Государственный Университет Молдовы

Факультет Математики и Информатики

Кафедра Информатики

**Лабораторная работа №2**

по курсу «Алгоритмы и Структуры Данных»

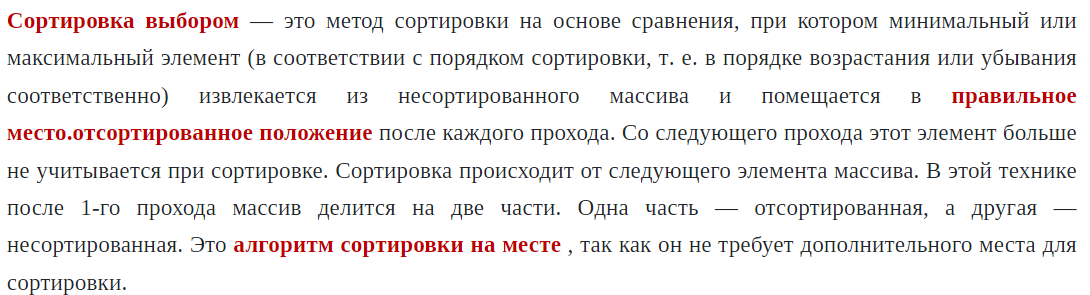
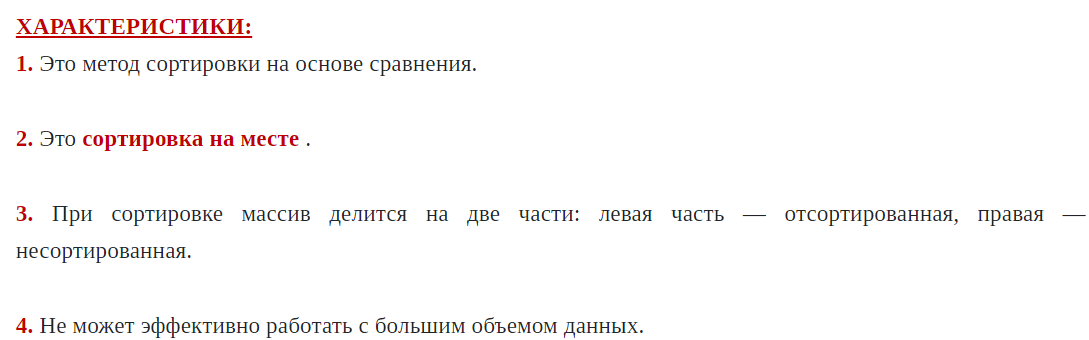
Методы сортировки.

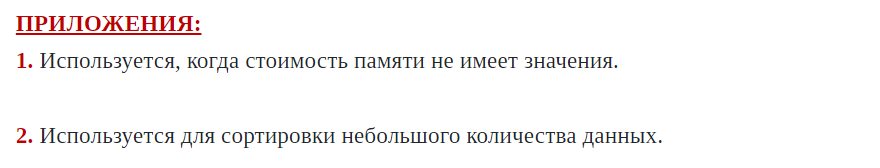
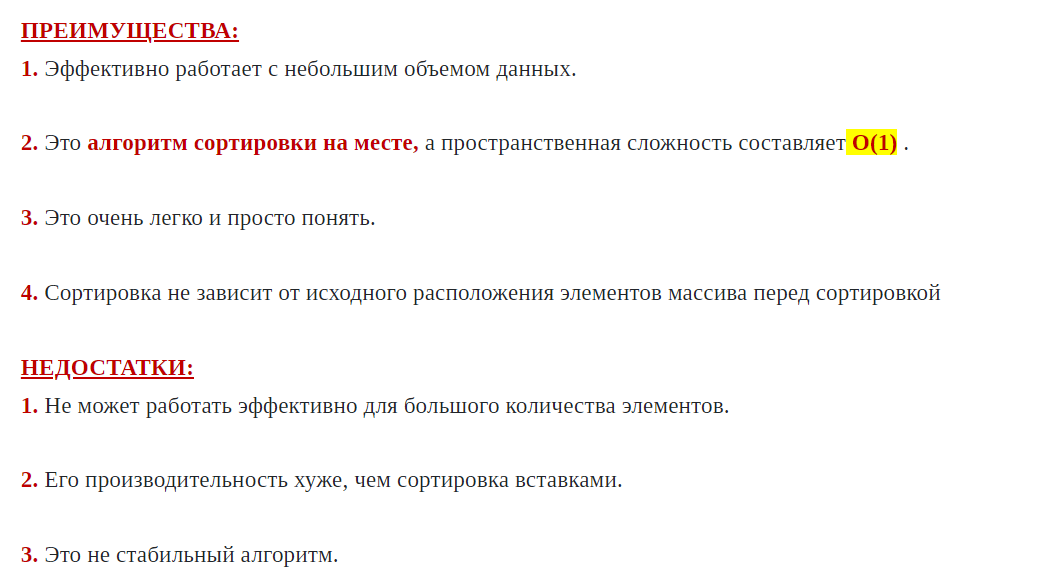
Выполнил студент группы I2102:

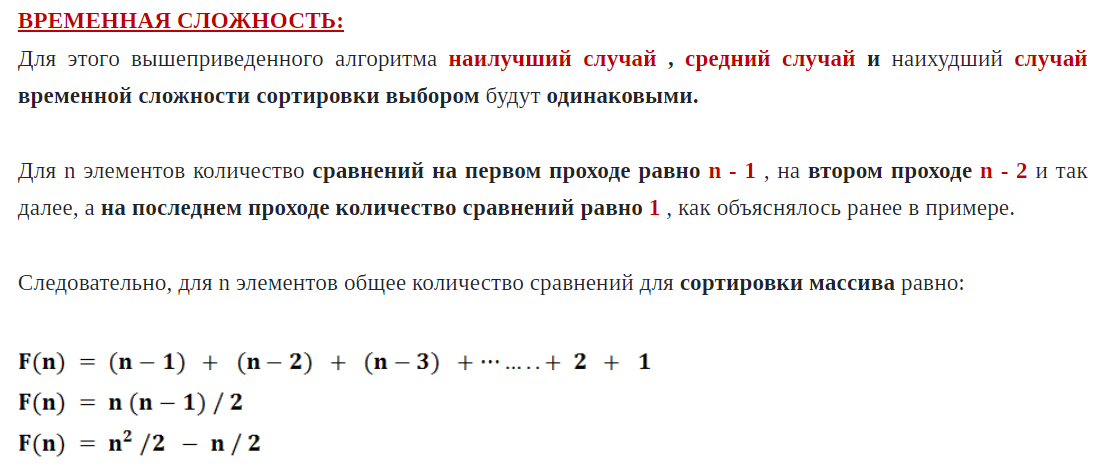
**Король Владимир**

Проверил преподаватель:

**Кроитор Михаил**



int i, j, t, min, pos;

for (i = 0; i < NM; i++) {

min = notebooks[i].price;

pos = i;

for (j = i + 1; j < NM; j++) {

if (notebooks[j].price < min) {

min = notebooks[j].price;

pos = j;

}

}

t = notebooks[i].price;

notebooks[i].price = notebooks[pos].price;

