МИНЕСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧЕРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Брестский государственный технический университет»

Кафедра «Интеллектуальные информационные технологии»

Лабораторная работа №1

По дисциплине «Дискретная математика»

За 3 семестр

Тема: «Множества»

Выполнила:

студентка 2 курса

группы АС-56

Карпенко М.В.

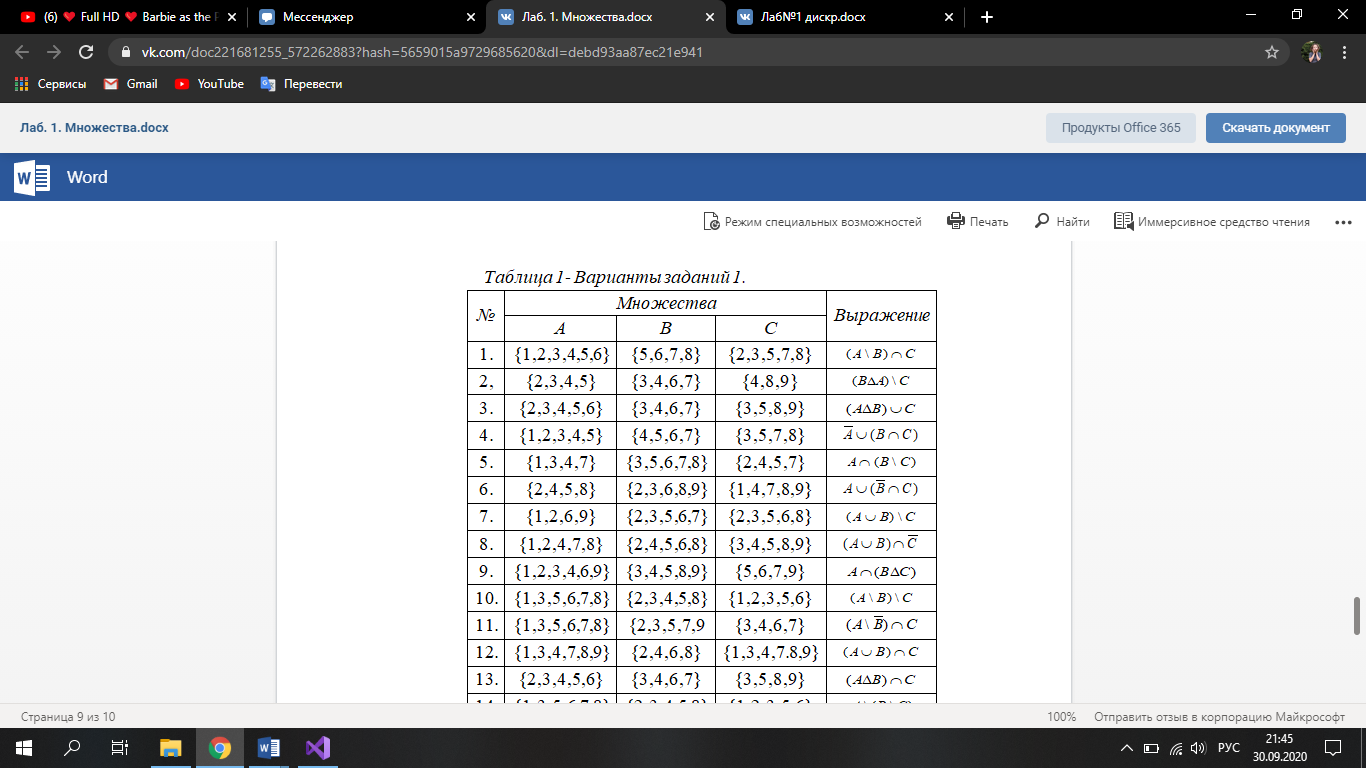
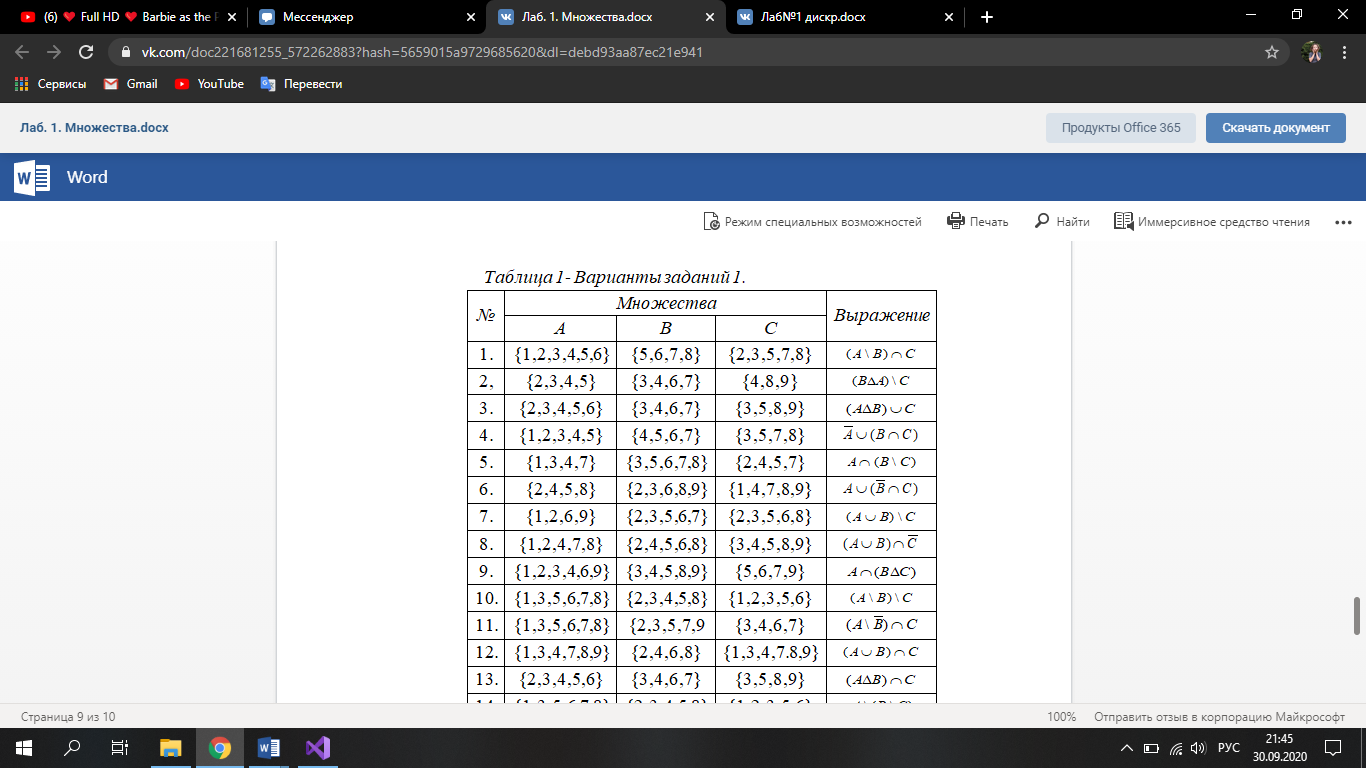
Проверил:

Глущенко Т.А.

Брест 2020

*Задание 1. (вариант 7)*

На универсуме U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11} заданы множества A, B, C. Для указанных множеств:  
1. Для заданного множества A построить булеан.  
2. На выбор реализовать сортировку слиянием либо алгоритмом «слияние» построить объединение множеств A и B.  
3. Реализовать операцию пересечения множеств, используя маску.  
4. Найти прямое произведение и его мощность.  
5. Вычислить программно выражение согласно варианту, проиллюстрировать результат диаграммой Эйлера-Венна.  
6. Указать примеры покрытий и разбиения множества A.  
8. Построить для вашего множества (или множества меньшей мощности) бинарный код Грея.  
9. Составить таблицу для различных сортировок (быстрой, слиянием, выбором, вставки, пятая на выбор) с указанием «О большого», достоинств и недостатков данного алгоритма сортировки.

*Код программы для заданий 1, 2, 3, 4, 5, 8:*

#include <iostream>

#include <vector>

#include <string>

using namespace std;

void boolmask(int U[], int A[], int B[], int C[], int a[], int b[], int c[]);

void crossing(int U[], int a[], int b[], int c[]);

void multiplication(int A[], int B[]);

void expression(int U[], int a[], int b[], int c[]);

void directProduct(int A, int B);

unsigned int grayencode(unsigned int g);

void boolean(int A[], int num, string s)

{

if (num == 4)

{

cout << "{ " << s << " }," << '\n';

return;

}

boolean(A, num + 1, s);

s = s + to\_string(A[num]) + ",";

boolean(A, num + 1, s);

}

int main()

{

setlocale(0, "");

int U[11] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11 };

int A[4] = { 1,2,6,9 };

int B[5] = { 2,3,5,6,7 };

int C[5] = { 2,3,5,6,8 };

int a[11];

int b[11];

int c[11];

cout << "Булеан множества А: ";

cout << "{\n";

boolean(A, 0, "");

cout << "}\n" << endl;

//алгоритм слияние

vector<int> mas;

int j = 0, i = 0;

while (i < 4)

{

while (j < 5)

{

if (A[i] < B[j])

{

mas.push\_back(A[i]);

i++;

}

if (A[i] == B[j])

{

mas.push\_back(A[i]);

i++;

j++;

}

if (A[i] > B[j])

{

mas.push\_back(B[j]);

j++;

}

}

if (j == 5)

{

mas.push\_back(A[i]);

i++;

}

}

cout << "Алгоритм слияние: \n";

cout << "{ ";

for (int i = 0; i < mas.size(); i++)

{

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

}

cout << "}\n" << endl;

boolmask(U, A, B, C, a, b, c);

cout << "Пересечение множеств: \n";

crossing(U, a, b, c);

cout << "Прямое произведение А\*В: \n";

multiplication(A, B);

cout << "Вычисление выражения (А+В)-С: \n";

expression(U, a, b, c);

cout << "Бинарный код Грея: \n";

for (int i = 0; i < 11; i++)

{

int temp;

temp = grayencode(U[i]);

cout << U[i] << "(x10) = ";

for (int i = 2; i > -1; i--)

{

if (temp >> i & 1)

cout << 1;

else

cout << 0;

}

cout << "(Gray)\n";

}

system("pause");

return 0;

}

void boolmask(int U[], int A[], int B[], int C[], int a[], int b[], int c[])

{

int x = 0;

for (int i = 0; i < 11; i++)

{

if (U[i] == A[x])

{

a[i] = 1;

x++;

}

else a[i] = 0;

}

x = 0;

for (int i = 0; i < 11; i++)

{

if (U[i] == B[x])

{

b[i] = 1;

x++;

}

else b[i] = 0;

}

x = 0;

for (int i = 0; i < 11; i++)

{

if (U[i] == C[x])

{

c[i] = 1;

x++;

}

else c[i] = 0;

}

};

void crossing(int U[], int a[], int b[], int c[])

{

cout << "{";

for (int i = 0; i < 11; i++)

{

if ((a[i] && b[i]) && c[i])

{

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

}

}

cout << "}\n" << endl;

};

void multiplication(int A[], int B[])

{

cout << "{";

for (int i = 0; i < 4; i++)

{

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

{

cout << "(" << A[i] << "," << B[j] << ")";

if (i != 4 && j != 4)

cout << ",";

}

cout << "\n";

}

cout << "}\n" << endl;

};

void expression(int U[], int a[], int b[], int c[])

{

cout << "{";

for (int i = 0; i < 11; i++)

{

if ((a[i] || b[i]) && !c[i])

{

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

}

}

cout << "}\n" << endl;

};

void directProduct(int\* A, int\* B)

{

for (int i = 0; i < 4; i++)

{

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

{

cout << "{" << A[i] << "," << B[j] << "}, ";

}

}

};

