Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Анализ и сравнение сортировок»

 Выполнила:

Студентка1 курса 6 группы

Литвинчук Дарья Валерьевна

Преподаватель: доц Белодед Н.И.

2024, Минск

**Содержание**

[Сортировка пузырьком 3](#_Toc166155425)

[Сортировка вставками 5](#_Toc166155426)

[Сортировка выбором 6](#_Toc166155427)

[Сортировка слиянием 8](#_Toc166155428)

[Быстрая сортировка 10](#_Toc166155429)

[Вывод 12](#_Toc166155430)

# **Сортировка пузырьком**

Сортировка пузырьком (Bubble Sort) — один из самых известных алгоритмов сортировки. Здесь нужно последовательно сравнивать значения соседних элементов и менять числа местами, если предыдущее оказывается больше последующего. Таким образом элементы с большими значениями оказываются в конце списка, а с меньшими остаются в начале. Этот процесс продолжается до тех пор, пока все элементы не будут упорядочены.

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

(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];

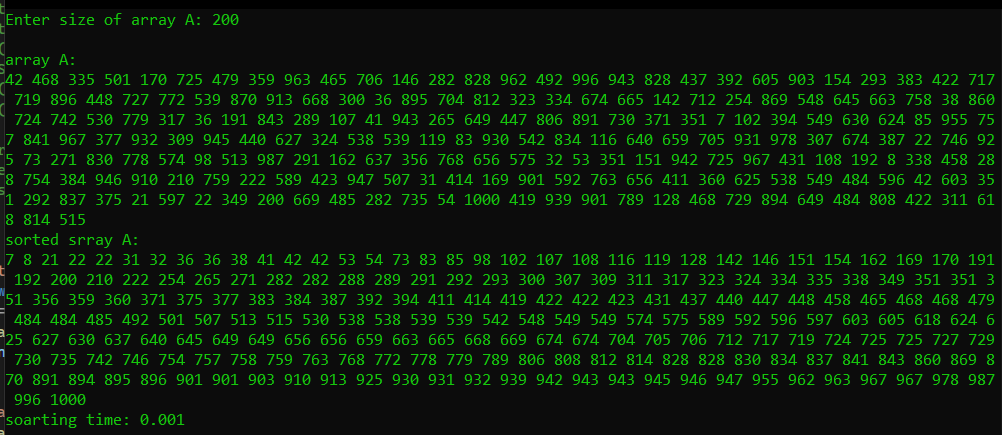
arr[j] = buff;

}

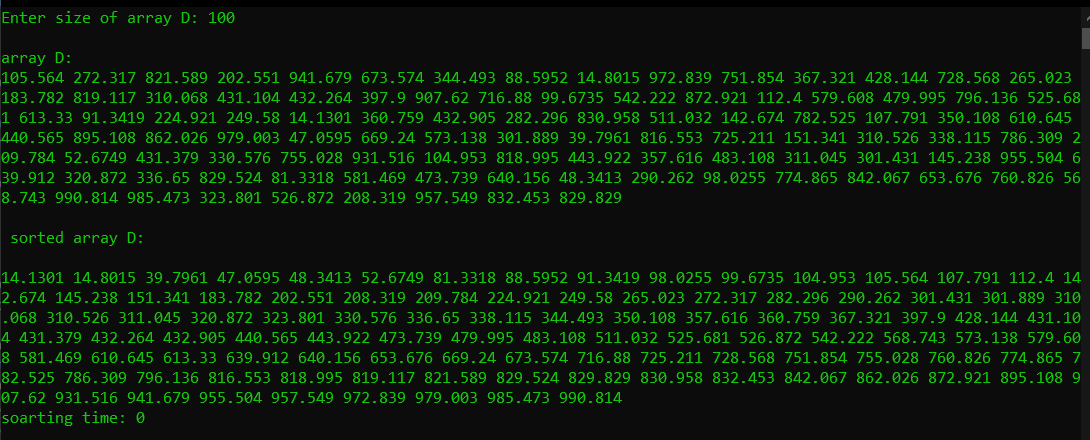
}

}

}



Результат сортировки случайно сгенерированного массива типа int



Результат сортировки случайно сгенерированного массива типа double

Результат исследования (время указано в секундах):