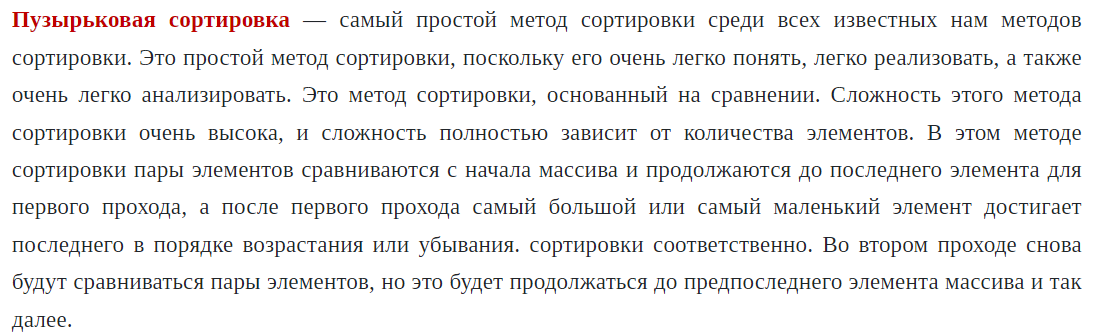
notebooks[pos].price = t;

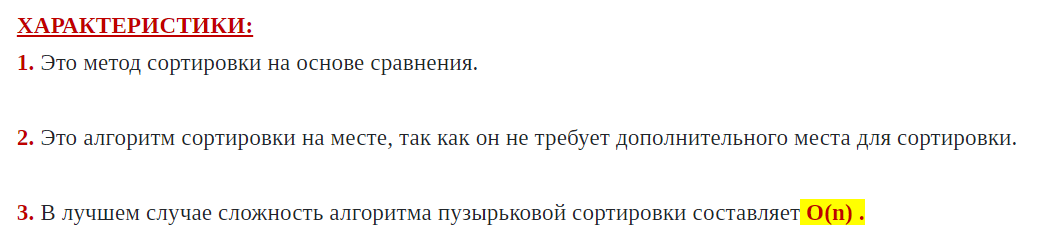
}

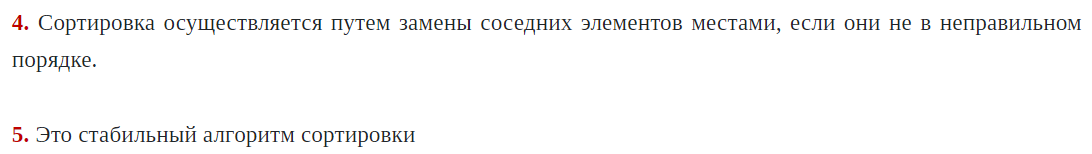
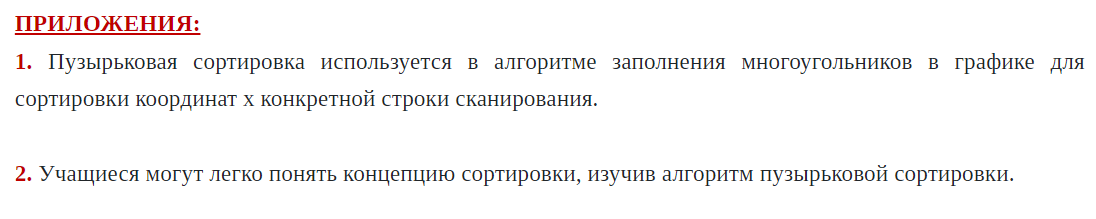
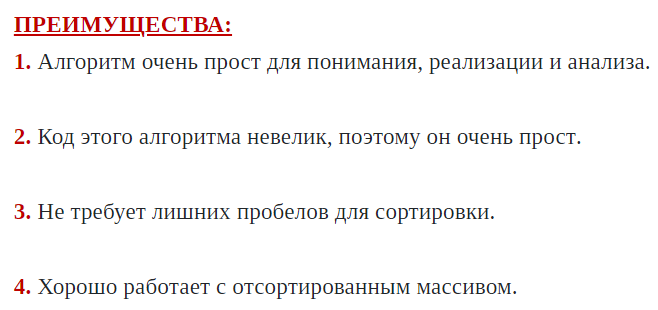
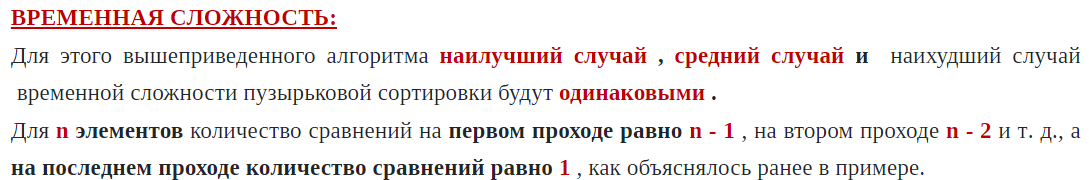
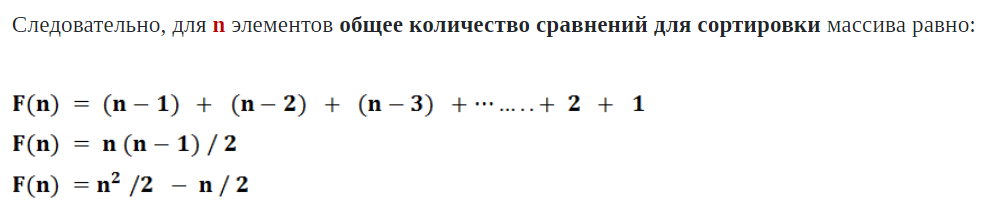
for (int i = 0; i < NM; i++) {

cout << notebooks[i].id << "/" << notebooks[i].name << "/" << notebooks[i].year << "/" << notebooks[i].price << "/" << notebooks[i].ram << endl;

}





int i, j, t;

for (i = 0; i < NM; i++) {

for (j = 0; j < NM - i - 1; j++) {

if (notebooks[j].price > notebooks[j + 1].price) {

t = notebooks[j].price;

notebooks[j].price = notebooks[j + 1].price;

notebooks[j + 1].price = t;

}

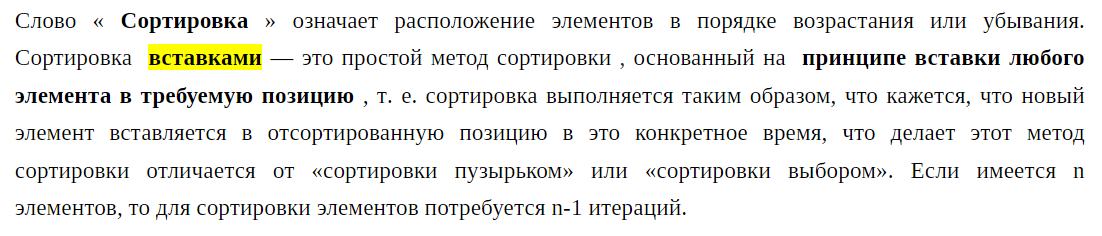
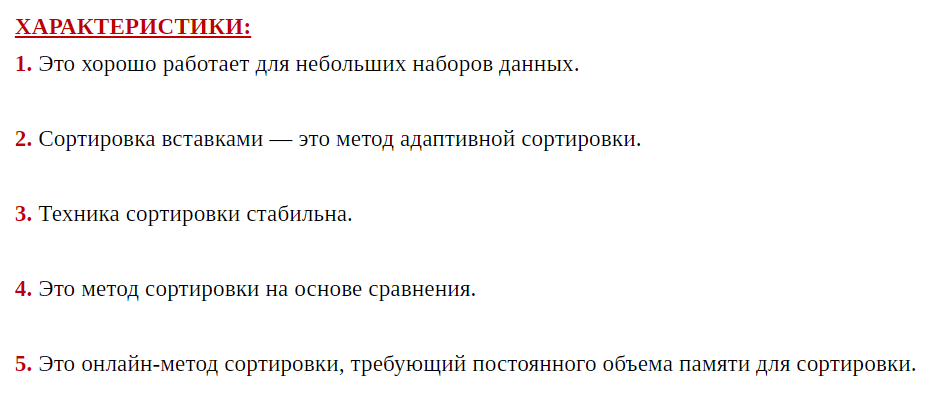
}

