Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Отчет по заданию №1

«Методы сортировки»

Вариант 1 / 3 / 3 / 5

Выполнил:
студент 102 группы
Грознецкий А. Е.

Преподаватель:
Смирнов Л. В.

Москва
2023

Содержание

Постановка задачи

Введение

Была поставлена задача реализовать два метода сортировки массива и провести их экспериментальное сравнение. Элементами массива являются целые числа типа ***int***, т. е. целые знаковые 32-битные числа. Упорядочивание чисел должно происходить ***по неубыванию модулей*** (без учета знака). Для сравенения были выбраны сортировки: ***методом Шелла*** и ***пирамидальная сортировка***.

Массивы чисел

Для каждого из реализуемых методов сортировки необходимо предусмотреть возможность работы с массивами длины от 1 до N ($N \geqslant 1$). Память для хранения массива чисел следует выделять *динамически*.

Экспериментальное сравнение

Экспериментальное сравнение методов сортировок заключается в сопоставлении **числа сравнений** элементов и **числа перестановок** элементов. Важным условием сравнения является сравнение на одинаковых исходных массивах, при этом следует рассмотреть массивы разной длины: $n = 10, 100, 1000, 10000$. Тестирование сортировок должно производиться на различных типах исходных данных:

- элементы уже упорядочены (1);
- элементы упорядочены в обратном порядке (2);
- расстановка элементов случайна (3, 4).

Результаты экспериментов должны быть оформлены на основе нескольких запусков программы, их необходимо оформить в виде таблицы.

Результаты экспериментов

Сортировка методом Шелла

Описание метода

Сортировка **методом Шелла** является усовершенствованным вариантом *сортировки вставками*, где в массиве чисел индуктивно рассматривается *подмассив* с фиксированным левым краем (упорядоченный на предыдущем шаге индукции), к которому добавляется *следующий элемент*, место которого однозначно опрделяется за линейное (от длины подмассива) число сравнений. Таким образом, для каждого подмассива длины от 1 до N — длины самого массива — выполняется линейное число сравнений, и итоговая сложность является *квадратичной.*

Усовершенстововавание сортирвки вставками (то есть метод Шелла) состоит в том, что сравнение происходит не только рядом стоящих элементов, но и элементов, удаленных на определенное расстояние d. Это позволяет *грубо просеять* массив для более эффективного применения сортировки вставками в финале.

В классическом методе Шелла для массива длины N происходит $\lfloor \log_2 N \rfloor$ *просеиваний*, для каждого из которых расстояние d определяется как:

$$d_i = \lfloor \frac{N}{2^i} \rfloor \tag{1}$$

Таким образом, на i-м *просеивании* происходит лишь сортировка вставками всех подмассивов, в которых элементы отстоят друг от друга на расстоянии d_i.

Теоретическая оценка

Проведем теоретическую оценку количества сравнений при сортировке *методом Шелла.* Для удобства подсчёта примем $N = 2^m$. В таком случае выражение (1) примет вид:

$$d_i = \lfloor \frac{N}{2^i} \rfloor = 2^{m-i} \tag{2}$$

Тогда для начала посчитаем число сравнений k_i, соответствующее i-му *просеиванию.* Не трудно заметить, что для элементов массива с 1 по d_i нет элемента, который отстоит от них на расстоянии d_i, значит для них не будет происходит сравнений с левостоящими элементами. Для элементов с $d_i + 1$ по $2d_i$ есть лишь один элемент на расстоянии, кратном d_i, а следовательно для них возможно (и реализуется в худшем слуаче) одно сравнение. Аналогично рассуждая для всех элементов, получим выражение для значения k_i в худшем случае:

$$k_i = 0d_i + 1d_i + 2d_i + 3d_i + \cdots + (2^i - 1)d_i = d_i \sum_{j=0}^{2^i-1} j = d_i \frac{2^i(2^i - 1)}{2}$$

