Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный технологический университет»

**РЕФЕРАТ**

на тему:

“Очередь с приоритетом”

Выполнил:

Халалеенко Андрей Николаевич

Проверил:

Белодед Николай Иванович

Минск 2022

Оглавление

1. Сортировка - это
2. Рассматриваемые виды сортировок
3. Пузырьковая сортировка
4. Сортировка выбором
5. Сортировка вставками
6. Быстрая сортировка
7. Вывод
   * + 1. **Сортировка – это**

Сортировка – это последовательное расположение или разбиение на группы чего-либо в зависимости от выбранного критерия.

Сортировка распределяет элементы в порядке, удобном для работы. Если отсортировать массив чисел в порядке убывания, то первый элемент всегда будет наибольшим, а последний наименьшим. Поэтому желательно хранить информацию упорядочено, чтобы было проще проводить над ней операции.

* + - 1. **Рассматриваемые виды сортировок**

Пузырьковая сортировка (Bubble sort);

Сортировка выбором (Selection sort);

Сортировка вставками (Insertion sort);

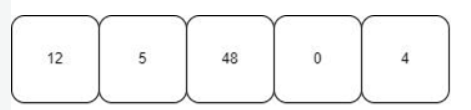
Быстрая сортировка (Quick sort);

* + - 1. **Пузырьковая сортировка**

В пузырьковой сортировке каждый элемент сравнивается со следующим. Если два таких элемента не стоят в нужном порядке, то они меняются между собой местами. В конце каждой итерации (далее называем их проходами) наибольший/наименьший элемент ставится в конец списка.

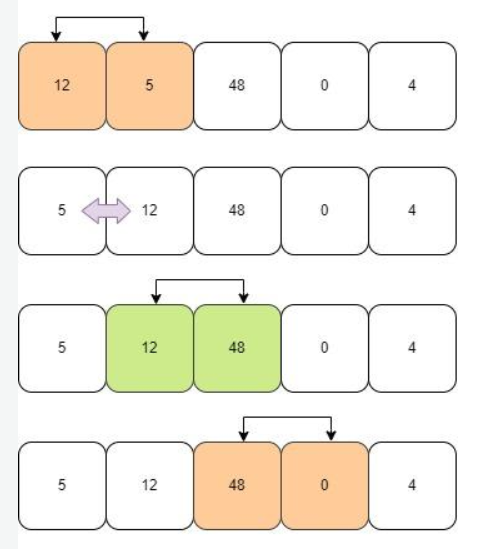
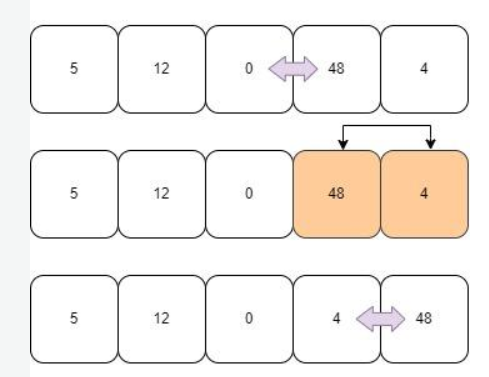
Пример:

Имеем массив из пяти элементов

****

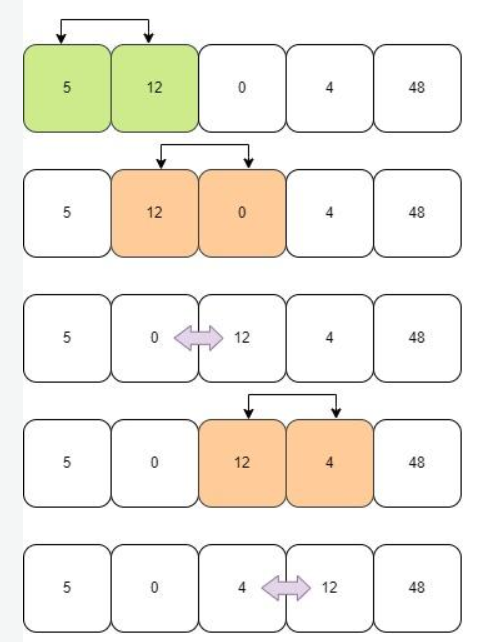
Оранжевым отмечаются элементы, которые нужно поменять местами. Зеленые уже стоят в нужном порядке.

