Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе №3

по курсу «Анализ Алгоритмов»

на тему: «Алгоритмы сортировки»

Студент группы ИУ7-51Б _____ (Подпись, дата) Шубенина Д. В. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Строганов Ю. В. (Фамилия И.О.)

Москва — 2023 г.

Содержание

Введение

Целью данной лабораторной работы является изучение, реализация и исследование алгоритмов сортировки.

Необходимо выполнить следующие задачи:

1) ознакомиться со следующими алгоритмами сортировки:
   - блинная;
   - быстрая;
   - гномья;
2) реализовать данные алгоритмы;
3) оценить трудоемкости реализуемых алгоритмов;
4) выполнить сравнительный анализ алгоритмов по затрачиваемым ресурсам (времени, памяти);
5) описать и обосновать полученные результаты в отчете.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы блинной, быстрой и гномьей сортировки.

1.1 Блинная сортировка

Единственная операция, допустимая в алгоритме — переворот элементов последовательности до какого-либо индекса. В отличие от традиционных алгоритмов, в которых минимизируют количество сравнений, в блинной сортировке требуется сделать как можно меньше переворотов [1].

1.2 Быстрая сортировка

Процесс сортировки массива $A[p..r]$ алгоритмом быстрой сортировки состоит из трех этапов:

1) массив $A[p..r]$ разбивается на два (возможно, пустых) подмассива $A[p..q-1]$ и $A[q+1..r]$, таких, что каждый элемент $A[p..q-1]$ меньше или равен $A[q]$, который, в свою очередь, не превышает любой элемент подмассива $A[q+1..r]$; индекс q вычисляется в ходе процедуры разбиения;

2) подмассивы $A[p..q-1]$ и $A[q+1..r]$ сортируются с помощью рекурсивного вызова процедуры быстрой сортировки;

3) поскольку подмассивы сортируются на месте, весь массив $A[p..r]$ уже оказывается отсортированным — никаких дополнительных операций объединений подмассивов не требуется [2].

Элемент $A[q]$ также называется опорным элементом. В разных реализациях он может выбираться по-разному, например опорным элементом может быть выбран первый элемент массива, средний, случайный или медиана первого, среднего и последних элементов.

1.3 Гномья сортировка

Алгоритм ищет первую пару соседних элементов, которые находятся в неправильном порядке, и затем меняет их местами. Он пользуется тем фактом, что после обмена элементов может появиться новая пара, нарушающая порядок, только до или после переставленных элементов. Алгоритм не проверяет, отсортированы ли элементы после текущей позиции, поэтому необходимо лишь проверить порядок до переставленных элементов [3].

Вывод

В данном разделе были рассмотрены алгоритмы блинной, быстрой и гномьей сортировок.

2 Конструкторская часть

В данном разделе будут приведены псевдокоды алгоритмов блинной, быстрой и гномьей сортировок, оценки их трудоемкостей.

2.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

— на вход программе подается два массива;

— результатом сортировки является массив, выводимый на экран;

— программа должна позволять производить измерения процессорного времени, затрачиваемого на выполнение реализуемых алгоритмов.

2.2 Разработка алгоритмов

На листинге 2.2 приведен псевдокод алгоритма блинной сортировки, на 2.1 — псевдокод вспомогательной функции `Flip`.

На листинге 2.3 приведен псевдокод алгоритма быстрой сортировки (разбиение Хоара).

На листинге 2.4 приведен псевдокод алгоритма гномьей сортировки.

Приведенные реализации предполагают, что на вход подаются корректные данные.