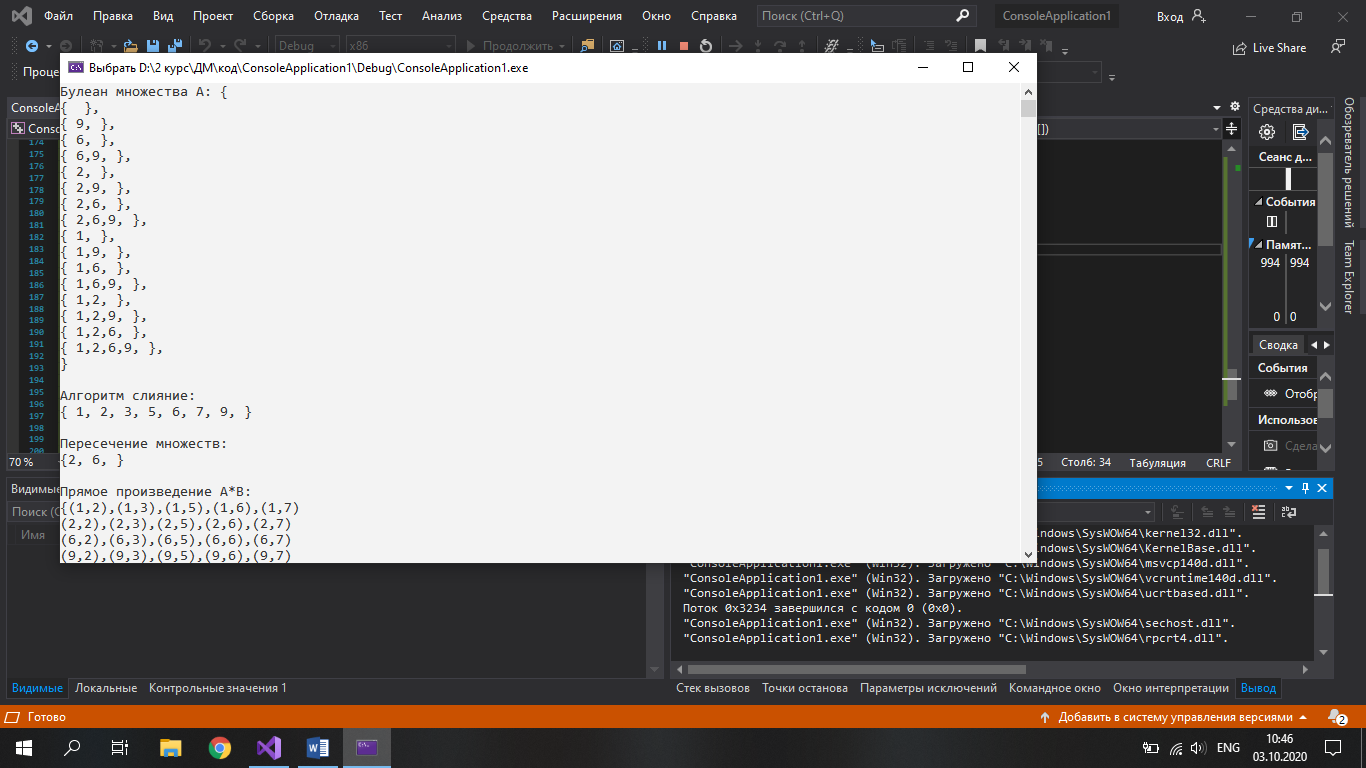
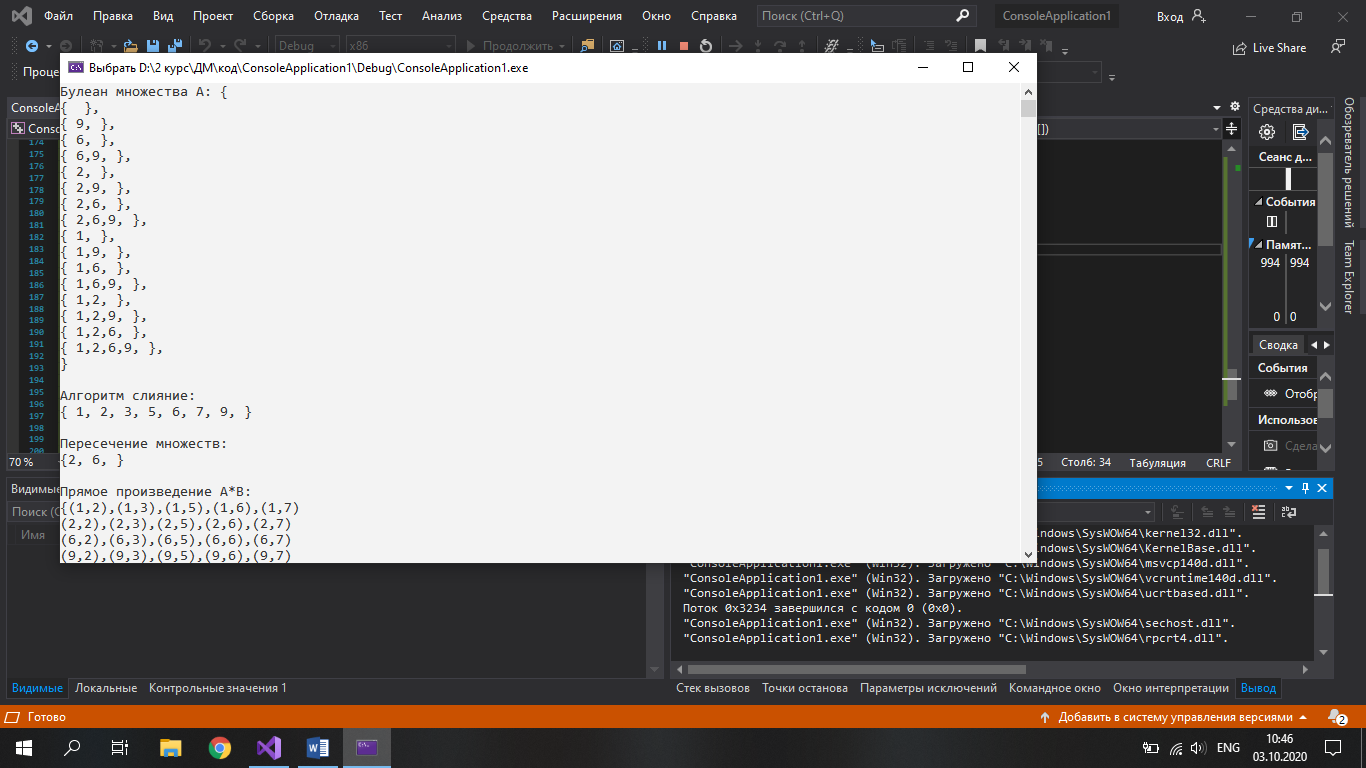
unsigned int grayencode(unsigned int g)