То есть:

$$k_i = d_i 2^{i-1}(2^i - 1) \tag{3}$$

Подставляя в выражение (3) полученное ранее выражение (2), получим:

$$k_i = 2^{m-i}2^{i-1}(2^i - 1) = 2^{m-1}(2^i - 1)$$

Тогда результирующее число сравнений $\overline{C}(N)$ *в худшем случае* будет суммой числа сравнений по всем $\lfloor \log_2 N \rfloor = m$ *просеиваниям*:

$$\overline{C}(N) = \sum_{i=1}^{m} k_i = \sum_{i=1}^{m} 2^{m-1}(2^i - 1) = 2^{m-1}\sum_{i=1}^{m}(2^i - 1) =$$
$$= 2^{m-1}(\sum_{i=1}^{m} 2^i - \sum_{i=1}^{m} 1) = 2^{m-1}((2^m - 1) \cdot 2 - m) =$$
$$= 2^m(2^m - 1 - \frac{m}{2}) = N(N - 1 - \frac{\log_2 N}{2}) = O(N^2)$$

Таким образом, верхняя граница сложности сортировки **методом Шелла** в классическом случае геометрической прогрессии — $O(N^2)$.

Подсчет числа сравнений в лучшем случае (массив уже отсортирован) проведем так же для $N = 2^m$. Отличаться будут лишь выражения для k_i, так как для оставновки очередного *просеивания* будет достаточно одного сравнения. Имеем:

$$k_i = d_i(2^i - 1) = 2^{m-i}(2^i - 1) \tag{4}$$

Получим нижнюю границу числа сравнений $\underline{C}(N)$:

$$\underline{C}(N) = \sum_{i=1}^{m} k_i = \sum_{i=1}^{m} 2^{m-i}(2^i - 1) = 2^m \sum_{i=1}^{m} \frac{2^i - 1}{2^i} =$$
$$= 2^m \sum_{i=1}^{m}(1 - \frac{1}{2^i}) = 2^m(\sum_{i=1}^{m} 1 - \sum_{i=1}^{m} \frac{1}{2^i}) = 2^m(m - (1 - \frac{1}{2^m})) =$$
$$= 2^m(m - 1 + \frac{1}{2^m}) = N(\log_2 N - 1 + \frac{1}{N}) = O(N \log N)$$

Таким образом, в лучшем случае сложность сортировки **методом Шелла** для интервалов, образующих геометрическую прогрессию, — $O(N \log N)$.

Экспериментальная оценка

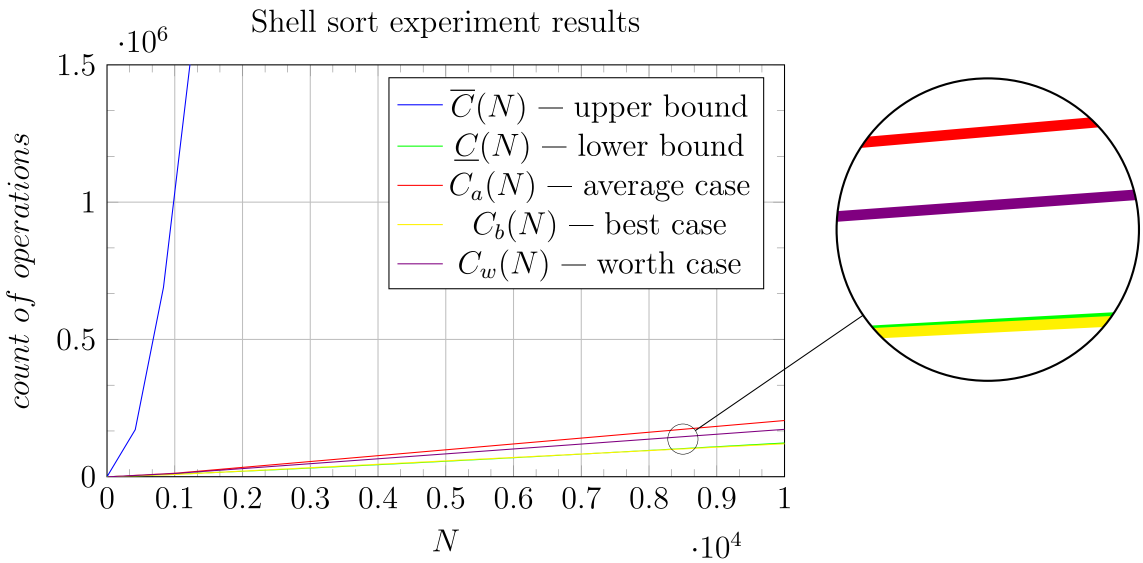
В экспериментальной оценке попытаемся выявить среднюю сложность сортировки **методом Шелла** и сравнить ее с тереотичесими нижней и верхней границами, а также сравнить нижние и верхние теоретические границы с экспериментальными.