Листинг 2.1 Вспомогательная функция Flip

```
1: procedure FLIP(arr, maxNumPos, n)
2:     i ← maxNumPos
3:     while i < n do
4:         swap ← arr[i]
5:         arr[i] ← arr[n]
6:         arr[n] ← swap
7:         n ← n − 1
8:         i ← i + 1
9:     end while
10: end procedure
```

Листинг 2.2 Алгоритм блинной сортировки

Вход: ссылка на массив arr размером n

Выход: отсортированный массив arr

```
1: procedure PANCAKESORT(arr, n)
2:     if n ≥ 2 then
3:         i ← n
4:         while i > 1 do
5:             maxNumPos ← 0
6:             for a ← 0, i do
7:                 if arr[i] > arr[maxNumPos] then
8:                     maxNumPos ← a
9:                 end if
10:            end for
11:            if maxNumPos ≠ (i − 1) and maxNumPos ≥ 0 then
12:                FLIP(arr, maxNumPos, i)
13:            end if
14:        end while
15:    end if
16: end procedure
```

Листинг 2.3 Алгоритм быстрой сортировки (разбиение Хоара)

Вход: ссылка на массив arr, индексы st и end

Выход: отсортированный массив arr

```
1:  procedure QSortHoare(arr, st, end)
2:      if st ≥ 0 and end ≥ 0 and st < end then
3:          l ← st
4:          r ← end
5:          piv ← arr[(l+r)/2]
6:          loop
7:              do
8:                  l ← l + 1
9:              while arr[l] < piv
10:             do
11:                 r ← r − 1
12:             while arr[r] > piv
13:             if l < r then
14:                 break
15:             end if
16:             swap ← arr[l]
17:             arr[l] ← arr[r]
18:             arr[r] ← swap
19:         end loop
20:         QSortHoare(arr, st, r)
21:         QSortHoare(arr, r + 1, end)
22:     end if
23: end procedure
```

Листинг 2.4 Алгоритм гномьей сортировки

Вход: ссылка на массив arr размером n

Выход: отсортированный массив arr

```
1:  procedure GnomeSort(arr, n)
2:      i ← 1
3:      j ← 2
4:      while i < n do
5:          if arr[i − 1] < arr[i] then
6:              i ← j
7:              j ← j + 1
8:          else
9:              swap ← arr[i − 1]
10:             arr[i − 1] ← arr[i]
11:             arr[i] ← swap
12:             i ← i − 1
13:             if i == 0 then
14:                 i ← j
15:                 j ← j + 1
16:             end if
17:         end if
18:     end while
19: end procedure
```

2.3 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

— *массив* — динамический массив значений целого типа.

2.4 Модель вычисления для проведения оценки трудоемкости

Введена модель вычислений, которая потребуется для определения трудоемкости каждого отдельного взятого алгоритма умножения матриц.

1) Трудоемкость базовых операций имеет:

   — значение 1 для операций:

$$+, -, =, + =, - =, ==, ! =, <, >, <=, >=, \\ [], ++, --, \&\&, ||, >>, <<, \&, | \tag{2.1}$$

   — значение 2 для операций:

$$*, /, \%, * =, / =, \% = . \tag{2.2}$$

2) Трудоемкость условного оператора:

$$f_{\text{if}} = \begin{cases} \min(f_1, f_2), & \text{лучший случай} \\ \max(f_1, f_2), & \text{худший случай.} \end{cases} \tag{2.3}$$

3) Трудоемкость цикла

$$f_{\text{for}} = f_{\text{инициализация}} + f_{\text{сравнение}} + \\ + M_{\text{итераций}} \cdot (f_{\text{тело}} + f_{\text{инкремент}} + f_{\text{сравнение}}). \tag{2.4}$$

4) Трудоемкость передачи параметра в функцию и возврат из нее равен 0.

2.5 Трудоемкость алгоритмов

Ниже приведены оценки трудоемкостей алгоритмов блинной, быстрой и гномьей сортировок.

Пусть требуется отсортировать массив A размером N.

