БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

**Оценка сортировок массива. Тестирование кода.**

**По:** Основы алгоритмизации и программирования

**Выполнил:** Корнелюк

Валентин Владимирович

1 курс 4 группа ПОИТ

**Преподаватель:**

Белодед Николай Иванович

**г. Минск**

2023 г.

**Оглавление**

[Сортировка пузырьком 3](#_Toc135306388)

[Сортировка вставками 6](#_Toc135306389)

[Сортировка выбором 8](#_Toc135306390)

[Сортировка слиянием 10](#_Toc135306391)

[Пирамидальная сортировка 12](#_Toc135306392)

[Быстрая сортировка(сортировка Хоара) 14](#_Toc135306393)

[Сравнение скорости сортировки разных алгоритмов 16](#_Toc135306394)

[Вывод 17](#_Toc135306395)

# **Сортировка пузырьком**

Сортировка пузырьком (Bubble Sort) - это простой алгоритм сортировки, который последовательно проходит по массиву и сравнивает пары соседних элементов. Если элементы находятся в неправильном порядке (например, больший элемент стоит перед меньшим), то они меняются местами. Этот процесс продолжается до тех пор, пока все элементы не будут упорядочены.

Сложность сортировки пузырьком в худшем и среднем случаях составляет O(n^2), где n - количество элементов в массиве.

Код функций, реализующих сортировку (в дальнейшем, для уменьшения объема текста, будет представлена перегрузка функций только для типа int):

void sortBubble(int\* arr, int N)

{

int buff = 0;

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

{

for (int j = N - 1; j > i; j--)

{

if (arr[j] < arr[j - 1])

{

buff = arr[j - 1];

arr[j - 1] = arr[j];

arr[j] = buff;

}

}

}

}

void sortBubble(double\* arr, int N)

{

double buff = 0;

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

{

for (int j = N - 1; j > i; j--)

{

if (arr[j] < arr[j - 1])

{

buff = arr[j - 1];

arr[j - 1] = arr[j];

arr[j] = buff;

}

}

}

}

void sortBubble(string\* arr, int N)

{

string temp;

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

{

for (int j = N - 1; j > i; j--)

{

if (arr[j] < arr[j - 1]) // Сравниваем строки в лексикографическом порядке

{

temp = arr[j - 1];

arr[j - 1] = arr[j];

arr[j] = temp;