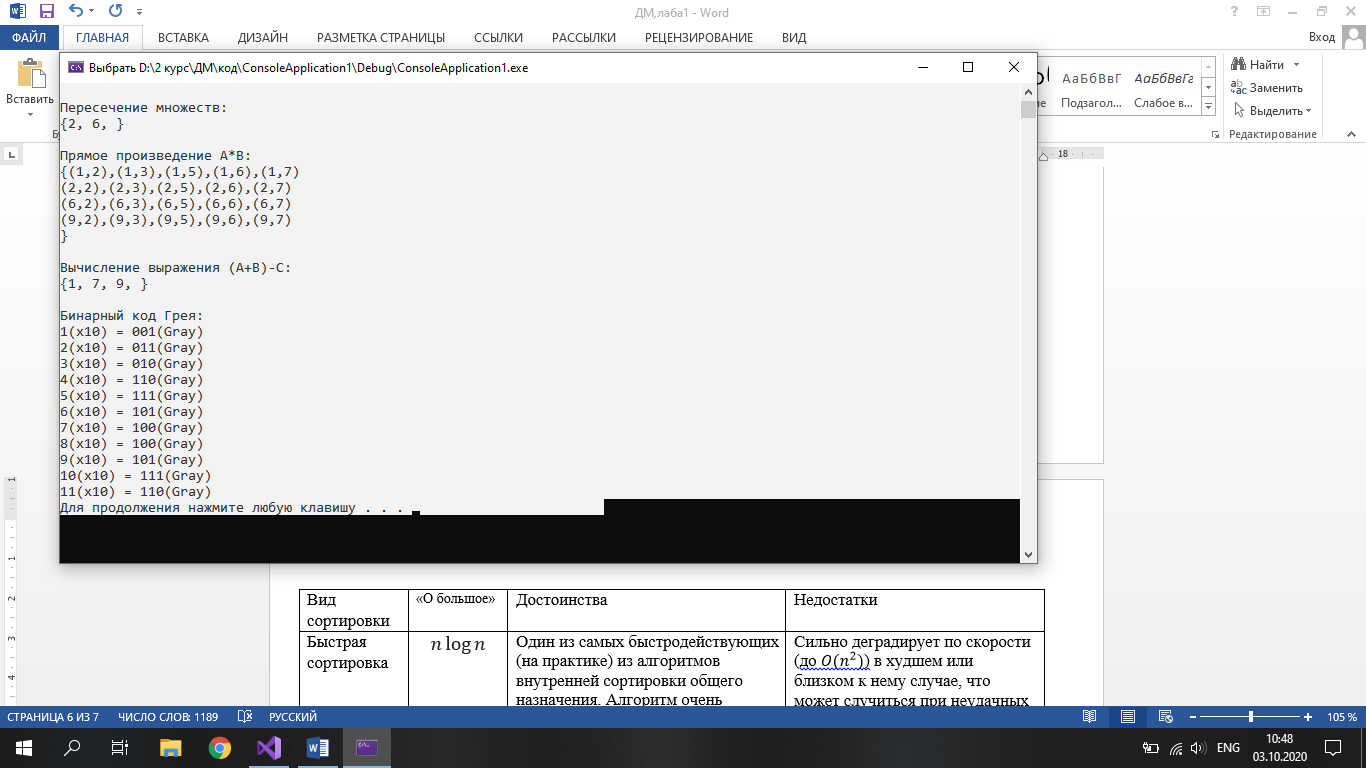
{

return g ^ (g >> 1);