Для начала сравним теоретические и экспериментальные оценки верхней и нижней границ, для этого проведем серию запсуков программы сортировки на различных исходных данных и массивах различной длины и отобразим результаты в таблице 1 (средние значения округляются вниз).

Таблица 1: Результаты работы сортировки методом Шелла

N	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	22	27	28	27	26
	Перемещения	0	13	7	11	7
100	Сравнения	503	668	904	851	731
	Перемещения	0	260	450	400	277
1000	Сравнения	8006	11716	15877	15065	12666
	Перемещения	0	4700	8372	7584	5164
10000	Сравнения	120005	172578	258153	268041	204694
	Перемещения	0	62560	143232	153063	89713

Изобразим *числа сравнений* из таблицы 1 на графике: средние значения $C_a(N)$ (столбец *среднее значение*), значения уже упорядоченного массива $C_b(N)$ (столбец 1) и значения обратно упорядоченного массива $C_w(N)$ (столбец 2), а также теоретические верхнюю и нижнюю границы, т. е. функции $\overline{C}(N)$ и $\underline{C}(N)$.



Какие выводы можно сделать:

1. Не трудно видеть, что $C_a(N)$ лежит между $\underline{C}(N)$ и $\overline{C}(N)$, что и ожидалось.

2. Также видно, что $C_b(N)$ лежит ниже теоретической границы $\underline{C}(N)$. Объясняется это тем, что при подсчете теоретической нижней границы рассматривался случай числа элементов массива $N = 2^m$ и как видно из формулы (3) реальное число сравнений должно быть меньше, так как в результирующей сумме каждое d_i будет округлено вниз — формула (1).

3. Кажется странным, что $C_w(N)$ — *worth case* — лежит ниже среднего значения $C_a(N)$, но дело в том, что обратно упорядоченный массив не является тем самым *worth case* для сортировки **методом Шелла**. Худшим случаем для алгоритма *Shell sort* является массив, в котором на каждом

просеивании будут возникать независимые подмассивы, изменение порядка элементов в которых *не улучшает ситуации*. То есть массив вида: $a_1, a_{\frac{n}{2}}, a_2, a_{\frac{n}{2}+1}, a_3, a_{\frac{n}{2}+2}, \ldots, a_{\frac{n}{2}-1}, a_n$ — применение к такому массиву *просеиваний* с шагом d_i отличным от 1 не изменяет его, поэтому худший случай достигается при орагнизации массива таким образом, что каждый из подмассивов (четный и нечетный) обладают свойствами исходного массива. Здесь подразумевается, что $a_1, a_2, a_3, \ldots, a_n$ — упорядочен.

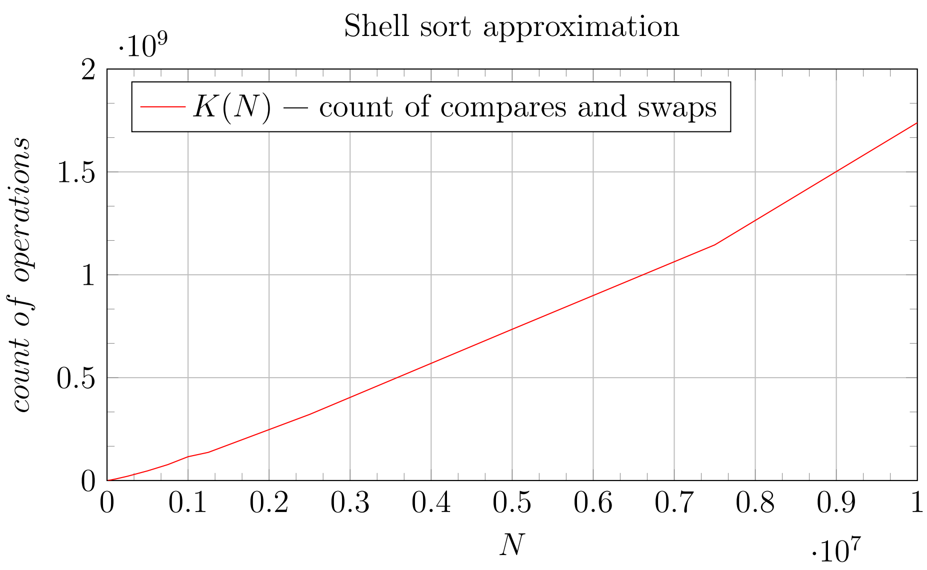
Для дальнейшего сравнения алгоритмов *Shell sort* и *Heap sort* построим график функции $K(N) = C(N) + S(N)$ — суммарного числа операций: *сравнения* $C(N)$ и *перестановки* $S(N)$. Результаты запусков *Shell sort* на *рандомных* массивах **большей длины** отображены в таблице 2.