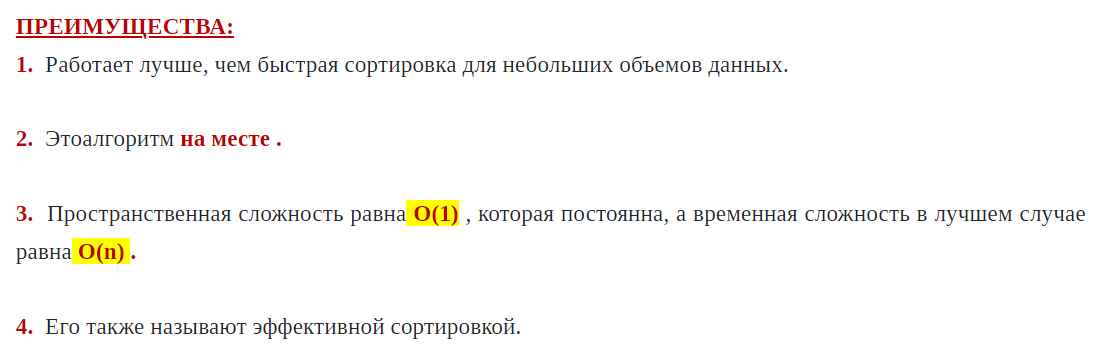
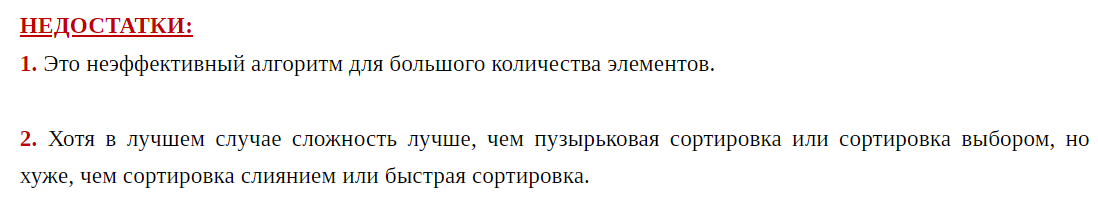
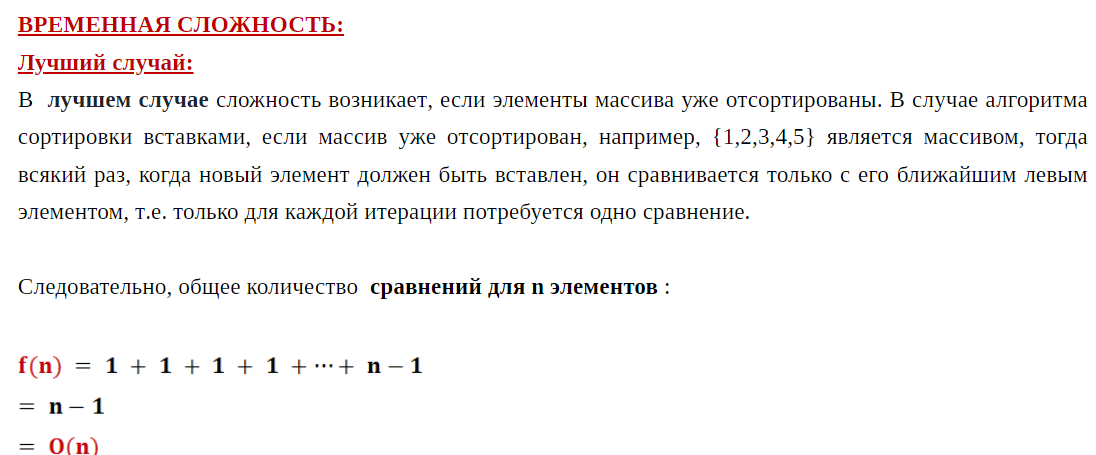
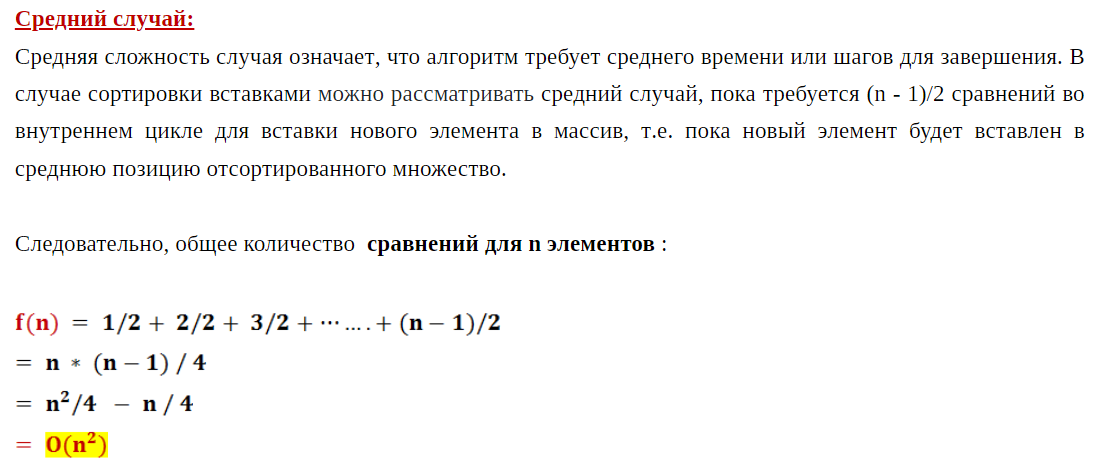
}

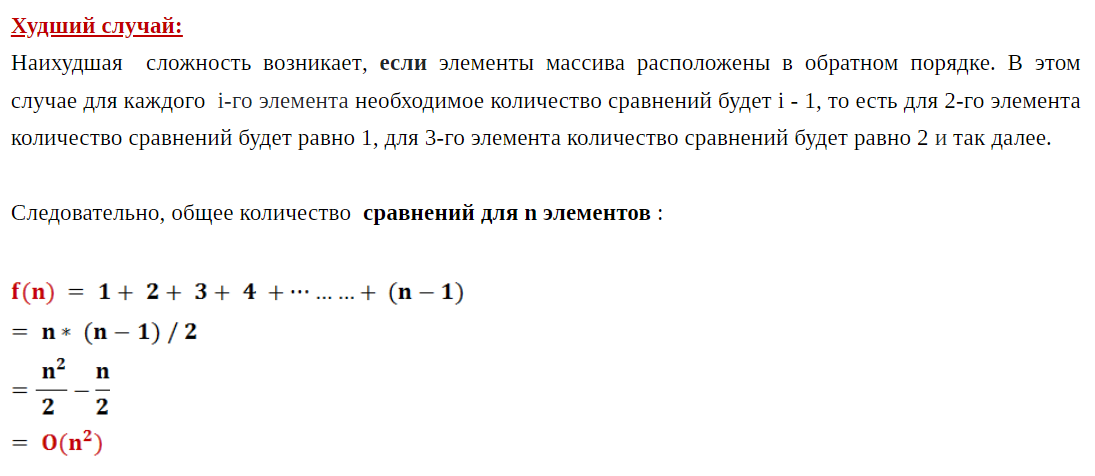
for (int i = 0; i < NM; i++) {

cout << notebooks[i].id << "/" << notebooks[i].name << "/" << notebooks[i].year << "/" << notebooks[i].price << "/" << notebooks[i].ram << endl;