2.6 Блинная сортировка

Трудоемкость блинной сортировки складывается из:

— трудоемкости прохода по $N-1$ элементам массива

$$f_{mainPass} = 2 + (N-1) \cdot (2 + f_{body} + f_{flip}) \tag{2.5}$$

— трудоемкости прохода по $N-1$ элементам массива во внутреннем цикле

$$f_{body} = 2 + (N-1) \cdot \left(2 + 3 + 3 + \begin{cases} 0, & \text{лучший случай} \\ 1, & \text{худший случай} \end{cases} \right) \tag{2.6}$$

— трудоемкости переворачивания элементов массива

$$f_{flip} = 3 + \begin{cases} 0, & \text{лучший случай} \\ 2 + (N-2) \cdot (2+7), & \text{худший случай} \end{cases} \tag{2.7}$$

Таким образом, трудоемкость блинной сортировки в худшем случае равна

$$\begin{gathered} f_{PanSortWorst} = 2 + (N-1) \cdot (4 + 9(N-1) + 5 + 9(N-2)) = \\ = 18N^2 - 36N + 20 \approx O(N^2) \end{gathered} \tag{2.8}$$

и в лучшем случае:

$$f_{PanSortBest} = 2 + (N-1) \cdot (4 + 8(N-1) + 3) = 8N^2 - 9N + 3 \approx O(N^2) \tag{2.9}$$

2.7 Быстрая сортировка

Трудоемкость быстрой сортировки (с выбором среднего элемента в качестве опорного) складывается из:

— трудоемкости прохода по N элементам массива (или подмассива)

$$f_{body} = f_{split} \cdot (5 + 7 + N \cdot (4 + f_{swap})) \tag{2.10}$$

— трудоемкости перестановки элементов

$$f_{swap} = 1 + \begin{cases} 0, & \text{лучший случай} \\ 2 + 3 + 2 + 2, & \text{худший случай} \end{cases} \tag{2.11}$$

— трудоемкости разбиения массива

$$f_{split} = \begin{cases} \log_2 N, & \text{лучший случай} \\ N, & \text{худший случай} \end{cases} \tag{2.12}$$

Таким образом, трудоемкость быстрой сортировки в худшем случае равна

$$f_{QSortWorst} = N \cdot (12 + 14N) = 12N + 14N^2 \approx O(N^2) \tag{2.13}$$

и в лучшем случае:

$$f_{QSortBest} = \log_2 N(12 + 5N) = 12 \log_2 N + 5N \log_2 N \approx O(N \log_2 N) \tag{2.14}$$

2.8 Гномья сортировка

Трудоемкость гномьей сортировки складывается из:

— трудоемкости прохода по $N - 1$ элементам массива

$$f_{body} = 2 + 2 + (N - 1) \cdot (2 + f_{backward}) \tag{2.15}$$

— трудоемкости обратного прохода по N элементам массива для выпол-

нения их обмена

$$f_{backward} = 6 + \begin{cases} 1, & \text{лучший случай} \\ N \cdot (3 + 4 + 2 + 1), & \text{худший случай} \end{cases} \quad (2.16)$$

Таким образом, трудоемкость гномьей сортировки в худшем случае равна

$$f_{GnSortWorst} = 4 + (N - 1) \cdot (8 + 10N) = 10N^2 - 2N - 4 \approx O(N^2) \quad (2.17)$$

и в лучшем случае:

$$f_{GnSortBest} = 4 + (N - 1) \cdot (8 + 1) = 9N - 5 \approx O(N) \quad (2.18)$$

Вывод

В данном разделе на основе теоретических данных были построены схемы реализуемых алгоритмов сортировки. Также была проведена оценка трудоемкостей алгоритмов.

3 Технологическая часть

В данном разделе приведены средства реализации программного обеспечения, сведения о модулях программы, листинг кода и функциональные тесты.

3.1 Средства реализации

В качестве языка программирования, используемого при написании данной лабораторной работы, был выбран C++ [4], так как в нем имеется контейнер `std::vector`, представляющий собой динамический массив данных произвольного типа, и библиотека `<ctime>` [5], позволяющая производить замеры процессорного времени.