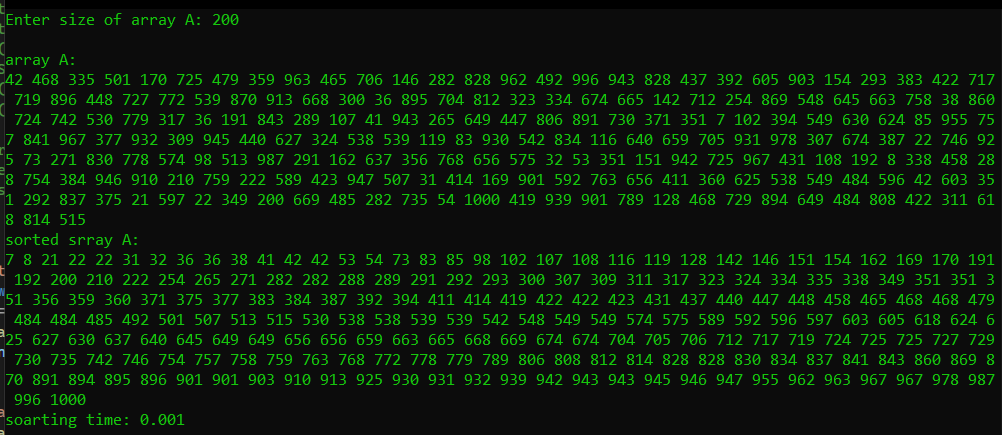
}

}

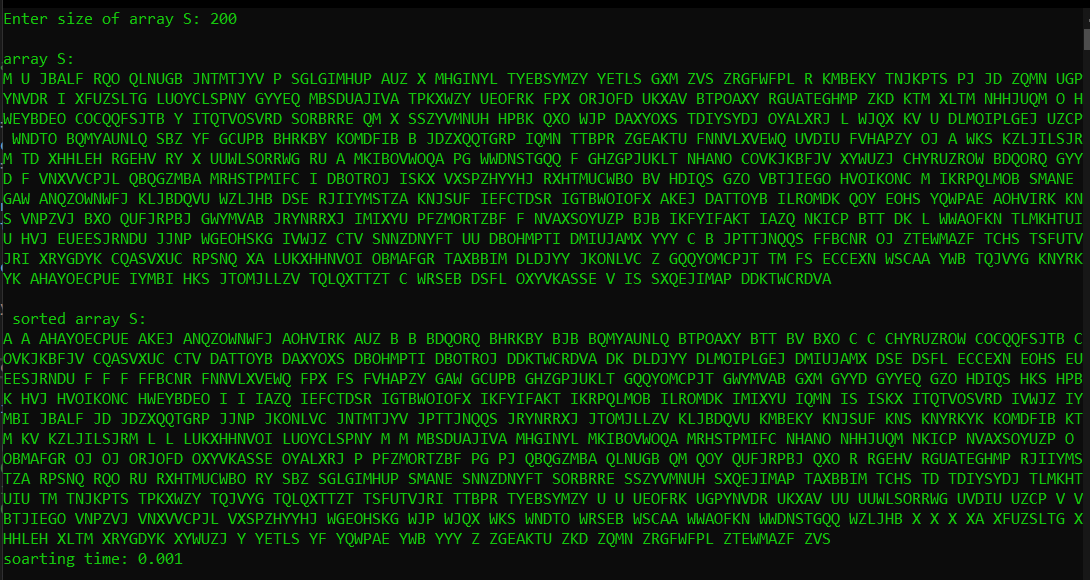
}

}

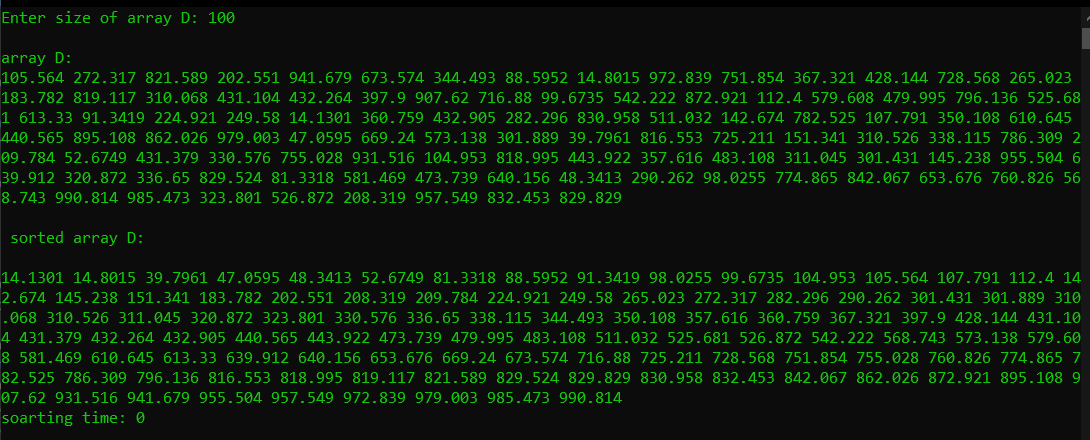
Демонстрация корректной работы алгоритма:



Результат сортировки случайно сгенерированного массива типа int



Результат сортировки случайно сгенерированного массива типа string

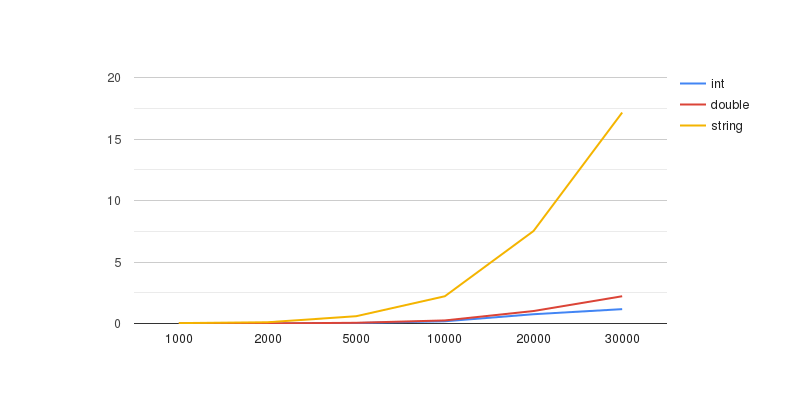


Результат сортировки случайно сгенерированного массива типа double

В результате исследования были получены следующие значения(время указано в секундах):

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во элементов\тип | int | double | string |
| 1000 | |  | | --- | | 0.001 | | |  | | --- | | 0.003 | | |  | | --- | | 0.032 | |
| 2000 | 0.005 | 0.008 | 0.102 |
| 5000 | 0.038 | 0.057 | 0.593 |
| 10000 | 0.18 | 0.25 | 2.215 |
| 20000 | 0.754 | 1.016 | 7.518 |
| 30000 | 1.617 | 2.216 | 17.151 |
| 50000 | 4.446 | 7.190 | Очень долго |

Пердставленные данные на графике:



# **Сортировка вставками**

Сортировка вставками (Insertion Sort) - это простой алгоритм сортировки, который работает путем построения отсортированной последовательности по одному элементу за один проход по массиву. На каждом шаге текущий элемент вставляется на правильное место в уже отсортированной части массива.

Сложность сортировки вставками (Insertion Sort) в худшем, среднем и лучшем случаях составляет O(n^2), где n - количество элементов в массиве.

Код функции, реализующей сортировку:

void sortInsert(int\* arr, int N)

{

int num;

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

for (int j = i; j > 0 && arr[j - 1] > arr[j]; j--) {

num = arr[j - 1];

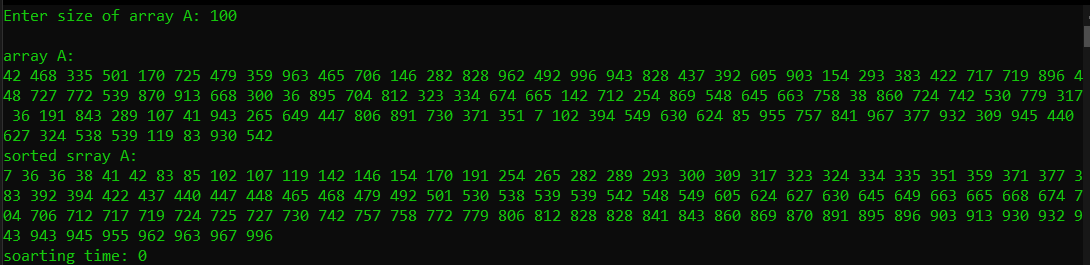
arr[j - 1] = arr[j];

arr[j] = num;

}

}

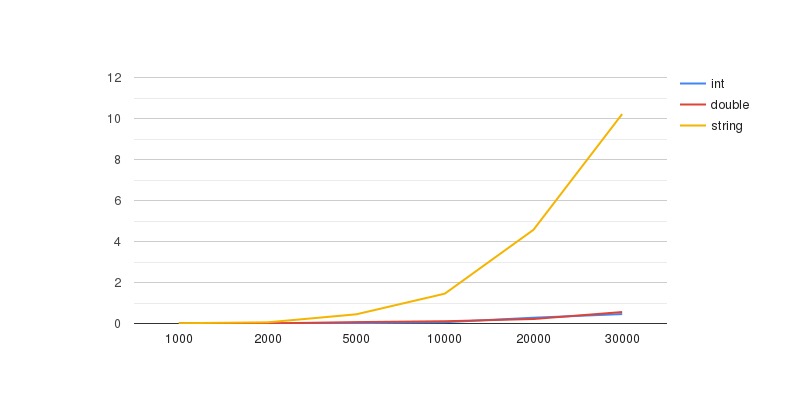
Демонстрация корректной работы алгоритма:



В результате исследования были получены следующие значения:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во элементов\тип | int | double | string |
| 1000 | |  |  | | --- | --- | | |  | | --- | | 0.001 | | | |  |  | | --- | --- | | |  | | --- | | 0.001 | | | |  |  | | --- | --- | | |  | | --- | | 0.013 | | |
| 2000 | 0.003 | 0.003 | 0.058 |
| 5000 | 0.016 | 0.068 | 0.448 |
| 10000 | 0.063 | 0.11 | 1.457 |
| 20000 | 0.282 | 0.22 | 4.573 |
| 30000 | 0.462 | 0.56 | 10.218 |
| 50000 | 1.354 | 1.205 | Очень долго |

Пердставленные данные на графике:

****

# **Сортировка выбором**

Сортировка выбором (Selection Sort) - это простой алгоритм сортировки, который на каждом шаге находит минимальный (или максимальный) элемент из оставшихся неотсортированных элементов и помещает его в конец (или начало) отсортированной части массива. Таким образом, на каждом шаге увеличивается длина отсортированной части массива, а длина неотсортированной части уменьшается.

Сложность сортировки выбором (Selection Sort) в худшем, среднем и лучшем случаях составляет O(n^2), где n - количество элементов в массиве.

Код функции, реализующей сортировку:

void sortSelection(int\* arr, int N)

{

int min = 0;

int buf = 0;

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

{

min = i;

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

min = (arr[j] < arr[min]) ? j : min;

if (i != min)

{

buf = arr[i];

arr[i] = arr[min];

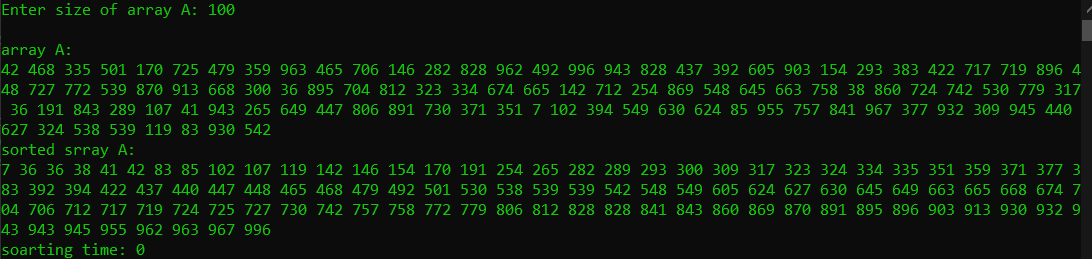
arr[min] = buf;

}

}

}

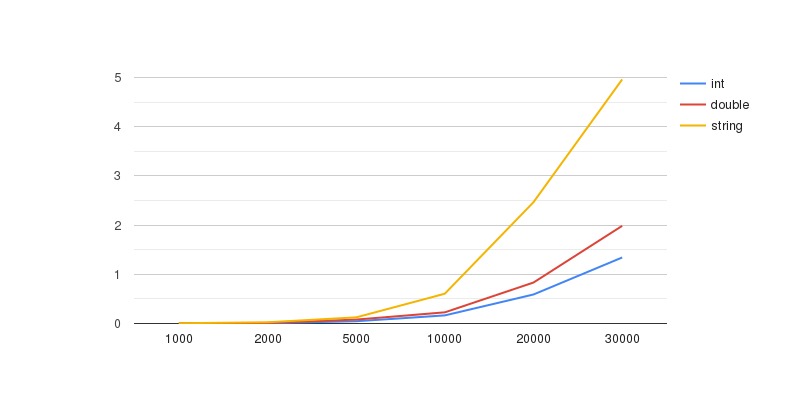
Демонстрация корректной работы алгоритма:



В результате исследования были получены следующие значения:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во элементов\тип | int | double | string |
| 1000 | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0.001 | | | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0.002 | | | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0.005 | | | |
| 2000 | 0.007 | 0.011 | 0.025 |
| 5000 | 0.046 | 0.08 | 0.125 |
| 10000 | 0.165 | 0.227 | 0.606 |
| 20000 | 0.59 | 0.833 | 2.466 |
| 30000 | 1.342 | 1.985 | 4.961 |
| 50000 | 3.558 | 4.835 | Очень долго |

Пердставленные данные на графике:

****

# **Сортировка слиянием**

Сортировка слиянием (Merge Sort) - это эффективный алгоритм сортировки, основанный на принципе "разделяй и властвуй". Алгоритм разделяет исходный массив на две равные (или примерно равные) части, рекурсивно сортирует каждую из них, а затем объединяет их в один отсортированный массив.

Сложность сортировки слиянием (Merge Sort) составляет O(n log n), где n - количество элементов в массиве.

Код функции, реализующей сортировку:

void merge(int\* arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int\* L = new int[n1];

int\* R = new int[n2];

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

L[i] = arr[left + i];

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

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

int i = 0;

int j = 0;

int k = left;

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

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

delete[] L;

delete[] R;

}

void mergeSort(int\* arr, int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

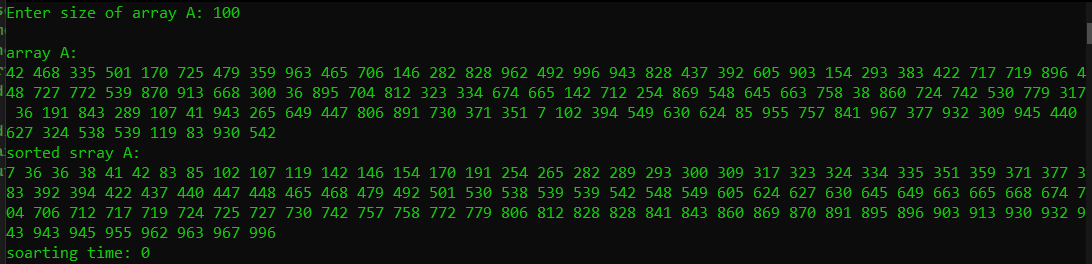
mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

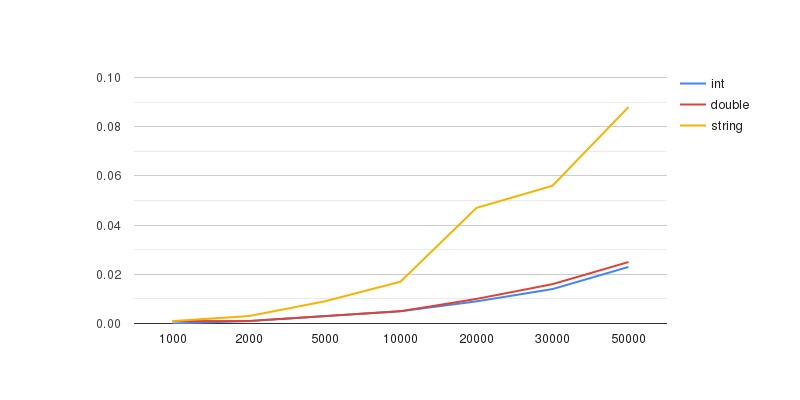
Демонстрация корректной работы алгоритма:



В результате исследования были получены следующие значения:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во элементов\тип | int | double | string |
| 1000 | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0 | | | | |  |  |  |  | | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0.001 | | | | | |  |  |  |  | | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0.001 | | | | |
| 2000 | 0.001 | 0.001 | 0.003 |
| 5000 | 0.003 | 0.003 | 0.009 |
| 10000 | 0.005 | 0.005 | 0.017 |
| 20000 | 0.009 | 0.01 | 0.047 |
| 30000 | 0.014 | 0.016 | 0.056 |
| 50000 | 0.023 | 0.025 | 0.088 |
| 1000000 | 0.362 | 0.382 | 23.97 |

Пердставленные данные на графике:



# **Пирамидальная сортировка**

Пирамидальная сортировка (Heap Sort) является алгоритмом сортировки, основанным на структуре данных под названием "пирамида" или "куча". Пирамида представляет собой специальное двоичное дерево, в котором значение каждого узла больше (или меньше) значений его потомков.