Таблица 2: Экспериментальные данные сортировки методом Шелла

N	10000	50000	100000	250000	500000	750000	...
$C(N)$	267459	1899020	4256522	12236202	27610867	45237571	...
$S(N)$	152527	1224551	2806907	8363321	19363502	32121299	...
$K(N)$	419986	3123571	7063429	20599523	46974369	77358870	...

...	1000000	1250000	2500000	5000000	7500000	10000000
...	66656167	80010998	185038268	418826708	649556120	977036440
...	49157989	56898655	136300443	316338734	495849092	762049747
...	115814156	136909653	321338711	735165442	1145405212	1739086187

На основе данных из таблицы 2 построим график зависимости *общего числа операций* в алгоритме сортировки **методом Шелла** от *длины массивы*, к которому обратимся позже:



Пирамидальная сортировка

Описание метода

Пирамидальная сортировка — это метод сортировки, основанный на такой структуре данных, как бинарная куча, благодаря устройству которой, возможно получить наибольший (наименьший) элемент в убывающей (возрастающей) куче за $O(1)$. Сам же алгоритм можно разделить на два этапа:

1. Итеративное построение кучи из исходного массива.

2. Итеративное извлечение максимального элемента из кучи и его добавление в конец массива.

Причем реализация кучи возможна в самом массиве данных, благодаря чему алгоритм *Heap sort* не требует дополнительной памяти (требует $O(1)$).

Теоретическая оценка

Оценку сложности алгоритма можно разбить на две оценки его независимых частей: оценка сложности алгоритма *построения пирамиды* и оценка сложности *извлечения элементов из готовой пирамиды*. Начнем с первого: как уже было отмечено, построение пирамиды — итеративный процесс, поэтому проведем подсчёт числа сравнений на отдельной итерации, то есть при добавлении i-го элемента к куче размера $i-1$, нетрудно понять, что это требует $\log_2 i$ сравнений в худшем случае. Воспользуемся ранее введёнными при подсчёте алгоритмической сложности сортировки *Shell sort* обозначениями и запишем число сравнений на i-й итерации:

$$k_i = \log_2 i \tag{5}$$

Тогда результирующее число сравнений для создания пирамиды это:

$$C_1(N) = \sum_{i=1}^{N} k_i = \sum_{i=1}^{N} \log_2 i = \log_2 \prod_{i=1}^{N} i = \log_2 N! \tag{6}$$

Не трудно заметить, что *число сравнений* элементов массива второго этапа $C_2(N) = C_1(N)$, так как происходит суммирование тех же самых величин, но лишь в другом порядке, что не меняет результата. Поэтому результирующая сложность пирамидальной сортировки есть $C(N) = C_1(N) + C_2(N) = 2\log_2 N!$

Далее, для того чтобы привести выражение $C(N) = 2\log_2 N!$ к более знакомому виду, воспользуемся **формулой Муавра — Стирлинга** приближенного вычисления гамма-функции:

$$\ln\Gamma(N+1) = \ln N! = N\ln N - n + O(\ln N) \tag{7}$$

Получим:

$$C(N) = 2\log_2 N! = \frac{2}{\ln 2}\ln N! = \frac{2}{\ln 2}(N\ln N - N + O(\ln N)) = O(N\log N) \tag{8}$$

Таким образом, алгоритм **пирамидальной сортировки** имеет *квазилинейную* сложность: $O(N \log N)$.

Экспериментальная оценка

В экспериментальной оценке получим среднюю сложность **пирамидальной сортировки** и сравним её с теоретической. Для этого проведем серию запсуков программы сортировки на различных исходных данных и отобразим результаты в таблице 3 (средние значения округляются вниз). Здесь важно отметить, что запуски происходят на тех же самых массивах, что и *Shell sort* (о том, как это было реализовано подробно рассказано в соотетствующем разделе *спецификация функций*).