Проход 1:

**** ****

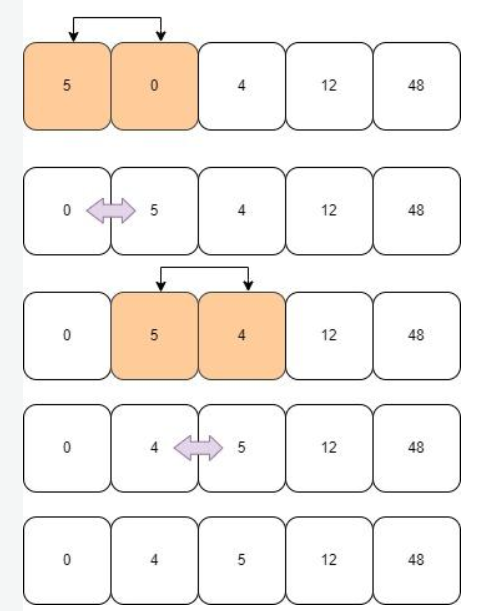
Наибольший элемент — число 48 — оказался в конце списка.

Проход 2:

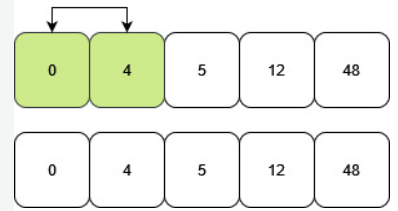


Наибольший элемент уже занимает место в конце массива. Чтобы поставить следующее число по убыванию, можно пройтись лишь до 4-й позиции, а не пятой.

Проход 3:

****

Проход 4:



После четвертого прохода получаем отсортированный массив.

Реализация пузырьковой сортировки на С/С++

#include <iostream>

using namespace std;

void bubbleSort(int list[], int listLength)

{

while (listLength--)

{

bool swapped = false;

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

{

if (list[i] > list[i + 1])

{

swap(list[i], list[i + 1]);

swapped = true;

}

}

if (swapped == false)

break;

}

}

int main()

{

int list[5] = { 3,19,8,0,48 };

cout << "Input array ..." << endl;

for (int i = 0; i < 5; i++)

cout << list[i] << '\t';

cout << endl;

bubbleSort(list, 5);

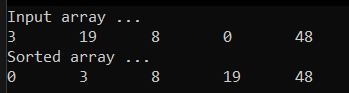
cout << "Sorted array ..." << endl;

for (int i = 0; i < 5; i++)

cout << list[i] << '\t';

cout << endl;

}



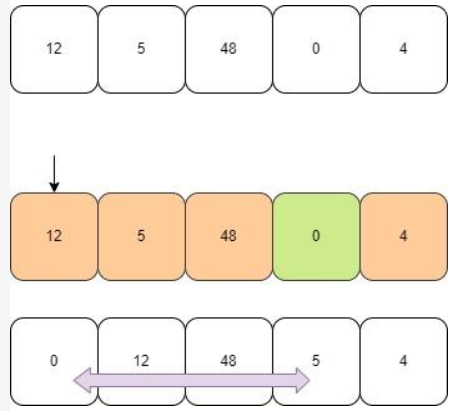
* + - 1. **Сортировка выбором**

Находится наименьшее значение в массиве и ставится на позицию, откуда начали проход. Потом двигаемся на следующую позицию.

Пример:

Возьмем тот же массив из пяти элементов и отсортируем его.