|  |  |  |
| --- | --- | --- |
| Кол-во эл-ов\тип | Int | double |
| 1000 | |  | | --- | | 0.001 | | |  | | --- | | 0.003 | |
| 2000 | 0.005 | 0.008 |
| 5000 | 0.040 | 0.057 |
| 10000 | 0.18 | 0.25 |
| 20000 | 0.749 | 1.016 |
| 30000 | 1.618 | 2.216 |
| 50000 | 4.446 | 7.190 |

# **Сортировка вставками**

Сортировка вставками (Insertion Sort) — это алгоритм сортировки, на каждом шаге которого массив постепенно перебирается слева направо. При этом каждый последующий элемент размещается так, чтобы он оказался между ближайшими элементами с минимальным и максимальным значением.

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

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;

}

}

void sortInsert(double\* arr, int N){

double num;

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

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

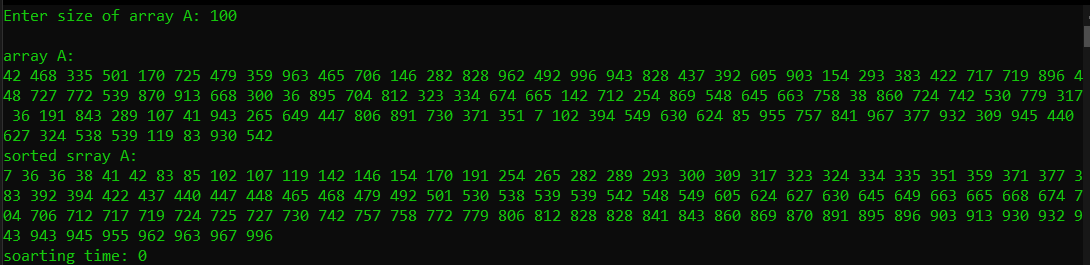
num = arr[j - 1];

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

arr[j] = num;

}

}



Результат сортировки случайно сгенерированного массива типа int

Результат исследования:

|  |  |  |
| --- | --- | --- |
| Кол-во эл-ов\тип | int | double |
| 1000 | |  |  | | --- | --- | | |  | | --- | | 0.001 | | | |  |  | | --- | --- | | |  | | --- | | 0.001 | | |
| 2000 | 0.003 | 0.003 |
| 5000 | 0.016 | 0.068 |
| 10000 | 0.063 | 0.11 |
| 20000 | 0.282 | 0.22 |
| 30000 | 0.462 | 0.56 |
| 50000 | 1.354 | 1.205 |

# **Сортировка выбором**

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

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;

}

}

}

void sortSelection(double\* arr, int N){

int min = 0;

double 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];

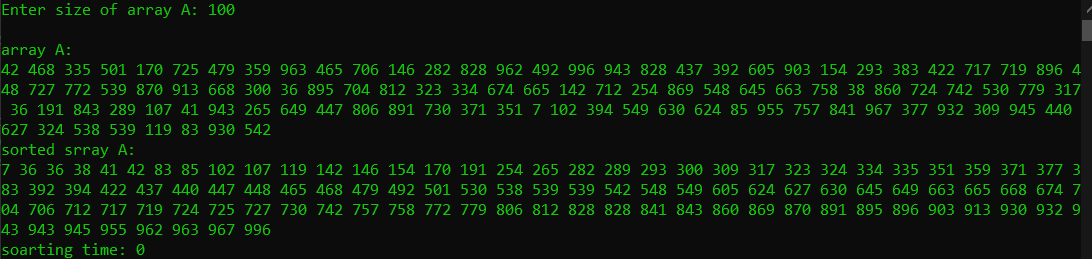
arr[i] = arr[min];

arr[min] = buf;