}



int i, j, t;

for (i = 1; i < NM; i++) {

t = notebooks[i].price; // сохранение вновь вставленного элемента во временнуб переменную t

for (j = i - 1; j >= 0; j--) {

if (t < notebooks[j].price) // замена элементов местами, если левый элемент больше, чем нвоый элмент

notebooks[j + 1].price = notebooks[j].price;

else

break;

}

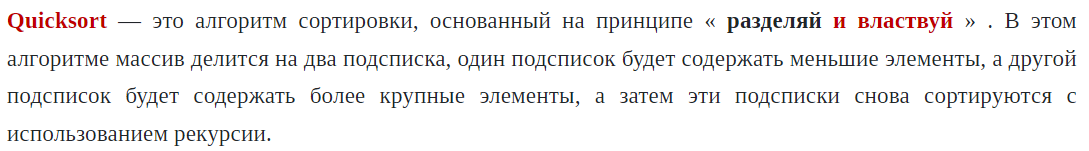
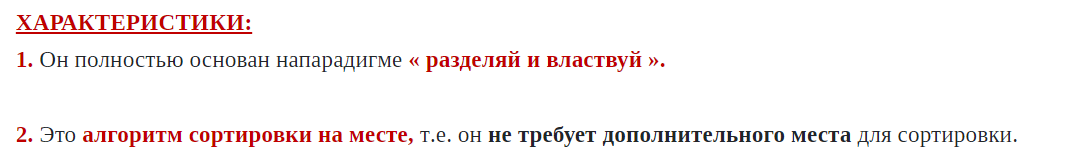
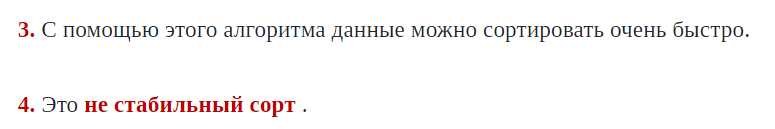
notebooks[j + 1].price = t;

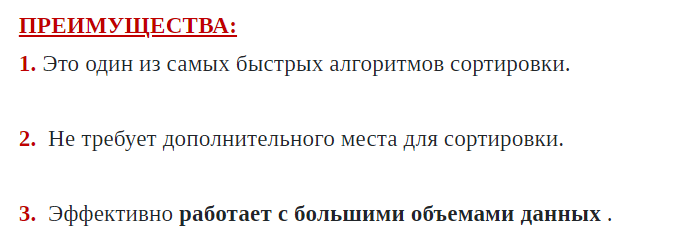
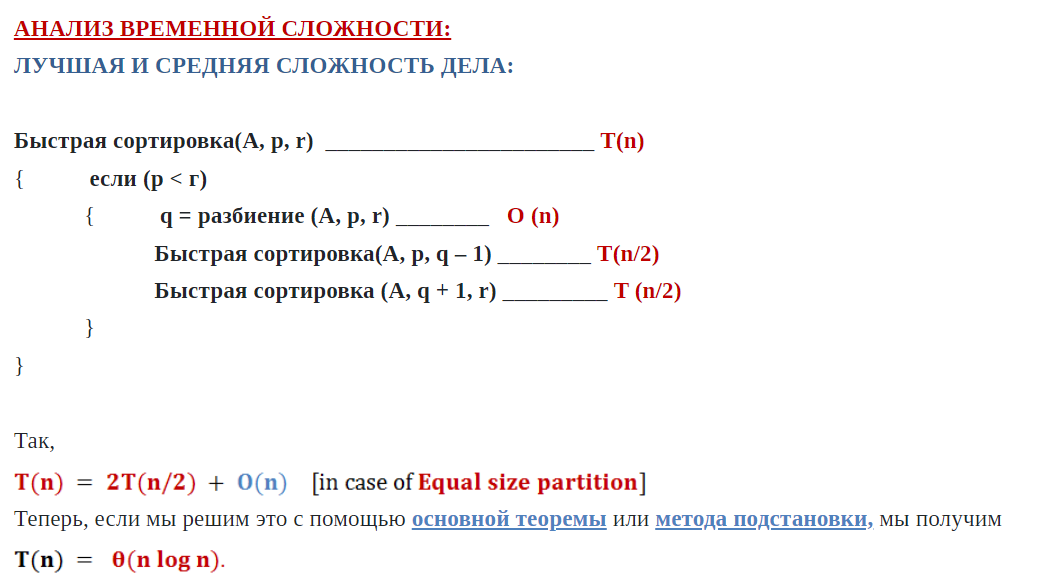
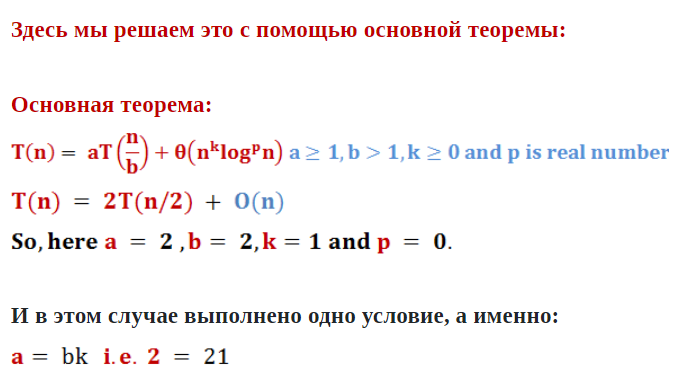
}