}

*Вывод:*





9. Составить таблицу для различных сортировок.

|  |  |  |  |
| --- | --- | --- | --- |
| Вид сортировки | «О большое» | Достоинства | Недостатки |
| Быстрая сортировка |  | Один из самых быстродействующих (на практике) из алгоритмов внутренней сортировки общего назначения. Алгоритм очень короткий: запомнив основные моменты, его легко написать «из головы». Требует лишь {\displaystyle O(\log n)} дополнительной памяти для своей работы. Хорошо сочетается с механизмами [кэширования](https://ru.wikipedia.org/wiki/%D0%9A%D1%8D%D1%88-%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C) и [виртуальной памяти](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C). Допускает естественное распараллеливание (сортировка выделенных подмассивов в параллельно выполняющихся подпроцессах). Допускает эффективную модификацию для сортировки по нескольким ключам: благодаря тому, что в процессе разделения автоматически выделяется отрезок элементов, равных опорному, этот отрезок можно сразу же сортировать по следующему ключу. Работает на [связных списках](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA) и других структурах с последовательным доступом, допускающих эффективный проход как от начала к концу, так и от конца к началу.{\displaystyle n\log n} | Сильно деградирует по скорости (до {\displaystyle O(n^{2})}) в худшем или близком к нему случае, что может случиться при неудачных входных данных. Прямая реализация в виде функции с двумя рекурсивными вызовами может привести к ошибке [переполнения стека](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D1%82%D0%B5%D0%BA%D0%B0), так как в худшем случае ей может потребоваться сделать {\displaystyle O(n)} вложенных рекурсивных вызовов. [Неустойчив](https://ru.wikipedia.org/wiki/%D0%A3%D1%81%D1%82%D0%BE%D0%B9%D1%87%D0%B8%D0%B2%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0). |
| Сортировка слиянием |  | Работает даже на структурах данных последовательного доступа. Хорошо сочетается с [подкачкой](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB_%D0%BF%D0%BE%D0%B4%D0%BA%D0%B0%D1%87%D0%BA%D0%B8) и [кэшированием](https://ru.wikipedia.org/wiki/%D0%9A%D1%8D%D1%88-%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C) памяти. Неплохо работает в [параллельном](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) варианте: легко разбить задачи между процессорами поровну, но трудно сделать так, чтобы другие процессоры взяли на себя работу, в случае если один процессор задержится. Не имеет «трудных» входных данных. Устойчивая - сохраняет порядок равных элементов (принадлежащих одному классу эквивалентности по сравнению). | На «почти отсортированных» массивах работает столь же долго, как на хаотичных. Существует вариант сортировки слиянием, который работает быстрее на частично отсортированных данных, но он требует дополнительной памяти, в дополнении ко временному буферу, который используется непосредственно для сортировки. Требует дополнительной памяти по размеру исходного массива. |
| Сортировка выбором | {\displaystyle O(n^{2})} | Сортировка выбором проста в реализации, и в некоторых ситуациях стоит предпочесть ее наиболее сложным и совершенным методам. | Уступает в эффективности , так как затрачивается больше времени. |
| Сортировка вставкой |  | Если массив изначально частично отсортирован, то отсортированные места выполняются за константное время, то есть внутренний цикл просто не проходит. | Чем большее множество нужно отсортировать, тем большее время потребуется для выполнения сортировки. Наихудшее время выполнения данного алгоритма{\displaystyle O(n^{2})}, это может быть если входящий массив отсортирован по убыванию. |
| Сортировка пузырьком |  | Для понимания и реализации этот алгоритм — простейший. Не требуется дополнительных массивов | Эффективен он лишь для небольших массивов. Время алгоритма пропорционально квадрату количества элементов (самый медленный способ сортировки) |