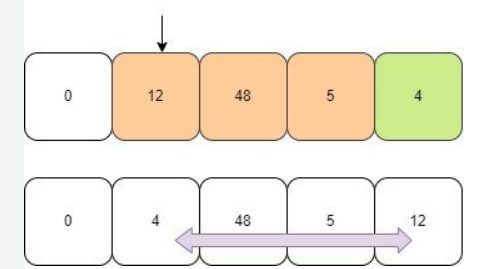
Проход 1:

****

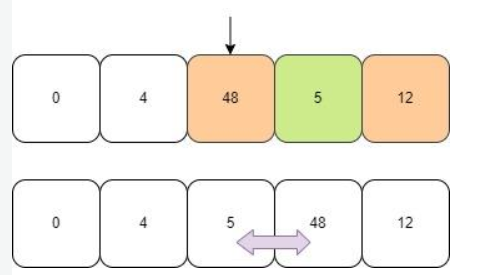
Зеленым отмечается наименьший элемент в подмассиве — он ставится в начало списка.

Проход 2:

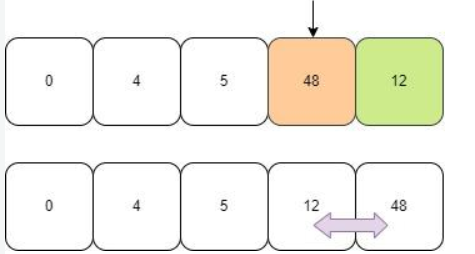
Число 4 — наименьшее в оставшейся части массива. Перемещаем четверку на вторую позицию после числа 0.



Проход 3:

****

Проход 4:



Реализация сортировки выбором на С/С++

#include <iostream>

using namespace std;

int findSmallestPosition(int list[], int startPosition, int listLength)

{

int smallestPosition = startPosition;

for (int i = startPosition; i < listLength; i++)

{

if (list[i] < list[smallestPosition])

smallestPosition = i;

}

return smallestPosition;

}

void selectionSort(int list[], int listLength)

{

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

{

int smallestPosition = findSmallestPosition(list, i, listLength);

swap(list[i], list[smallestPosition]);

}

return;

}

int main()

{

int list[5] = { 12, 5, 48, 0, 4 };

cout << "Input array ..." << endl;

for (int i = 0; i < 5; i++)

cout << list[i] << '\t';

cout << endl;

selectionSort(list, 5);

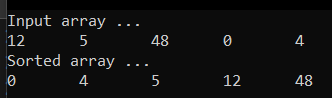
cout << "Sorted array ..." << endl;

for (int i = 0; i < 5; i++)

cout << list[i] << '\t';

cout << endl;

}

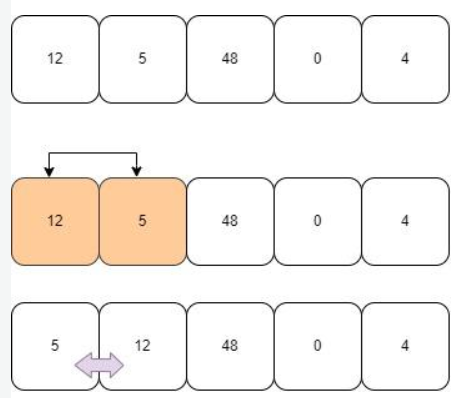


* + - 1. **Сортировка вставками**

В сортировке вставками начинаем со второго элемента. Проверяем между собой второй элемент с первым и, если надо, меняем местами. Сравниваем следующую пару элементов и проверяем все пары до нее.

