case 4:
	{
		int l = 0, r = n - 1;
		if (l >= r)
		{
			for (int i = 0; i < n; i++)
			{
				cout << arr[i] << " ";
			}

			return menu(n, max, min);
		}
		int mid = (l + r) / 2;
		mergeSort(arr, l, mid);
		mergeSort(arr, mid + 1, r);
		merge(arr, l, mid, r);

		cout << endl;

		for (int i = 0; i < n; i++)
		{
			cout << arr[i] << " ";
		}

		break;
	}
	case 5:
	{
		shellSort(arr, n);

		for (int i = 0; i < n; i++)
		{
```

```
                cout << arr[i] << " ";
            }

            break;
        }
        case 6:
        {
            quickSortH(arr, 0, n - 1);

            for (int i = 0; i < n; i++)
            {
                cout << arr[i] << " ";
            }

            break;
        }
        case 7:
        {
            naturalMerge(arr, n);

            cout << endl;

            for (int i = 0; i < n; i++)
            {
                cout << arr[i] << " ";
            }

            break;
        }
        case 8:
        {
            arr = polyphaseMerge(arr, n);

            cout << endl;

            for (int i = 0; i < n; i++)
            {
                cout << arr[i] << " ";
            }

            break;
        }
        default: cout << "Wrong number";
        }

        cout << endl << "!!!!!!!!!!!!!!!!!!!!!!!!" << endl;

        return menu(n, max, min);
}


int main()
{
        int n = 0, max = -1, min = -1;

        while (n <= 0)
        {
            cout << "Enter number of elements" << endl;
            cin >> n;
        }

        while (max < 0)
        {
            cout << "Enter maximum value (>= 0)" << endl;
            cin >> max;
        }
```

```
        while (min < 0)
        {
                cout << "Enter minimum value (>= 0)" << endl;
                cin >> min;
        }

        menu(n, max, min);

        return 0;
}
```

Блок-схема


void menu(int n, int max, int min)
cout << "Array:" << endl;
int* arr = createArray(n, max, min);
int num;
bool fl = true;
cout << "Enter number to sort:\n
0: Quit\n1: Bucket sort\n
2: Count sort\n
3: Lomuto sort\n
4: Merge sort\n
5: Shell sort\n
6: Hoar sort\n
7: Natural merge sort\n
8: Polyphase merge sort\n";
cin >> num;
switch (num)
case 0:
return;
case 1:
arr = bucketSort(arr, n);
case 2:
arr = countSort(arr, n);
case 3:
quickSortL(arr, 0, n - 1);
cout << endl;
for (int i = 0; i < n; i++)
cout << arr[i] << " ";
case 4:
int l = 0, r = n - 1;
if (l >= r)
for (int i = 0; i < n; i++)
cout << arr[i] << " ";
return menu(n, max, min);
int mid = (l + r) / 2;
mergeSort(arr, l, mid);
mergeSort(arr, mid + 1, r);
merge(arr, l, mid, r);
cout << endl;
for (int i = 0; i < n; i++)
cout << arr[i] << " ";
case 5:
shellSort(arr, n);
for (int i = 0; i < n; i++)
cout << arr[i] << " ";
case 6:
quickSortH(arr, 0, n - 1);
for (int i = 0; i < n; i++)
cout << arr[i] << " ";
case 7:
naturalMerge(arr, n);
cout << endl;
for (int i = 0; i < n; i++)
cout << arr[i] << " ";
case 8:
arr = polyphaseMerge(arr, n);
cout << endl;
for (int i = 0; i < n; i++)
cout << arr[i] << " ";
default:
cout << "Wrong number";
cout << endl << "!!!!!!!!!!!!!!!!!!!!!!!" << endl;
return menu(n, max, min);


int main()
int n = 0, max = -1, min = -1;
while (n <= 0)
cout << "Enter number of elements" << endl;
cin >> n;
while (max < 0)
cout << "Enter maximum value (>= 0)" << endl;
cin >> max;
while (min < 0)
cout << "Enter minimum value (>= 0)" << endl;
cin >> min;
menu(n, max, min);
return 0;