}

}

}



Результат сортировки случайно сгенерированного массива типа int

Результат исследования:

|  |  |  |
| --- | --- | --- |
| Кол-во эл-ов\тип | int | double |
| 1000 | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0.001 | | | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0.002 | | | |
| 2000 | 0.006 | 0.011 |
| 5000 | 0.046 | 0.08 |
| 10000 | 0.165 | 0.227 |
| 20000 | 0.60 | 0.833 |
| 30000 | 1.342 | 1.985 |
| 50000 | 3.558 | 4.835 |

# **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.

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;

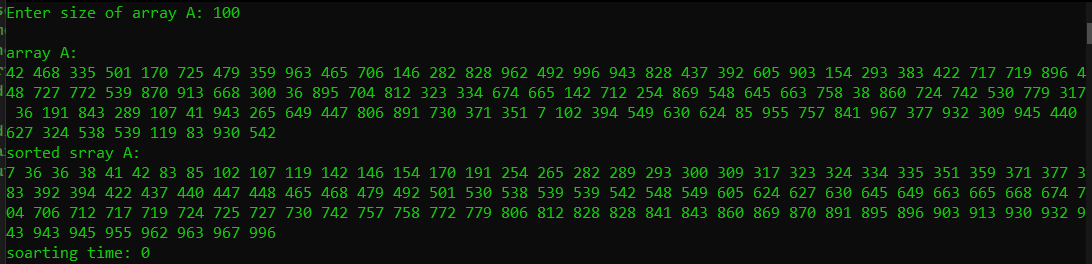
mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}



Результат сортировки случайно сгенерированного массива типа int

Результат исследования:

|  |  |  |
| --- | --- | --- |
| Кол-во эл-ов\тип | int | double |
| 1000 | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0 | | | | |  |  |  |  | | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0.001 | | | | |
| 2000 | 0.001 | 0.001 |
| 5000 | 0.003 | 0.003 |
| 10000 | 0.005 | 0.005 |
| 20000 | 0.009 | 0.01 |
| 30000 | 0.014 | 0.016 |
| 50000 | 0.023 | 0.024 |
| 1000000 | 0.358 | 0.382 |

# **Быстрая сортировка**

Быстрая сортировка (Quicksort) - это алгоритм сортировки, основанный на принципе "разделяй и властвуй". 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.

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);

}

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

double 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++;

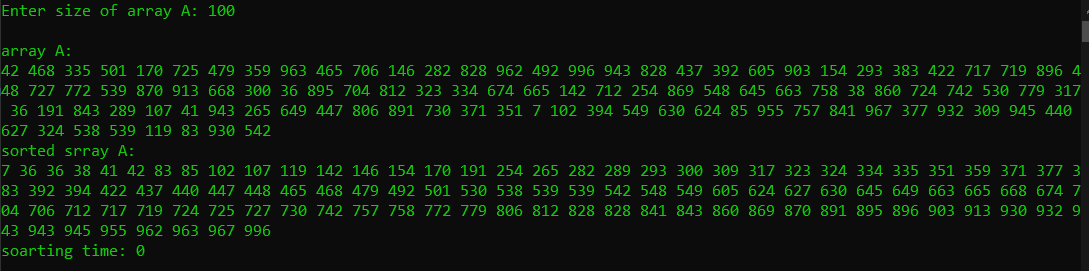
l--;}

} while (f < l);

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

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

}



Результат сортировки случайно сгенерированного массива типа int

Результат исследования:

|  |  |  |
| --- | --- | --- |
| Кол-во эл-ов\тип | int | double |
| 1000 | |  |  |  |  | | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0 | | | | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | | 0 | | | | | |
| 2000 | 0 | 0 |
| 5000 | 0.001 | 0.001 |
| 10000 | 0.002 | 0.002 |
| 20000 | 0.002 | 0.004 |
| 30000 | 0.005 | 0.005 |
| 50000 | 0.007 | 0.01 |
| 1000000 | 0.172 | 0.263 |

# **Вывод**

Сортировка является важной операцией в программировании и алгоритмах, которая позволяет упорядочить элементы в массиве. Различные алгоритмы сортировки имеют свои преимущества и недостатки, 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 не оптимально.

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.