N	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	41	35	38	38	38
	Перемещения	31	22	30	28	27
100	Сравнения	1081	944	1031	1023	1019
	Перемещения	641	517	579	583	580
1000	Сравнения	17583	15965	16863	16834	16811
	Перемещения	9709	8317	9101	9087	9053
10000	Сравнения	244460	226682	235458	235256	235464
	Перемещения	131957	116697	124285	124083	124255

Таблица 3: Результаты работы пирамидальной сортировки

Для построения графика функции $C(N)$ упростим выражение (8). В формуле (7) следующий член в разложении $O(\ln n)$ — это $\frac{1}{2}\ln(2\pi n)$ и формулу (7) можно записать в эквивалентной форме:

$$N! \sim \sqrt{2\pi N}\left(\frac{N}{e}\right)^N \tag{9}$$

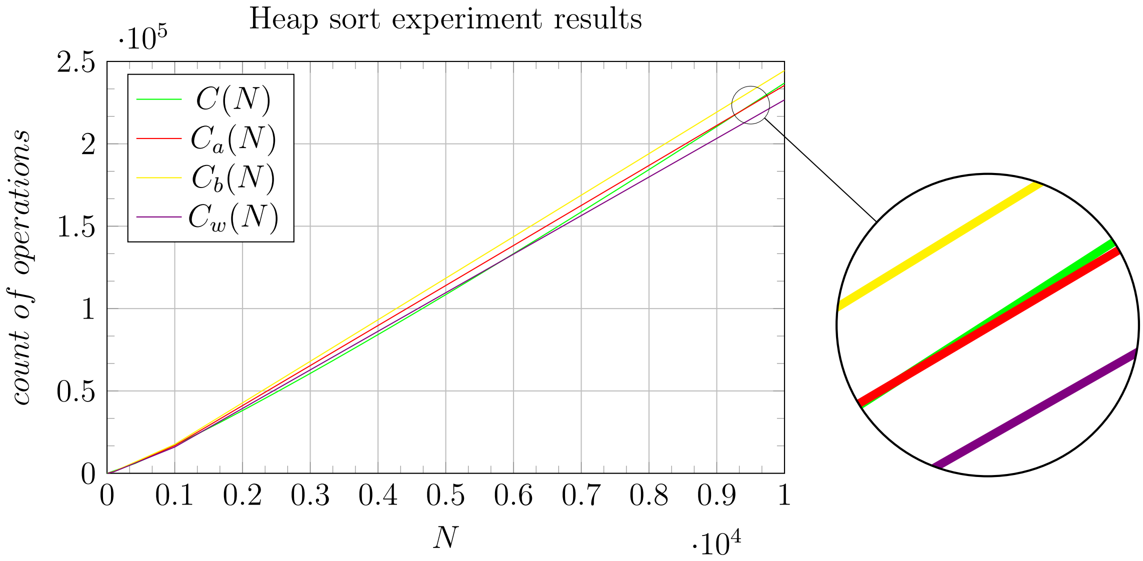
Для изображения графика функции $C(N)$ будем пользоваться этой формулой, так как она не дискретна, и уже поддерживается встроенным пакетом изображения графиков (в отличие от тоже не дискретной гамма-функции).

$$C(N) = \frac{2}{\ln 2}\ln N! \sim \frac{2}{\ln 2}\ln\left(\sqrt{2\pi n}\left(\frac{N}{e}\right)^N\right) = \frac{2N}{\ln 2}\ln\left(\frac{N}{e}(2\pi N)^{\frac{1}{2N}}\right) \sim$$
$$\sim \frac{2N}{\ln 2}\ln\left(\frac{N}{e}N^{\frac{1}{2N}}\right) = \frac{2N}{\ln 2}\ln\left(\frac{N^{\frac{2N+1}{2N}}}{e}\right) \sim \frac{2N}{\ln 2}\ln\frac{N}{e} = 2N\log_2\frac{N}{e}$$

Получим окончательно:

$$C(N) = 2N\log_2\frac{N}{e} \tag{10}$$

Теперь можем изобразить *числа сравнений* из таблицы 3 на графике: средние значения $C_a(N)$ (столбец *среднее значение*), значения уже упорядоченного массива $C_b(N)$ (столбец 1) и значения обратно упорядоченного массива $C_w(N)$ (столбец 2), а также график теоретической сложности *пирамидальной сортировки* $C(N)$.