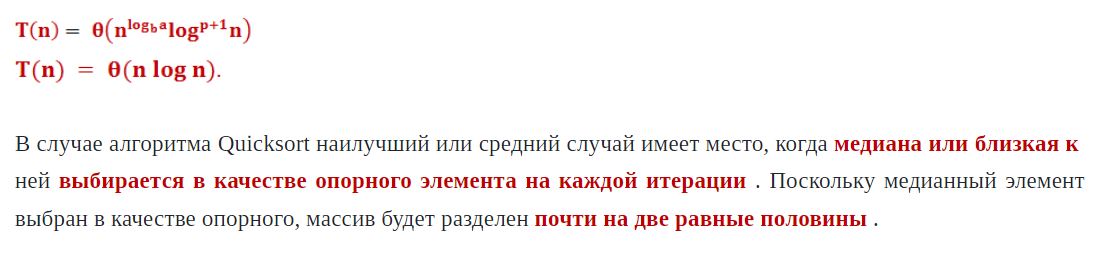
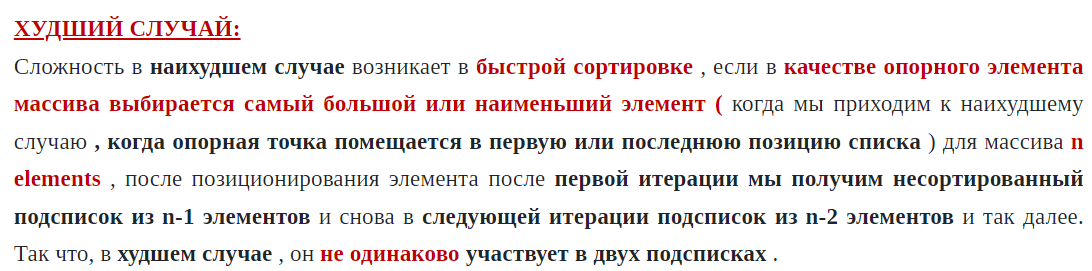
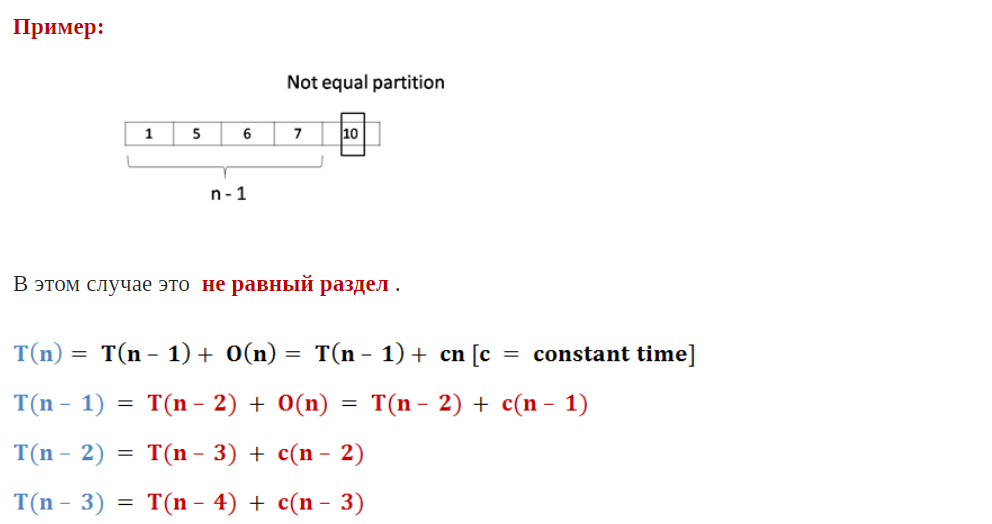
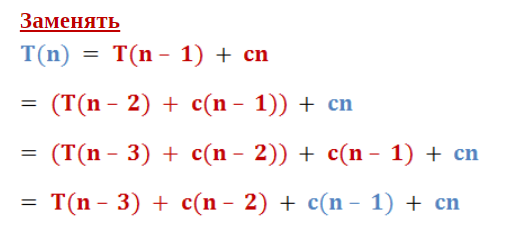
for (int i = 0; i < NM; i++) {

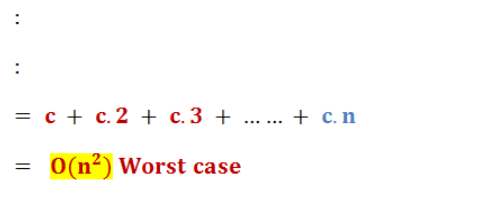
cout << notebooks[i].id << "/" << notebooks[i].name << "/" << notebooks[i].year << "/" << notebooks[i].price << "/" << notebooks[i].ram << endl;

}



void quickSort(NOTEBOOK\* notebooks, int left, int right) {

int pivot; // разрешающий элемент

int l\_hold = left; // левая граница

int r\_hold = right; // правая граница

pivot = notebooks[left].price;

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

while ((notebooks[right].price >= pivot) && (left < right))

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

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

notebooks[left].price = notebooks[right].price; // перемещаем элемент [right] на место разрешающего

left++; // сдвигаем левую границу вправо

}

while ((notebooks[left].price <= pivot) && (left < right))

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

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

notebooks[right].price = notebooks[left].price; // перемещаем эелемент [left] на место [right]

right--; // сдвигаем правую границу влево

}

}

notebooks[left].price = pivot; // ставим разрешающий элемент на место

pivot = left;

left = l\_hold;

right = r\_hold;

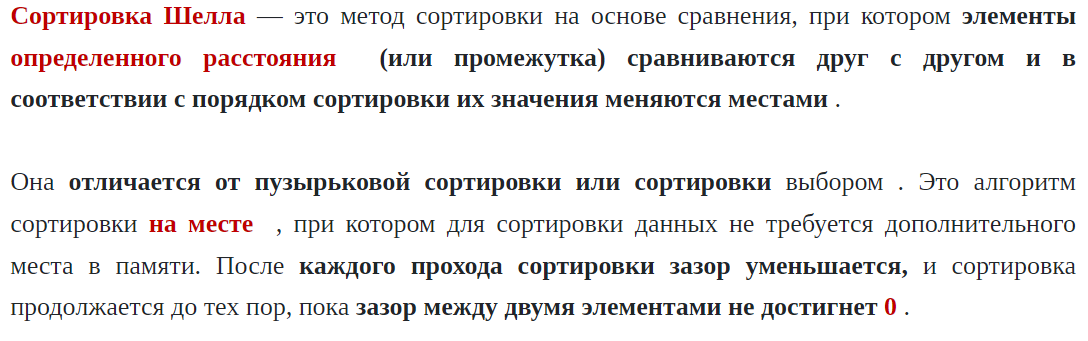
if (left < pivot) // рекурсивно вызываем сортировку для левой и правой части массива

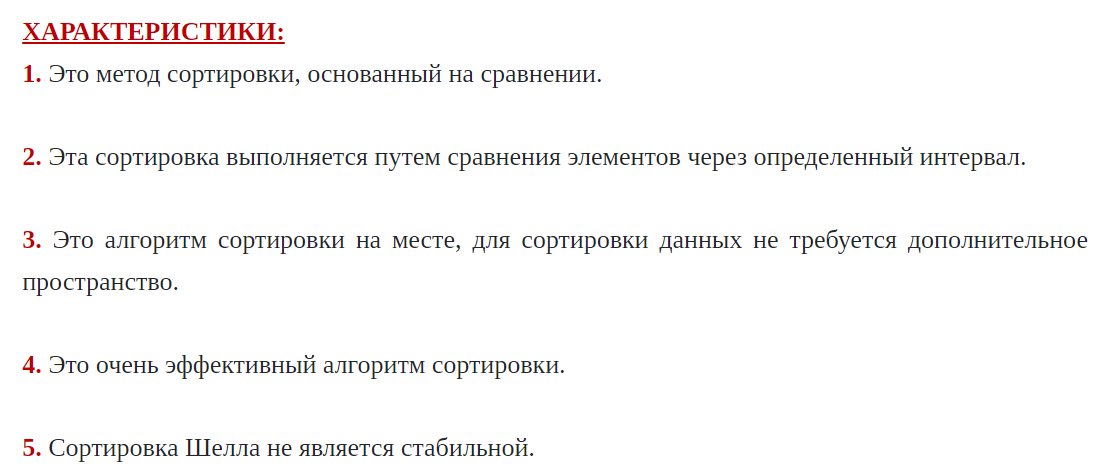
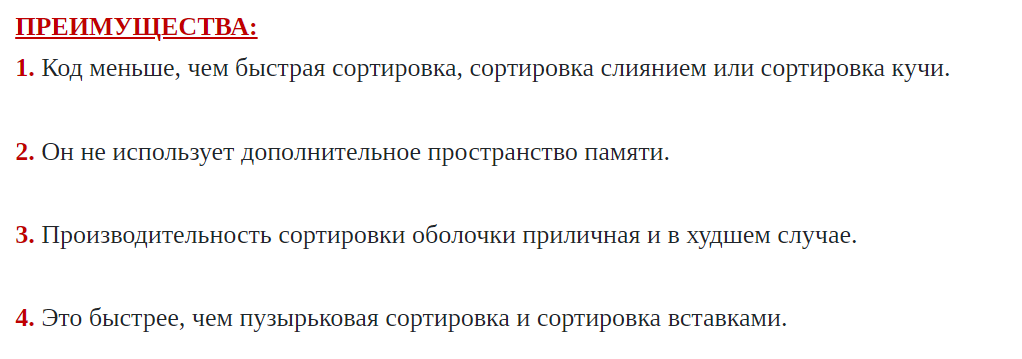
quickSort(notebooks, left, pivot - 1);