В качестве средства написания кода была выбрана кроссплатформенная среда разработки *Visual Studio Code* за счет того, что она предоставляет функционал для проектирования, разработки и отладки ПО.

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` — файл, содержащий точку входа в программу;

— `algorithms.cpp` — файл, содержащий реализации алгоритмов сортировки;

— `measure.cpp` — файл, содержащий функции, замеряющие процессорное время выполнения реализуемых алгоритмов.

3.3 Реализация алгоритмов

На листинге 3.1 представлена реализация алгоритма блинной сортировки.

Листинг 3.1 – Реализация алгоритма блинной сортировки и вспомогательной функции `flip`

```
1  static void flip(std::vector<int> &arr, int m, int n)
2  {
3      int swap, i;
4      for (i = m; i < --n; i++)
5      {
6          swap = arr[i];
7          arr[i] = arr[n];
8          arr[n] = swap;
9      }
10 }
11 void PancakeSort(std::vector<int> &arr)
12 {
13     if (arr.size() < 2)
14         return;
15     for (int i = arr.size(); i > 1; --i)
16     {
17         int maxNumPos = 0;
18         for (int a = 0; a < i; ++a)
19             if (arr[a] > arr[maxNumPos])
20                 maxNumPos = a;
21
22         if (maxNumPos == (i - 1))
23             continue;
24         if (maxNumPos >= 0)
25             flip(arr, maxNumPos, i);
26     }
27 }
```

Листинг 3.2 – Реализация алгоритма гномьей сортировки

```
1  void GnomeSort(std::vector<int> &arr)
2  {
3      int i = 1, j = 2;
4      while (i < (int)arr.size())
5      {
6          if (arr[i - 1] < arr[i])
7          {
8              i = j++;
9          }
10         else
11         {
12             int swap = arr[i - 1];
13             arr[i - 1] = arr[i];
14             arr[i] = swap;
15             --i;
16             if (i == 0)
17                 i = j++;
18         }
19     }
20 }
```

Листинг 3.3 – Реализация алгоритма быстрой сортировки

```
1  void QuickSort(std::vector<int> &arr, int start, int end)
2  {
3      if (start < 0 || end < 0 || start >= end)
4          return;
5
6      int pivot = arr[start];
7      int l = start - 1, r = end + 1;
8      while (true)
9      {
10         do
11         {
12             ++l;
13         } while (arr[l] < pivot);
14         do
15         {
16             --r;
17         } while (arr[r] > pivot);
18
```

```
19          if (l >= r)
20              break;
21
22          int swap = arr[l];
23          arr[l] = arr[r];
24          arr[r] = swap;
25      }
26      QuickSort(arr, start, r);
27      QuickSort(arr, r + 1, end);
28  }
```

3.4 Функциональные тесты

На таблице 3.1 представлены функциональные тесты алгоритмов сортировки.

Таблица 3.1 – Функциональные тесты алгоритмов сортировки

Массив	**Результат**			
	Блинная	**Гномья**	**Быстрая**	**Ожидаемый**
$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[5, 4, 3, 2, 1]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[3, 8, 6, 4, 1]$	$[1, 3, 4, 6, 8]$	$[1, 3, 4, 6, 8]$	$[1, 3, 4, 6, 8]$	$[1, 3, 4, 6, 8]$
$[2, 2, 2, 2]$	$[2, 2, 2, 2]$	$[2, 2, 2, 2]$	$[2, 2, 2, 2]$	$[2, 2, 2, 2]$
$[2, 1]$	$[1, 2]$	$[1, 2]$	$[1, 2]$	$[1, 2]$
$[7]$	$[7]$	$[7]$	$[7]$	$[7]$

Вывод

Были реализованы алгоритмы сортировки (блинная, быстрая, гномья). Проведено тестирование реализованных алгортимов.

4 Исследовательская часть

В данном разделе будут приведены исследование временных характеристик реализуемых алгоритмов и оценка их затрат по памяти.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— Процессор: Intel i5-1035G1 (8) @ 3.600 ГГц.