Какие выводы можно сделать:

1. Аналогично *Shell sort* в *Heap sort* можно заметить, что *worth case* не является *худшим*, и как бы это ни было странно, *worth case* даже лучше *best case*. Все потому, что в *Heap sort* первым шагом создается пирамида, для которой уже упорядоченный массив является худшим случаем, в отличие от обратно упорядоченного, где с добавлением последнего элемента происходит его *подъем* на самый верх.

2. Также стоит отметить, что средняя число сравнений $C_a(N)$ довольно точно совпадает с теоретическим (N).

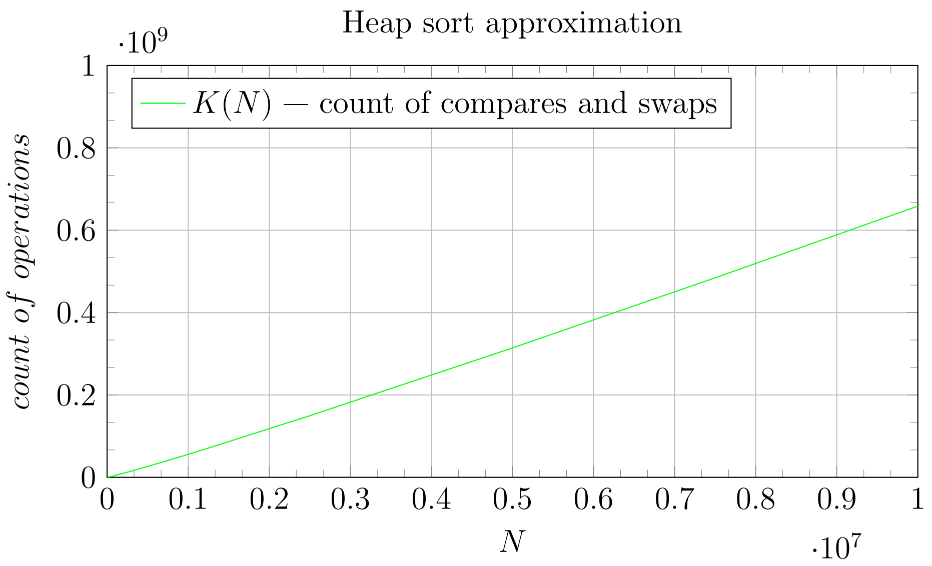
Далее, для сравнения алгоритмов *Shell sort* и *Heap sort* построим график зависимости *числа сравнений* в алгоритме **пирамидальной сортировки** от длины массивы, пробегающей **больший диапазон значений** — аналогично *Shell sort*. Результаты запусков программы на *сгенерированных случайным образом* массивах отображены в таблице 4.

Таблица 4: Экспериментальные данные сортировки методом Шелла

N	10000	50000	100000	250000	500000	750000	$\cdots$
$C(N)$	235331	1409597	3019675	8198049	17396712	27007867	$\cdots$
$S(N)$	124224	737333	1574994	4261890	9023743	13992818	$\cdots$
$K(N)$	359555	2146930	4594669	12459939	26420455	41000685	$\cdots$

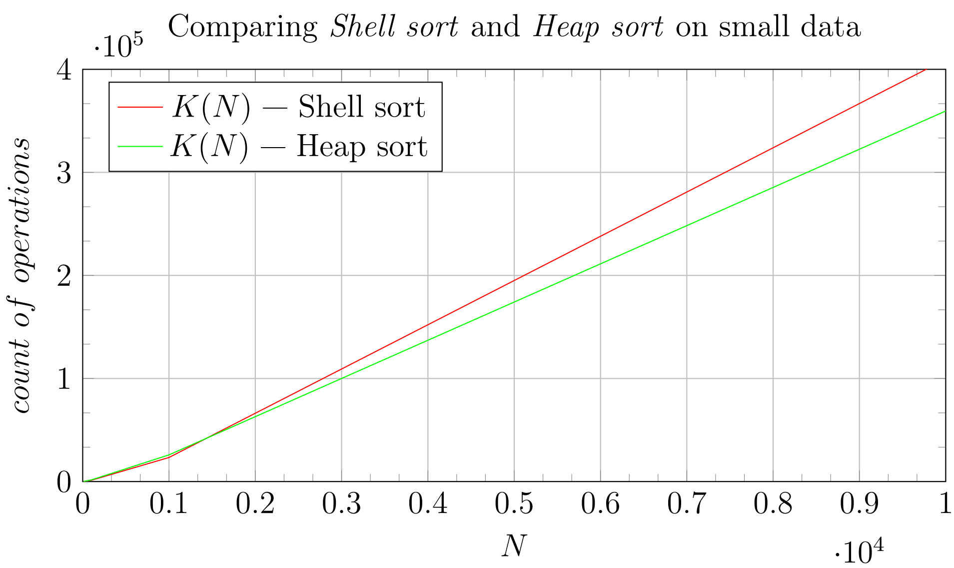
...	1000000	1250000	2500000	5000000	7500000	10000000
...	36794039	46829302	98659720	207320663	319711305	434644541
...	19048032	24228898	50957404	106916725	164739927	223835869
...	55842071	71058200	149617124	314237388	484451232	658480410