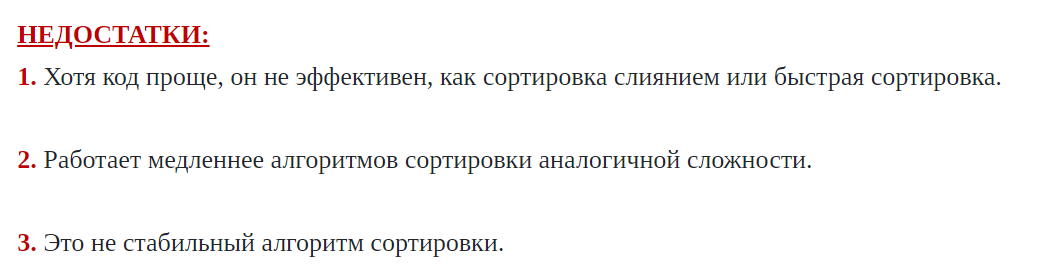
if (right > pivot)

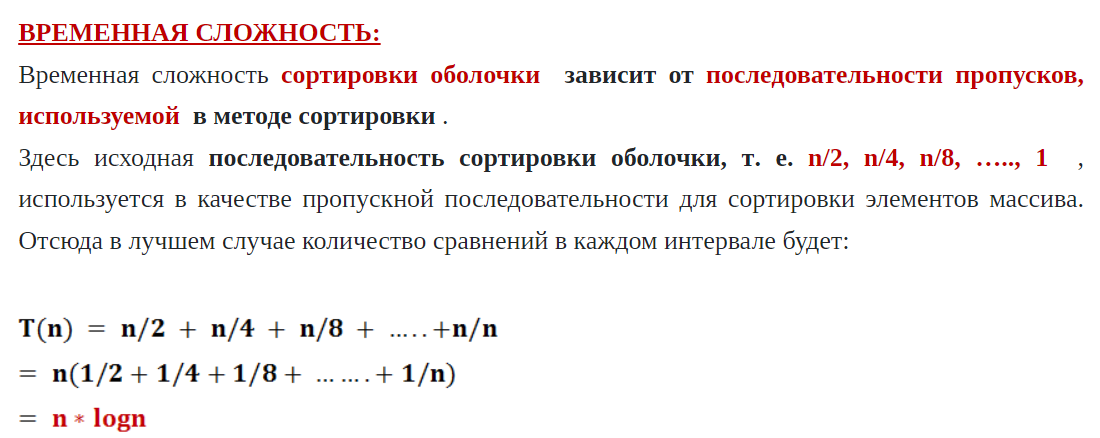
quickSort(notebooks, pivot + 1, right);

}







int step = NM / 2, tmp, j, i;

while (step > 0) {

for (i = step; i < NM; i++) {

tmp = notebooks[i].price;

for (j = i; j >= step && notebooks[j - step].price > tmp; j -= step)

notebooks[j].price = notebooks[j - step].price;

notebooks[j].price = tmp;

}

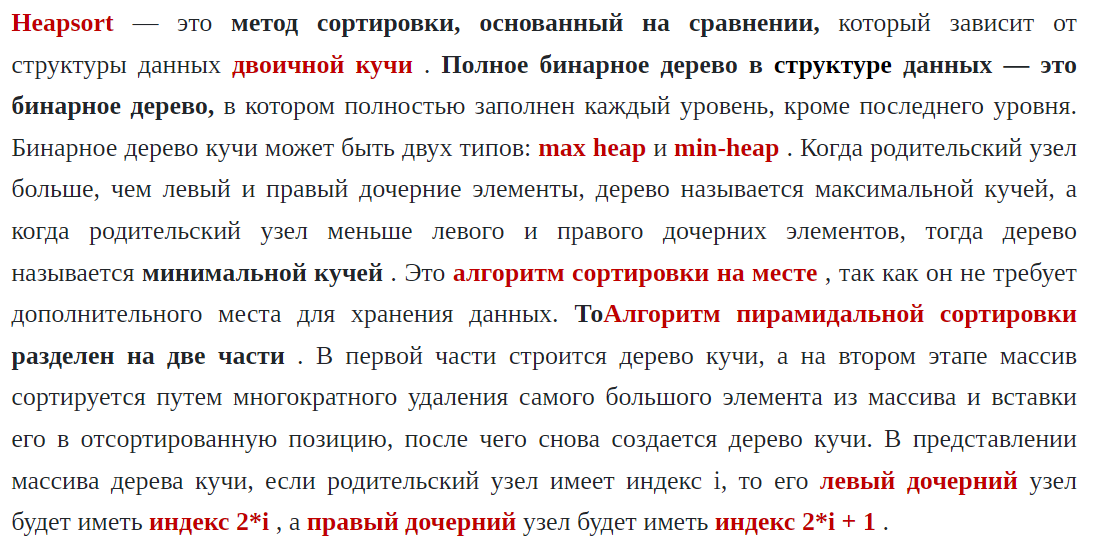
step = step / 2;