Проход 1:

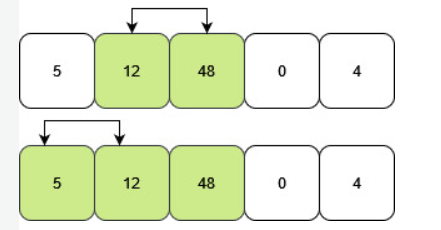
Начинаем со второй позиции



12 > 5 - элементы меняются местами

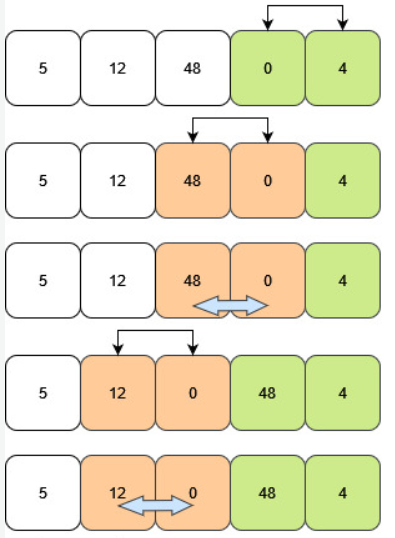
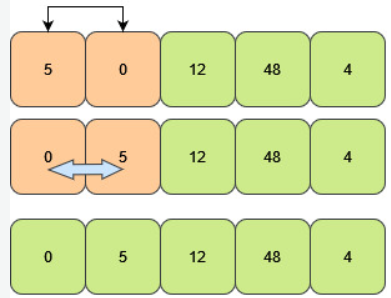
Проход 2:

Начинаем с третьей позиции. Проверяем вторую и третью позиции. Затем первую и вторую.



Проход 3:

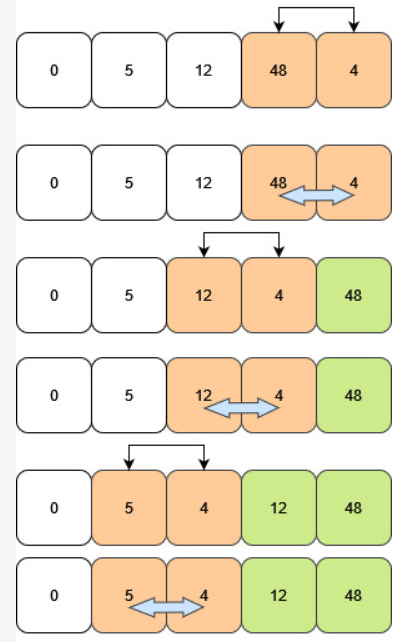
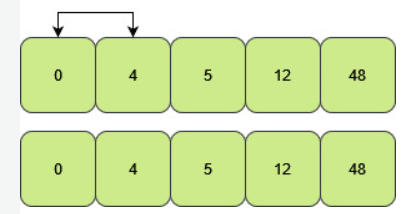
Начинаем с четвёртой позиции

Произошло три смены местами

Проход 4:

Начинаем с последней позиции

Реализация сортировки вставками на С/С++

#include <iostream>

using namespace std;

void insertionSort(int list[], int listLength)

{

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

{

int j = i - 1;

while (j >= 0 && list[j] > list[j + 1])

{

swap(list[j], list[j + 1]);

cout << "\ndid";

j--;

}

}

}

int main()

{

int list[8] = { 3,19,8,0,48,4,5,12 };

cout << "Input array ...\n";

for (int i = 0; i < 8; i++)

{

cout << list[i] << "\t";

}

insertionSort(list, 8);

cout << "\n\nSorted array ... \n";

for (int i = 0; i < 8; i++)

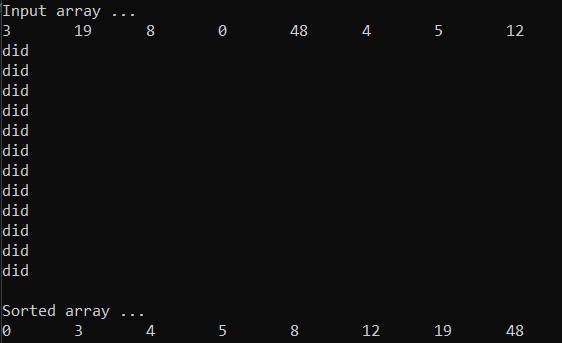
{

cout << list[i] << "\t";