— Оперативная память: 16 ГБайт.

— Операционная система: Manjaro Linux x86_64 (версия ядра Linux 5.15.131-1-MANJARO).

Во время проведения измерений времени ноутбук был подключен к сети электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 демонстрация работы программы для случая, когда пользователь выбирает опцию 1 «Сортировка массива» и передает в программу массив $[5, 3, 2, 9, 7]$.

```
                Меню
1. Сортировка массива с помощью:
   а) блинной сортировки;
   б) гномьей сортировки.
   в) быстрой сортировки;
2. Произвести замеры по времени реализуемых алгоритмов.
0. Выход.

Выберите опцию (0-2): 1

Введите размер сортируемого массива: 5
Введите сам массив: 5 3 2 9 7

Блинная сортировка: [ 2 3 5 7 9 ]
Гномья сортировка:  [ 2 3 5 7 9 ]
Быстрая сортировка: [ 2 3 5 7 9 ]

                Меню
1. Сортировка массива с помощью:
   а) блинной сортировки;
   б) гномьей сортировки.
   в) быстрой сортировки;
2. Произвести замеры по времени реализуемых алгоритмов.
0. Выход.

Выберите опцию (0-2): 0
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Временные характеристики

Исследование временных характеристик реализуемых алгоритмов производилось три раза:

1) на массивах, упорядоченных по возрастанию, размер которых изменяется от 1 до 1001 с шагом 100;

2) на массивах, упорядоченных по убыванию, размер которых изменяется от 1 до 1001 с шагом 100;

3) на неупорядоченных массивах, размер которых изменяется от 1 до 1001 с шагом 100;

В силу того, что время работы алгоритмов может колебаться в связи с различными процессами, происходящими в системе, для обеспечения более

точных результатов измерения для каждого алгоритма повторялись 100 раз, а затем бралось их среднее арифметическое значение.

На рисунке 4.2 показаны зависимости времени выполнения блинной, гномьей и быстрой сортировок от размера упорядоченного по возрастанию массива.