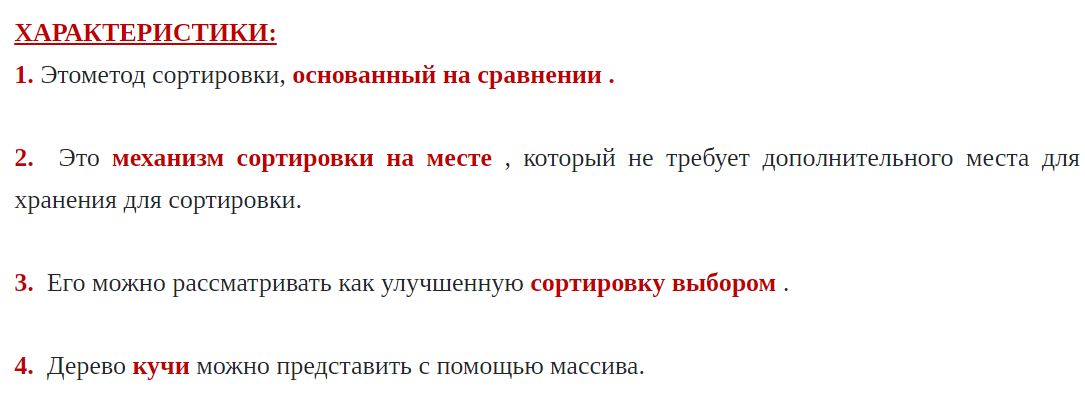
}

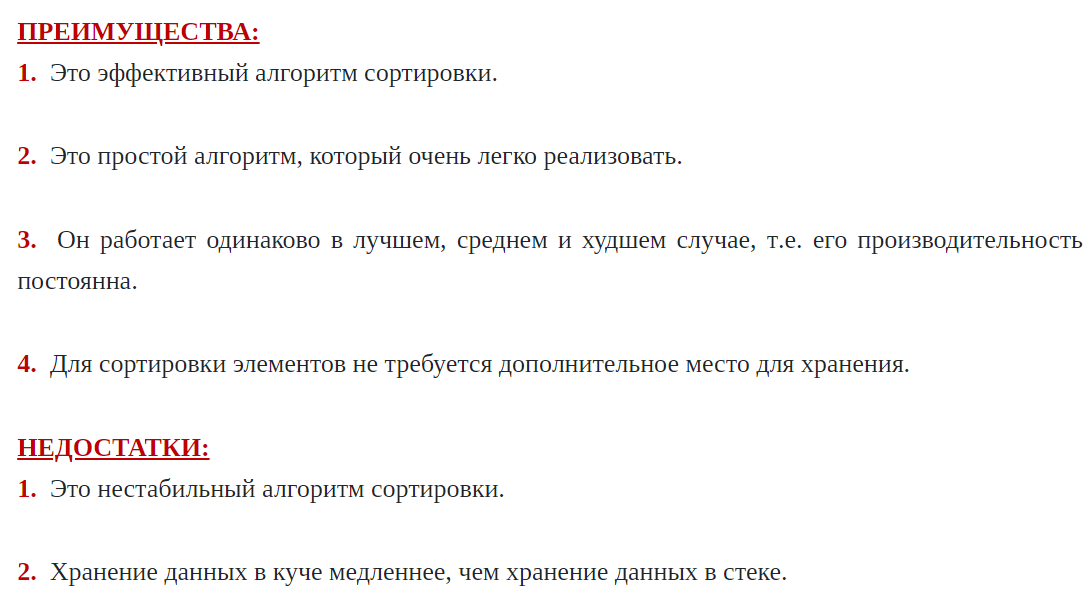
for (int i = 0; i < NM; i++) {

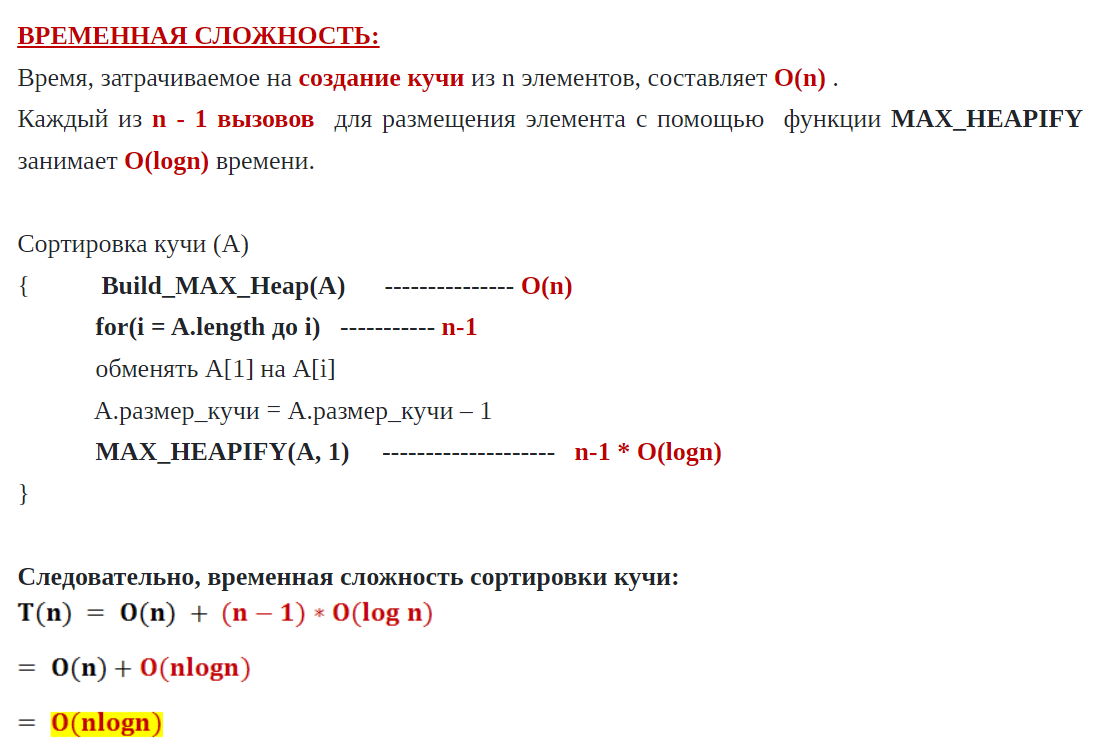
cout << notebooks[i].id << "/" << notebooks[i].name << "/" << notebooks[i].year << "/" << notebooks[i].price << "/" << notebooks[i].ram << endl;

}









void max\_heapify(NOTEBOOK\* notebooks, int n, int i) {

int par, left, right, tmp;

par = i; // самый большой элемент будет корнем

left = 2 \* i + 1; // элемент на позиции 2\*i+1 будет левым поддеревом

right = 2 \* i + 2; // элемент на позиции 2\*i+2 будет правым поддеревом

if (left < n && notebooks[left].price > notebooks[par].price) // если левое поддерево больше чем корень

par = left;

if (right < n && notebooks[right].price > notebooks[par].price) // если правое поддерево больше чем корень

par = right;

if (par != i) { // если наибольший элемент это не корень

tmp = notebooks[i].price;

notebooks[i].price = notebooks[par].price;

notebooks[par].price = tmp;

max\_heapify(notebooks, n, par); // рекурсивный вызов, чтобы снова упорядочить затронутое поддерево

}

}

void heapsort(NOTEBOOK\* notebooks, int n) {

int i, tmp;

for (i = n / 2 - 1; i >= 0; i--) // создание кучи

max\_heapify(notebooks, n, i);

for (i = n - 1; i >= 0; i--) { // выполнение heap\_sort путем извлечения элементов один за другим

// перемещение текущего корня в конец

tmp = notebooks[0].price;

notebooks[0].price = notebooks[i].price;

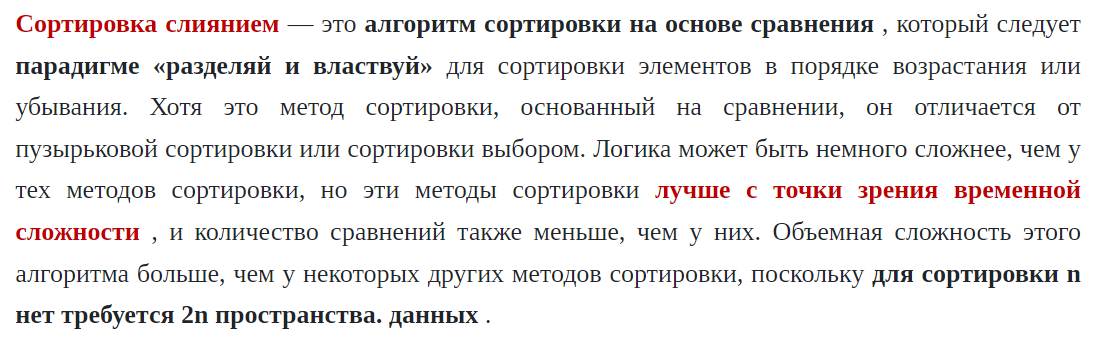
notebooks[i].price = tmp;