}

return 0;

}

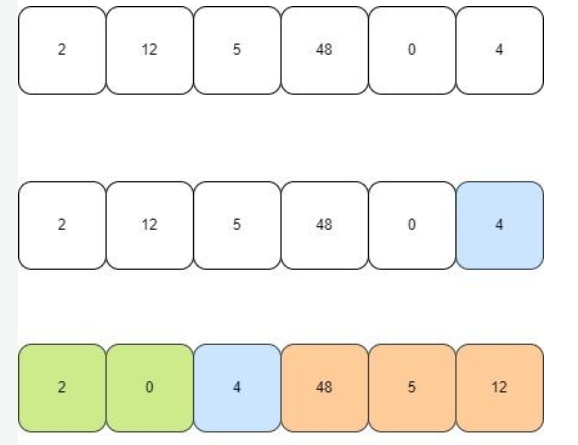


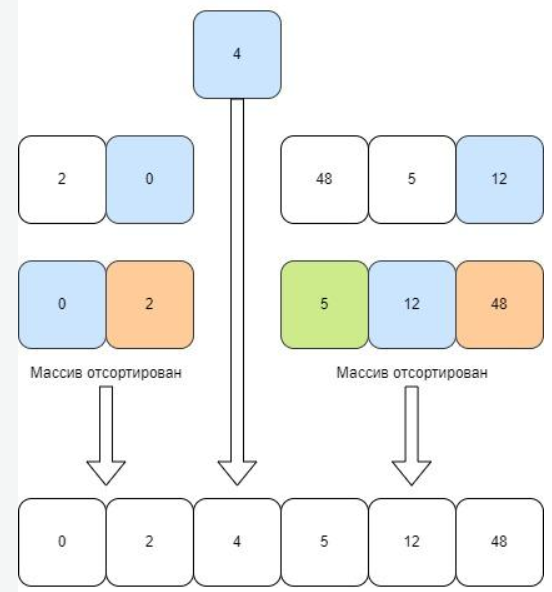
* + - 1. **Быстрая сортировка**

В основе быстрой сортировки лежит стратегия «разделяй и властвуй». Задача разделяется на более мелкие подзадачи. Подзадачи решаются отдельно, а потом решения объединяют. Точно так же, массив разделяется на подмассивы, которые сортируются и затем сливаются в один.

Пример:

В первую очередь выбираем опорный элемент. Отметим его синим. Все значения больше опорного элемента ставятся после него, остальные — перед.