На основе данных из таблицы 4 построим график зависимости *общего числа операций* в алгоритме **пирамидальной сортировки** от длины массивы, которым воспользуемся уже в следующем разделе:

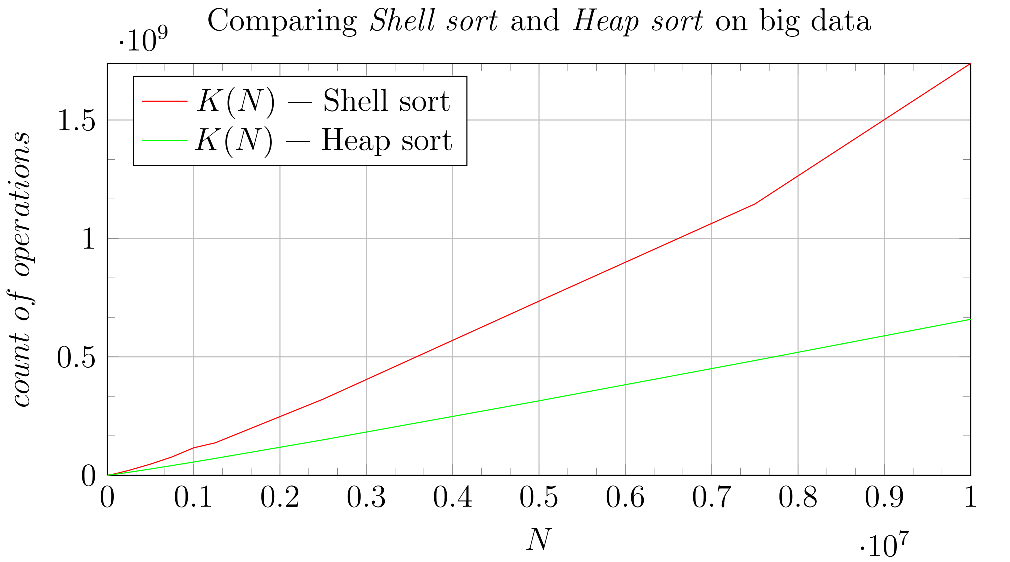


Сравнение алгоритмов *Shell sort* и *Heap sort*

Рассмотрим полученные данные и проанализируем их отношения. Для начала изобразим общее число операций *Shell sort* и *Heap sort* при $N = 10, 100, 1000, 10000$.