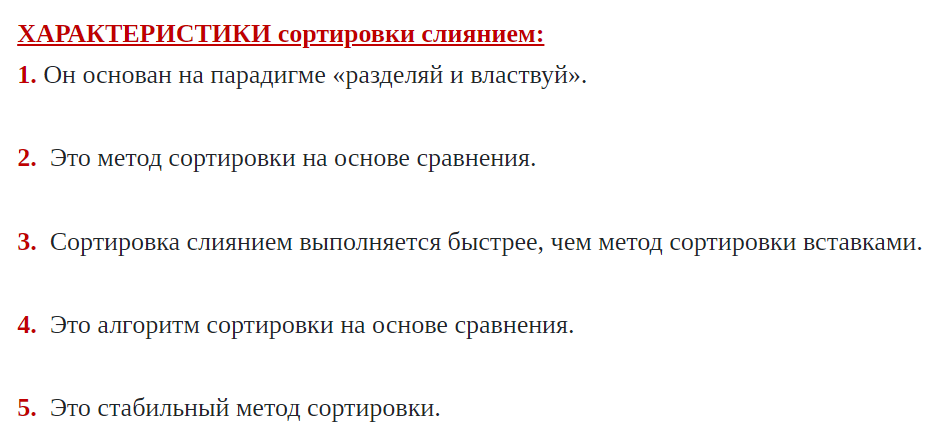
// перестановка элементов для уменьшения кучи

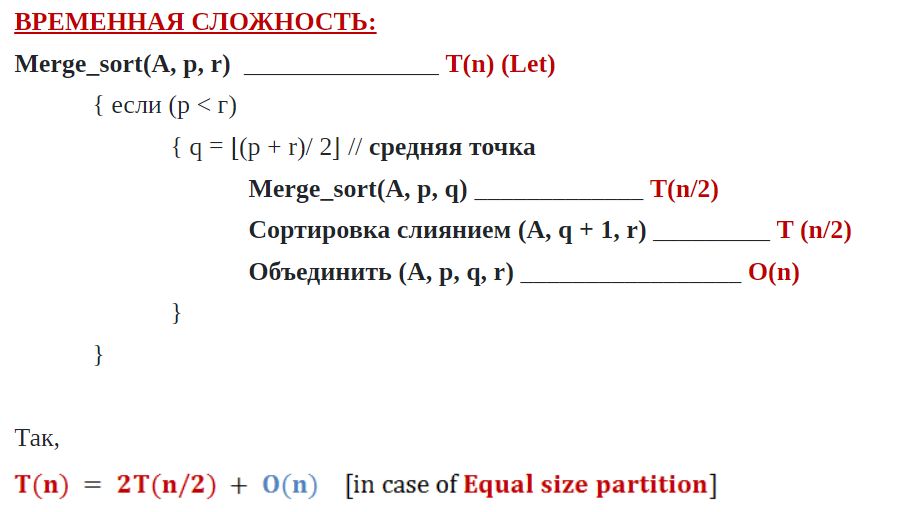
max\_heapify(notebooks, i, 0);

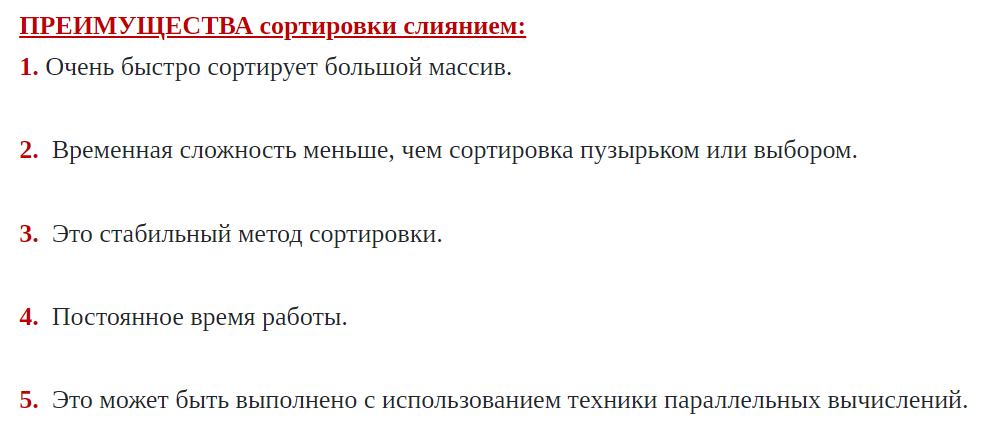
}

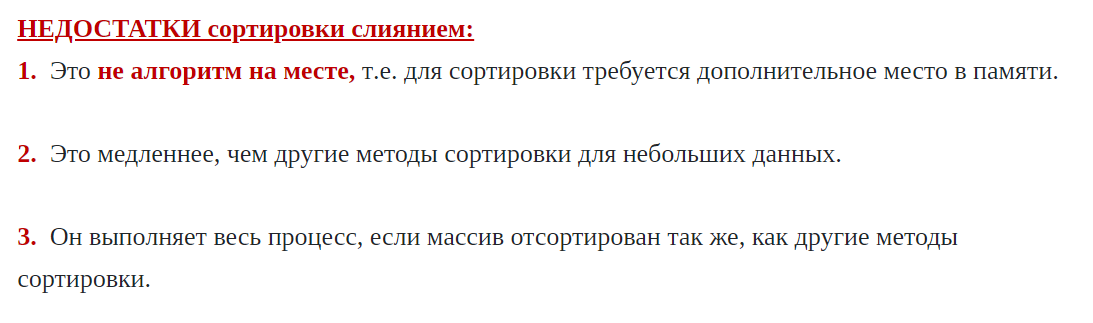
}











void merge(NOTEBOOK\* notebooks, int l, int m, int r) {

int n1, n2, i, j, k;

n1 = m - l + 1;

n2 = r - m;

//создание двух временных массивов

NOTEBOOK\* notebooks1 = new NOTEBOOK[n1];

NOTEBOOK\* notebooks2 = new NOTEBOOK[n2];

//копирование данных во временные массивы

for (i = 0; i < n1; i++)

notebooks1[i] = notebooks[l + i];

for (j = 0; j < n2; j++)

notebooks2[j] = notebooks[m + 1 + j];

// объеденение временных массивов в исходный массив

i = 0; // начальный индекс первого подмассива

j = 0; // начальный индекс второго подмассива

k = l; // начальный объединенного первого подмассива

while (i < n1 && j < n2) {

if (notebooks1[i].price <= notebooks2[j].price) {

notebooks[k] = notebooks1[i];

i++;

}

else {

notebooks[k] = notebooks2[j];

j++;

}

k++;

}

// заносим элементы левого подмассива(если есть)

while (i < n1) {

notebooks[k] = notebooks1[i];

i++;

k++;

}

// заносим элементы правого подмассива(если есть)

while (j < n2) {

notebooks[k] = notebooks2[j];

j++;

k++;

}

}

void mergesort(NOTEBOOK\* notebooks, int p, int r) {

if (p < r) {

// здесь мы вычисляем среднюю точку, из которой мы делим на два подсписка

int q = p + (r - p) / 2; // средняя точка

mergesort(notebooks, p, q);

mergesort(notebooks, q + 1, r);

// объединяем отсортированные подмассивы

merge(notebooks, p, q, r);

}

}

|  |  |
| --- | --- |
| Алгоритм | Время |
| Selection Sort | 0.206 |
| Bubble Sort | 0.308 |
| Insertion Sort | 0.232 |
| Quick Sort | 0.251 |
| Shell Sort | 0.234 |
| Heap Sort | 0.225 |
| Merge Sort | 0.204 |

Выводы: после всей проделанной работы можно сделать вывод, что мы освоили все основные алгоритмы, и теперь можем принимать их на практике