Процесс сортировки начинается с построения пирамиды из исходного массива. Затем наибольший (или наименьший) элемент, находящийся в корне пирамиды, помещается в конец массива. После этого происходит восстановление свойств пирамиды и повторное перемещение наибольшего (или наименьшего) элемента в конец массива. Этот процесс повторяется до тех пор, пока все элементы не будут отсортированы.

Сложность пирамидальной сортировки составляет O(n log n), где n - количество элементов в массиве.

Код функции, реализующей сортировку:

void heapify(int arr[], int n, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i) {

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

heapify(arr, n, largest);

}

}

void heapSort(int arr[], int n)

{

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

heapify(arr, n, i);

for (int i = n - 1; i >= 0; i--) {

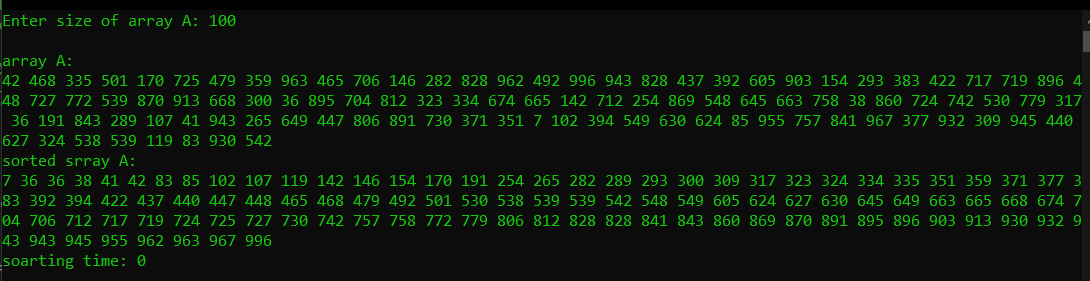
swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

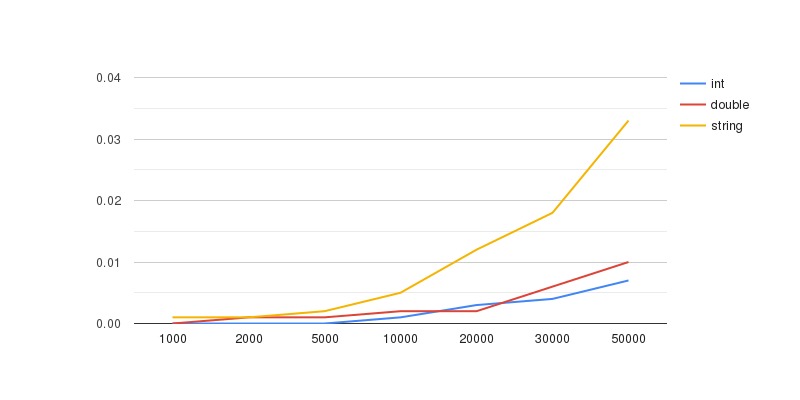
}

Демонстрация корректной работы алгоритма:



В результате исследования были получены следующие значения:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во элементов\тип | int | double | string |
| 1000 | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0 | | | | |  |  |  |  | | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0 | | | | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0.001 | | | | | |
| 2000 | 0 | 0.001 | 0.001 |
| 5000 | 0 | 0.001 | 0.002 |
| 10000 | 0.001 | 0.002 | 0.005 |
| 20000 | 0.003 | 0.002 | 0.012 |
| 30000 | 0.004 | 0.006 | 0.018 |
| 50000 | 0.007 | 0.01 | 0.033 |
| 1000000 | 0.225 | 0.269 | 20.269 |

Пердставленные данные на графике:

# **Быстрая сортировка(сортировка Хоара)**

Быстрая сортировка (Quicksort) - это алгоритм сортировки, основанный на принципе "разделяй и властвуй". Он относится к семейству сортировок сравнениями.