На рисунке 4.3 показаны зависимости времени выполнения блинной, гномьей и быстрой сортировок от размера упорядоченного по убыванию массива.

На рисунке 4.4 показаны зависимости времени выполнения блинной, гномьей и быстрой сортировок от размера неупорядоченного массива.

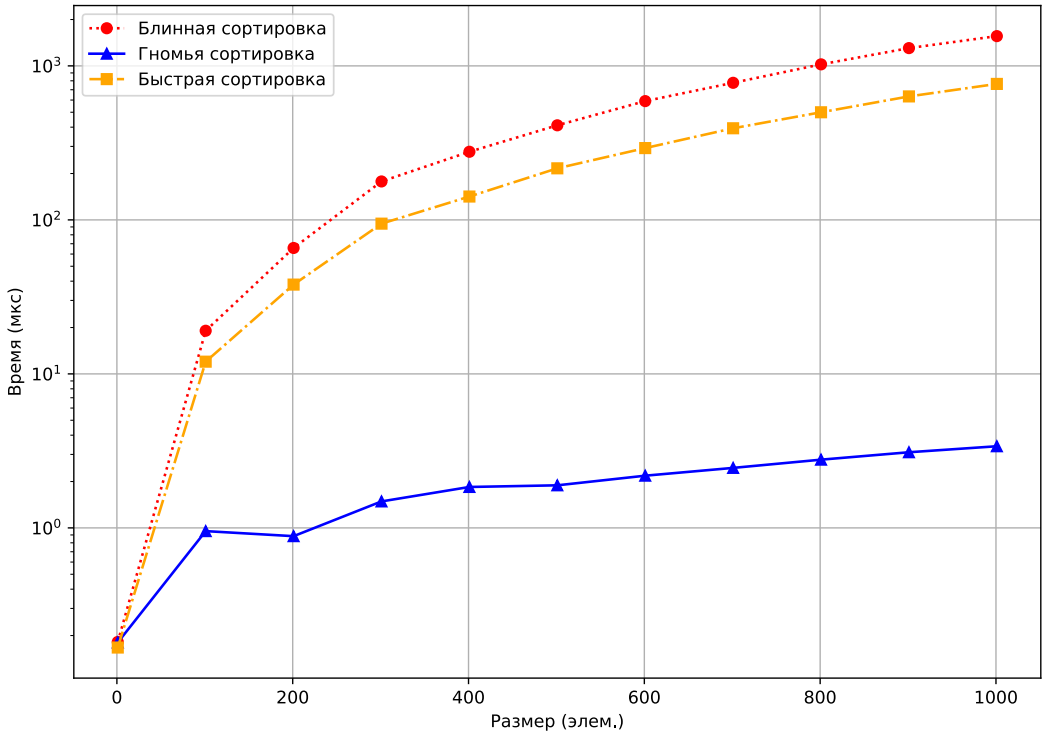

Рисунок 4.2 – Результат измерений времени работы реализуемых алгоритмов на массивах, упорядоченных по возрастанию

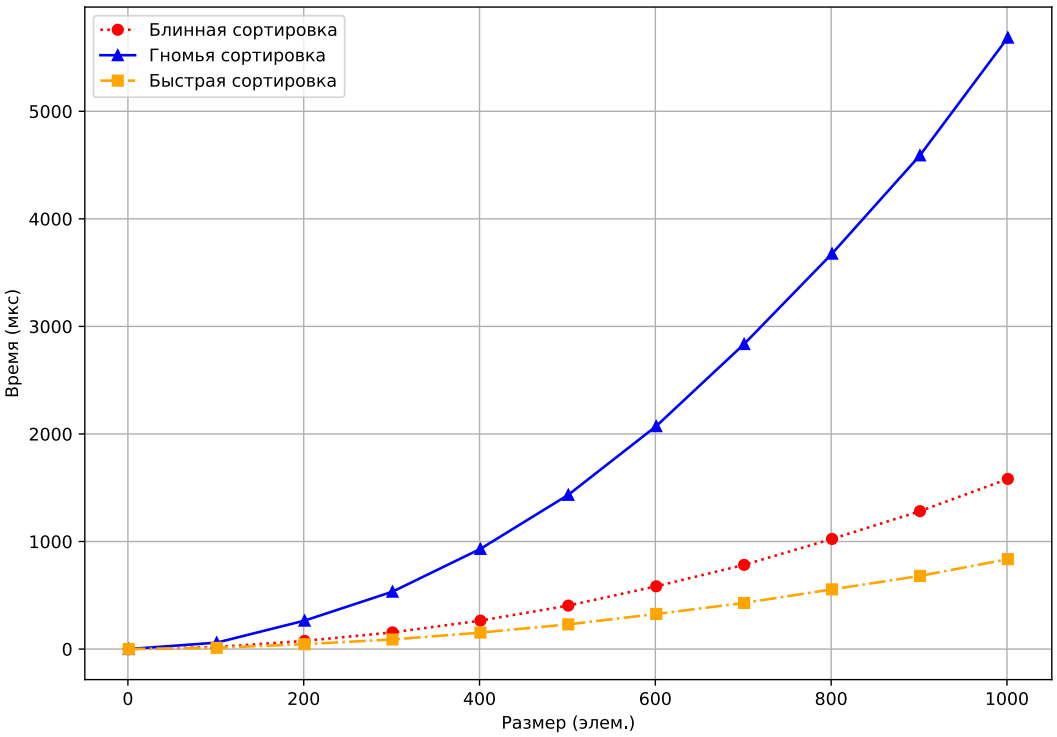

Рисунок 4.3 – Результат измерений времени работы реализуемых алгоритмов на массивах, упорядоченных по убыванию

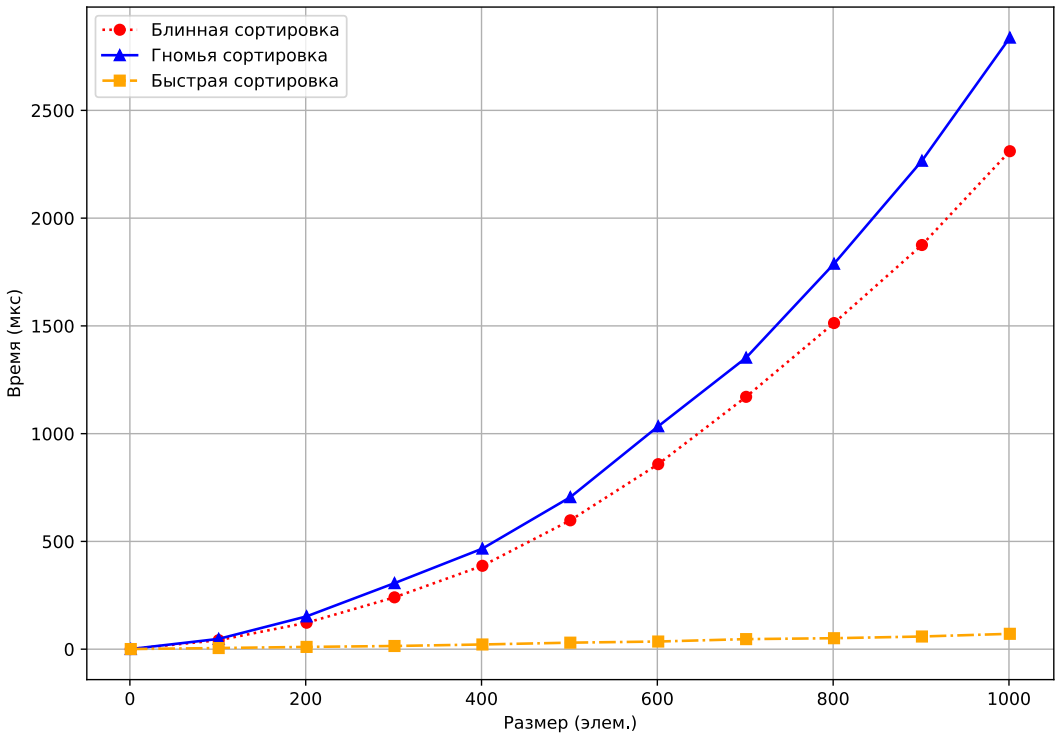

Рисунок 4.4 – Результат измерений времени работы реализуемых алгоритмов на неупорядоченных массивах

4.4 Характеристики по памяти

Введем следующие обозначения:

— size(v) — функция, вычисляющая размер входного параметра v в байтах;

— *int* — целочисленный тип данных.

Теоретически оценим объем используемой алгоритмами памяти при сортировке массива размером N.

4.4.1 Блинная сортировка

Оценка используемой блинной сортировкой памяти приведена в формуле 4.1.

$$M_{PanSort} = 6 \cdot \text{size}(int) + N \cdot \text{size}(int) + 8, \tag{4.1}$$

где $5 \cdot \text{size}(int)$ — размер дополнительных переменных,
$N \cdot \text{size}(int)$ — размер целочисленного массива,
8 — адрес возврата.

4.4.2 Гномья сортировка

Оценка используемой гномьей сортировкой памяти приведена в формуле 4.2.

$$M_{GnSort} = 3 \cdot \text{size}(int) + N \cdot \text{size}(int), \tag{4.2}$$

где $5 \cdot \text{size}(int)$ — размер дополнительных переменных,
$N \cdot \text{size}(int)$ — размер целочисленного массива.

4.4.3 Быстрая сортировка

Оценка используемой быстрой сортировкой памяти приведена в формуле 4.3.

$$M_{QSort} = M_{call} \cdot \begin{cases} \log_2 N, & \text{лучший случай} \\ N, & \text{худший случай} \end{cases} \tag{4.3}$$