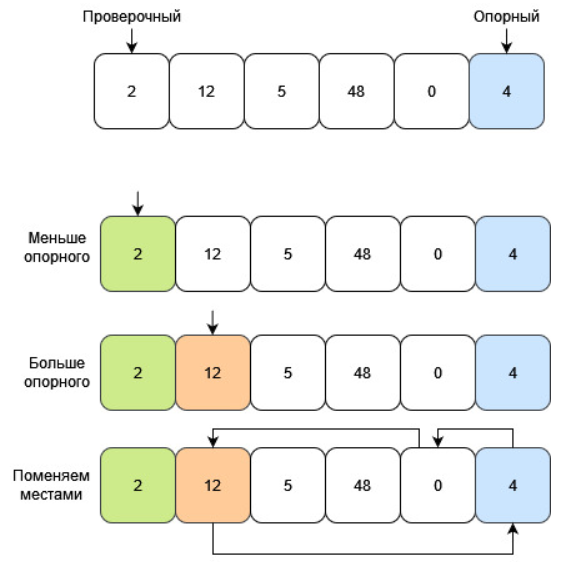


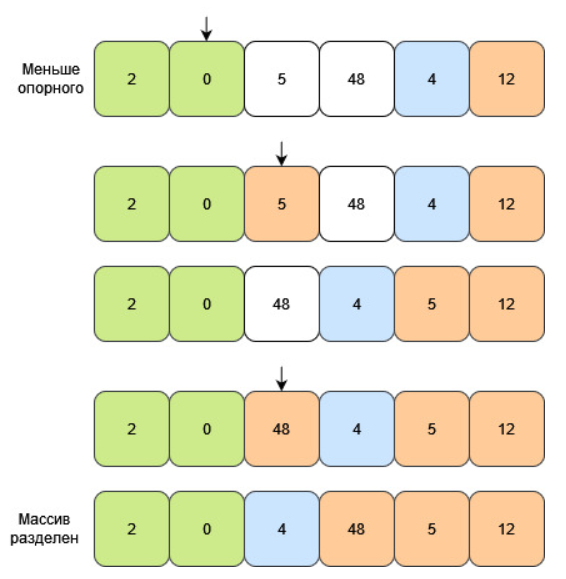


Массив разделяется по опорному элементу. В полученных массивах также выбираем опорный элемент и разделяем по нему.

Опорным может быть любой элемент. В данном случае выбирается последний в списке.

Чтобы расположить элементы большие — справа от опорного элемента, а меньшие — слева, будем двигаться от начала списка. Если число будет больше опорного, то оно ставится на его место, а сам опорный на место перед ним.





Реализация быстрой сортировки на С/С++

#include <stdio.h>

#include <stdlib.h>

#define SIZE 20

// Функция быстрой сортировки

void quickSort(int\* numbers, int left, int right)

{

int pivot; // разрешающий элемент

int l\_hold = left; //левая граница

int r\_hold = right; // правая граница

pivot = numbers[left];

while (left < right) // пока границы не сомкнутся

{

while ((numbers[right] >= pivot) && (left < right))

right--; // сдвигаем правую границу пока элемент [right] больше [pivot]

if (left != right) // если границы не сомкнулись

{

numbers[left] = numbers[right]; // перемещаем элемент [right] на место разрешающего

left++; // сдвигаем левую границу вправо

}

while ((numbers[left] <= pivot) && (left < right))

left++; // сдвигаем левую границу пока элемент [left] меньше [pivot]

if (left != right) // если границы не сомкнулись

{

numbers[right] = numbers[left]; // перемещаем элемент [left] на место [right]

right--; // сдвигаем правую границу влево

}

}

numbers[left] = pivot; // ставим разрешающий элемент на место

pivot = left;

left = l\_hold;

right = r\_hold;

if (left < pivot) // Рекурсивно вызываем сортировку для левой и правой части массива

quickSort(numbers, left, pivot - 1);

if (right > pivot)

quickSort(numbers, pivot + 1, right);

}

int main()

{

int a[SIZE];

printf("First: ");

// Заполнение массива случайными числами

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

a[i] = rand() % 201 - 100;

// Вывод элементов массива до сортировки

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

printf("%4d ", a[i]);

printf("\n");

quickSort(a, 0, SIZE - 1); // вызов функции сортировки

// Вывод элементов массива после сортировки

printf("New: ");

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

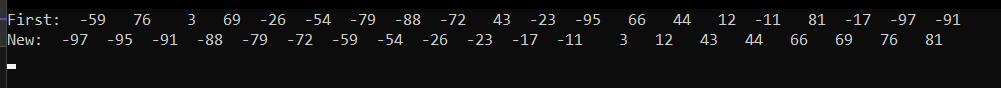
printf("%4d ", a[i]);

printf("\n");

getchar();

return 0;

}



* + - 1. **Вывод**

Каждая из сортировок по-своему хороша. Где-то удобнее использовать один алгоритм сортировки, где-то - другой. У каждой из них есть своё преимущество над другими в той или иной задаче.