Процесс быстрой сортировки начинается с выбора опорного элемента из массива. Обычно в качестве опорного элемента выбирается средний элемент, но можно использовать и другие стратегии выбора. Затем массив разделяется на две подгруппы: элементы, меньшие или равные опорному, и элементы, большие опорного. После разделения опорный элемент занимает свое окончательное положение в массиве.Далее процесс повторяется для каждой из подгрупп рекурсивно. Это означает, что массивы подгрупп снова разделяются и сортируются отдельно.

Разделение и сортировка продолжаются до тех пор, пока все подгруппы не станут достаточно маленькими (обычно размером 1 или 2 элемента). Затем массив считается отсортированным.

Сложность быстрой сортировки в среднем случае составляет O(n log n), где n - количество элементов в массиве.

Код функции, реализующей сортировку:

void sortQuick(int\* arr, int first, int last)

{

int mid, count;

int f = first, l = last;

mid = arr[(f + l) / 2];

do

{

while (arr[f] < mid)

f++;

while (arr[l] > mid)

l--;

if (f <= l)

{

count = arr[f];

arr[f] = arr[l];

arr[l] = count;

f++;

l--;

}

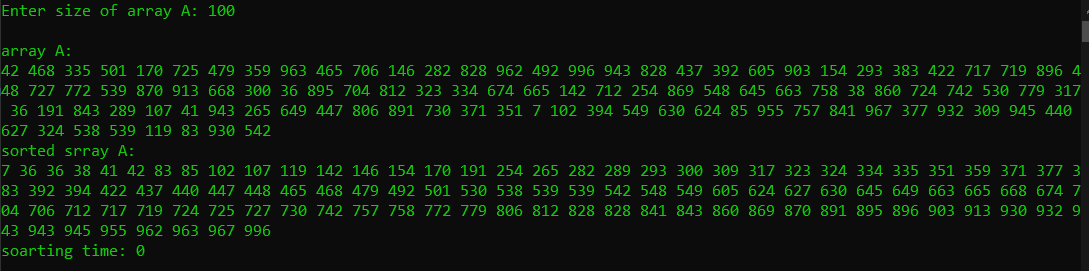
} while (f < l);

if (first < l) sortQuick(arr, first, l);

if (f < last) sortQuick(arr, f, last);

}

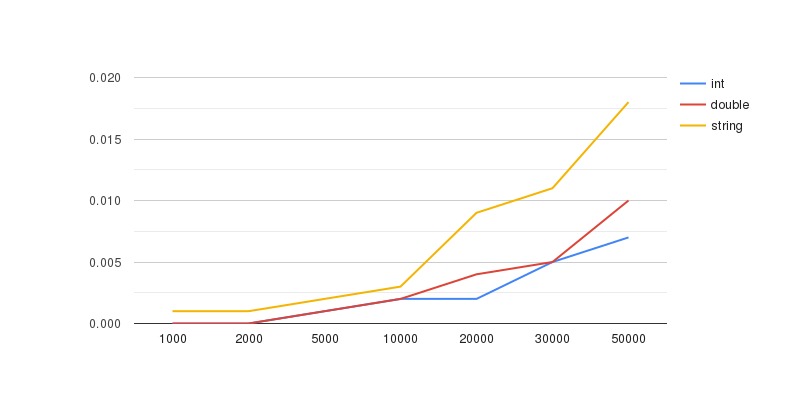
Демонстрация корректной работы алгоритма:



В результате исследования были получены следующие значения:

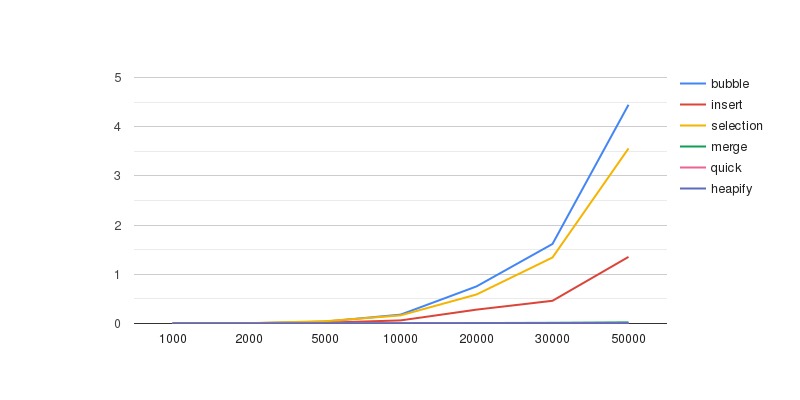
|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во элементов\тип | int | double | string |
| 1000 | |  |  |  |  | | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0 | | | | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0 | | | | | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0.001 | | | | | | |
| 2000 | 0 | 0 | 0.001 |
| 5000 | 0.001 | 0.001 | 0.002 |
| 10000 | 0.002 | 0.002 | 0.003 |
| 20000 | 0.002 | 0.004 | 0.009 |
| 30000 | 0.005 | 0.005 | 0.011 |
| 50000 | 0.007 | 0.01 | 0.018 |
| 1000000 | 0.174 | 0.265 | 5.470 |

Пердставленные данные на графике:



# **Сравнение скорости сортировки разных алгоритмов**

Если представить на графике результат времени работы разных алгоритмов сортировок, то получим следующий график (для типа int):



Таким образом можно наглядно видеть какие алгоритмы работают быстрее при различных размерах массива.

# **Вывод**

Сортировка - это важная операция в области алгоритмов и программирования, которая позволяет упорядочить элементы в массиве или коллекции. В ходе исследования различных алгоритмов сортировки мы обнаружили, что каждый алгоритм имеет свои сильные и слабые стороны, а также различную временную сложность.

Сортировка пузырьком, сортировка вставками и сортировка выбором являются простыми алгоритмами сортировки и подходят для небольших массивов или случаев, когда элементы уже частично упорядочены. Однако их временная сложность в худшем случае составляет O(n^2), что делает их неэффективными для больших массивов.

Сортировка слиянием и быстрая сортировка являются более эффективными алгоритмами сортировки. Сортировка слиянием имеет временную сложность O(n log n) во всех случаях, что делает ее хорошим выбором для обработки больших массивов. Быстрая сортировка также имеет временную сложность O(n log n) в среднем случае и обычно обладает лучшей производительностью по сравнению со слиянием на практике, но может иметь худший случай O(n^2), если опорный элемент выбирается неоптимально.

Окончательный выбор алгоритма сортировки зависит от конкретной задачи и требований к производительности. При работе с небольшими массивами или уже отсортированными данными, простые алгоритмы могут быть удобными и достаточно эффективными. В случае больших массивов или высоких требований к производительности, более сложные алгоритмы, такие как слияние или быстрая сортировка, обычно являются предпочтительными.

Дополнительно, стоит отметить, что сортировка массивов типа string может занимать больше времени по сравнению с массивами числовых типов данных, таких как int или double.

Причина этого заключается в том, что для сравнения строк необходимо выполнить дополнительные операции сравнения символов, что может занимать больше времени, особенно если строки имеют большую длину. В отличие от числовых значений, которые можно сравнивать непосредственно, строки требуют сравнения символ за символом, начиная с первого символа и продвигаясь далее до тех пор, пока не будет найдено отличие или достигнут конец строки.

Поэтому, при выборе алгоритма сортировки для массивов типа string, необходимо учитывать, что время выполнения может быть больше по сравнению с числовыми типами данных. При работе с большими массивами строк или в случае, когда производительность является критически важной, рекомендуется использовать оптимизированные алгоритмы сортировки или алгоритмы, специально разработанные для работы со строками.