где $M_{call} = (8 + 3 \cdot \text{size}(int) + N \cdot \text{size}(int))$ — размер памяти, затрачиваемый на один рекурсивный вызов,
$5 \cdot \text{size}(int)$ — размер дополнительных переменных,
$N \cdot \text{size}(int)$ — размер целочисленного массива,
$\log_2 N$ — глубина стека вызовов в лучшем случае,
N — глубина стека вызовов в худшем случае,
8 — адрес возврата.

Вывод

В результате исследования реализуемых алгоритмов по времени выполнения можно сделать следующие выводы:

1) алгоритм быстрой сортировки показал наибольшую скорость выполнения на упорядоченных по убыванию и неупорядоченных массивах (см. рисунки 4.3, 4.4);

2) на массивах, упорядоченных по возрастанию, наиболее эффективным оказался алгоритм гномьей сортировки (см. рисунок 4.2), однако в остальных случаях этот алгоритм показал наименьшую производительность;

3) на неупорядоченных массивах скорость работы быстрой сортировки выше, чем на упорядоченных.

В результате теоретической оценки алгоритмов по памяти можно сделать вывод о том, что алгоритм гномьей сортировки является наименее ресурсозатратным. Алгоритм быстрой сортировки, напротив, требует больше всего памяти, что объясняется тем, что алгоритм рекурсивный, и на каждый вызов функции требуется выделение памяти на стеке для сохранения информации, связанной с этим вызовом.

Заключение

В результате выполнения лаботарторной работы по исследованию алгоритмов сортировок решены следующие задачи:

1) описаны алгоритмы сортировки;
2) разработаны и реализованы соответствующие алгоритмы;
3) создан программный продукт, позволяющий протестировать реализованные алгоритмы;
4) проведена оценка трудоемкостей алгоритмов сортировки:
   - трудоемкость блинной сортировки в худшем случае — $O(N^2)$, в лучшем — $O(N^2)$;