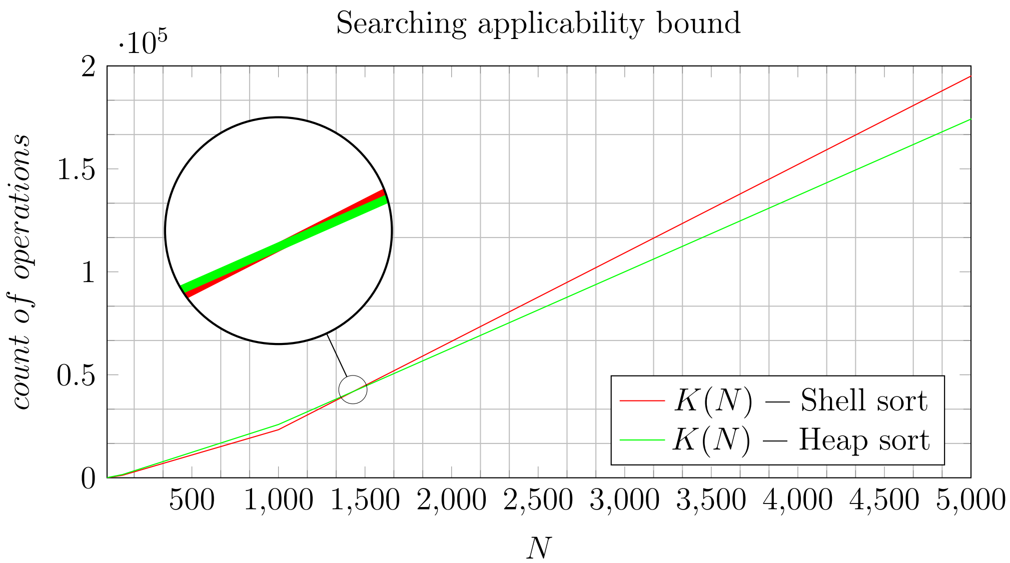


Заметим, что сначала график, соответствующий *Shell sort*, лежит ниже *Heap sort* — из этого можем сделать вывод, что для небольших массивов есть смысл использовать *Shell sort*. Но уже при достаточно больших N происходит обратное. Для иллюстрации этого факта построим графики общего числа операций $K(N)$ для N, пробегающего **больший диапазон значений**:



При большем N разница в числе операций становится существенной и *Shell sort* проигрывает алгоритму *Heap sort*. Как было написано выше, это происходит благодаря стабильной *квазилинейной* сложности алгоритма *Heap sort* — $O(N \log N)$ — что для алгоритма *Shell sort* является лишь сложностью лучшего случая (когда массив уже отсортирован).

Попробуем выяснить более точные границы диапазона резонного применения алгоритма *Shell sort*. Для этого приближенно найдем точку пересечения построенных графиков:



Получим границу резонного применения алгоритма *Shell sort*: $N \approx 1500$.

Вывод

1. Таким образом, алгоритм сортировки **методом Шелла** работает за меньшее число операций, чем алгоритм **пирамидальной сортировки** при $N \leqslant 1500$, а при больших N меньшее число операций имеет алгоритм **пирамидальной сортировки**, что объясняется его стабильной квазилинейной сложностью.

2. Откртым остаётся вопрос подсчёта *теоретической* оценки средней сложности сортировки **методом Шелла** для интервалов, образующих геометрическую прогрессию $d_i = \frac{N}{2^i}$.

Структура программы и спецификация функций

Программу можно разделить на два логически независимых *модуля*:

1. Алгоритмы сортировки *Shell sort* и *Heap sort*.

2. Функций, отвечющие за предоставление *данных* алгоритмам сортировки и обработку их *ответов*.

Первый модуль включает в себя функции:

- `void shell_sort(int *a, int n)` — функция сортировки **методом Шелла**

- `void heap_sort(int *a, int n)` — функция **пирамидальной сортировки**

- `void heapify(int *a, int n, int i)` — вспомогательная функция для **пирамидальной сортировки**, которая рекурсивно *поднимает* i-й элемент кучи на его место.

Второй модуль включает в себя функции:

- `int rand_int(void)` — функция генерирует рандомное число типа `int`.

- `int *generate_array(int n, char mode, long long seed)` — функция генерирует в области *динамической* памяти массив длины `n` определённого типа, в зависимости от значения параметра `mode`, который может принимать значения: `b`, `w` и `r`. Конфигурация `b` генерирует уже упорядоченный массив, `w` — обратно упорядоченный, `r` — рандомный массив чисел, для генерации которых используется функция `rand_int`. Стоит отметить, что массив однозначно определен значением `seed`, которое задаётся в начале работы программы и используется для генерации одинаковых массивов. Данная функция **требует отдельного освобождения выделенной памяти**.

- `int is_sorted(int *a, int n)` — *логическая* функция проверяет упорядоченность массива

- `void test_func(int, char, long long, void(int *, int))` — функция-оболочка для вызова функции сортировки: генерирует массив необходимой конфигурации; вызывает сортирующую функцию, подсчитывая при этом число операций *сравнения* и *перестановки*; дает отчёт по результатам сортировки и освобождает выделенную память.

Отдельно рассмотрим функцию `int main(void)`: она лишь вызывает `test_func` с нужными параметрами.

Отладка программы, тестирование функций

Написание программного кода происходило в среде разработки `CLion`, что позволило избежать ошибок компилляции и отладки программы. Тестрирование функций сортировки происходит автоматически при каждом выполнении.

Список цитируемой литературы

Вспомогательная литература не была использована.

Анализ допущенных ошибок

Ошибок допущено не было.