   - трудоемкость быстрой сортировки в худшем случае — $O(N^2)$, в лучшем — $O(N \log_2 N)$;
   - трудоемкость гномьей сортировки в худшем случае — $O(N^2)$, в лучшем — $O(N)$;
5) проведен сравнительный анализ процессорного времени выполнения реализованных алгоритмов:
   - алгоритм быстрой сортировки показал наибольшую скорость выполнения на упорядоченных по убыванию и неупорядоченных массивах;
   - на массивах, упорядоченных по возрастанию, наиболее эффективным оказался алгоритм гномьей сортировки, однако в остальных случаях этот алгоритм показал наименьшую производительность;
   - на неупорядоченных массивах скорость работы быстрой сортировки выше, чем на упорядоченных.
6) выполнена теоретическая оценка объема затрачиваемой памяти каждым из реализованных алгоритмов: алгоритм гномьей сортировки является наименее ресурсозатратным; алгоритм быстрой сортировки, напротив, требует больше всего памяти, что объясняется тем, что

алгоритм рекурсивный, и на каждый вызов функции требуется выделение памяти на стеке для сохранения информации, связанной с этим вызовом.

Список использованных источников

1. Gates William H., Papadimitriou Christos H. Bounds for sorting by prefix reversal // Discrete Mathematics. 1979. Т. 27, № 1. URL: — Режим доступа: https://www.sciencedirect.com/science/article/pii/0012365X79900682.

2. Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л. [и др.] // Алгоритмы: построение и анализ. Издательский дом «Вильямс», 2013. — С. 1328.

3. Black Paul E. gnome sort. Dictionary of Algorithms and Data Structures. 2022. URL: — Режим доступа: https://www.nist.gov/dads/HTML/gnomeSort.html.

4. Документация по Microsoft C++ [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170&viewFallbackFrom=vs-2017 (дата обращения: 25.09.2023).

5. Standard library header <ctime> [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/header/ctime.