МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания 4

Тема:

Определение эффективного алгоритма сортировки

Выполнил студент Цемкало А. Р.

Фамилия И.О.

группа ИКБО-10-20

Москва 2021

СОДЕРЖАНИЕ

Цель. Получить навыки по анализу вычислительной сложности нескольких алгоритмов сортировки и определение наиболее эффективного алгоритма. Разработать три алгоритма сортировки, определенные вариантом. Провести анализ вычислительной и емкостной сложности алгоритма на массивах, заполненных случайно. Определить наиболее эффективный алгоритм.

Вариант 3 (26 % 8 + 1).

Задание 1. Определение эффективного алгоритма в среднем случае.

1.1. Алгоритм сортировки по методу простого обмена (Пузырёк) (Exchange sort) с условием Айверсона.

Постановка задачи: разработать алгоритм простой сортировки, определенной в варианте, реализовать алгоритм. Сформировать таблицу Таблица 2 результатов сортировки по формату Таблица 1 для массива, заполненного случайными числами. Определить емкостную сложность алгоритма. Определить асимптотическую сложность алгоритма.

Описание подхода к решению: Пузырьковая сортировка заключается в сравнении соседних элементов и, при необходимости, в их перестановке. Условие Айверсона выполняется, если в очередном проходе внутреннего цикла не было выполнено ни одной операции перестановки, это позволяет закончить сортировку до завершения внешнего цикла.

Алгоритм: используются два цикла. Внешний проходится N − 1 раз. В этом цикле устанавливается (смещается) граница между отсортированной и неотсортированной частями массива. Во внутреннем цикле выполняются непосредственно операции сравнения и перестановки. Количество проходов внутреннего цикла в каждом следующем проходе внешнего цикла уменьшается (от N − 1 до 1). При этом используется условие Айверсона, что позволяет делать меньше проходов.

Код функции сортировки:

```
void bubble_sort(int* list, int n) {
	long long int compare = 2, swapping = 0;
	int k;
	for (int i = 0; i < n; i++) {
		compare++;
		k = 0;
		for (int j = n - 1; j > i; j--) {
			compare++;
			compare++;
			if (list[j - 1] > list[j]) {
				swapping++;
				int t = list[j];
				list[j] = list[j - 1];
				list[j - 1] = t;
				k = 1;
			}
		}
		compare++;
```

```
            if (k == 0) {
                    break;
            }
        }
        cout << "Number of comparisons: " << compare << endl;
        cout << " Number of swappings: " << swapping << endl;
}
```

Номер оператора		O(f(n)) для каждого оператора		
1	int k;	O(1)	O(1)	$O(n^2)$
2	for (int i = 0; i < n; i++) {	O(n)	O(n)	
3	k = 0;	O(1)		
4	for (int j = n - 1; j > i; j--) {	O(n)	O(n)	
5	if (list[j - 1] > list[j]) {	O(1)		

6	int t = list[j];
	list[j] = list[j - 1];
	list[j - 1] = t;
	k = 1;
		O(1)		

	}			
	}			
7	if (k == 0) {	O(1)	O(1)	
8	break;	O(1)		
	}			
	}			

Асимптотическая сложность алгоритма: квадратичная, так как в алгоритме используются два цикла. Т.е. $T(n)=\Theta(n^2)$.

Ёмкостная сложность: в данном случае ёмкостная сложность равна O(n), так как зависимость линейная (используется один одномерный массив).

Таблица 2

n	**T(n), сек**	**T_T=f(C+M)**	**$T_п$=Сф+Мф**
100	0.0039006	10000	9862+2360
1000	0.0067981	1000000	999880+248145
10000	0.168437	100000000	100006696+24952888
100000	22.0787	10000000000	10000091630+2506044511
1000000	2355.81	1000000000000	1000000845160+250123019598

1.2. Алгоритм шейкерной сортировки с условием Айверсона.

Постановка задачи: разработать алгоритм ускоренной сортировки, определенной в варианте, реализовать алгоритм. Сформировать таблицу Таблица 3 результатов сортировки по формату Таблица 1 для массива, заполненного случайными числами. Определить емкостную сложность алгоритма. Определить асимптотическую сложность алгоритма.

Описание подхода к решению: Шейкерная сортировка заключается в сравнении соседних элементов в двух направлениях поочередно и, при необходимости, в их перестановке. Условие Айверсона выполняется, если в очередном проходе внутренних циклов не было выполнено ни одной операции перестановки, сортировка заканчивается до завершения внешнего цикла.

Алгоритм: используются три цикла. Внешний проходится N – 1 раз. В этом цикле устанавливается (смещается) граница между отсортированной и неотсортированной частями массива. Во внутренних двух циклах в двух направлениях (по направлению на каждый) выполняются непосредственно операции сравнения и перестановки. Количество проходов внутреннего цикла в каждом следующем проходе внешнего цикла уменьшается (от N – 1 до 1). При этом используется условие Айверсона, что позволяет делать меньше проходов.

Код функции сортировки:

```
void cocktail_sort(int* list, int n) {
	long long int compare = 3, swapping = 0;
	int left = 0;
	int right = n - 1;
	int k;
	while (left < right) {
		compare++;
		k = 0;
		for (int i = left; i < right; i++) {
			compare++;
			compare++;
			if (list[i] > list[i + 1]) {
				swapping++;
				int t = list[i + 1];
				list[i + 1] = list[i];
				list[i] = t;
				k = 1;
			}
		}
		right--;
		for (int i = right; i > left; i--) {
			compare++;
			compare++;
			if (list[i] < list[i - 1]) {
				swapping++;
				int t = list[i - 1];
				list[i - 1] = list[i];
				list[i] = t;
				k = 1;
			}
		}
		left++;
		compare++;
		if (k == 0) {
			break;
		}
	}
	cout << " Number of comparisons: " << compare << endl;
	cout << " Number of swappings: " << swapping << endl;
}
```

Номер оператора		O(f(n)) для каждого оператора			

1	int left = 0;
	int right = n - 1;
	int k;
		O(1)	O(1)	O(1)	O(n^2)

2	while (left < right) {	O(n)	O(n)	O(n)	
3	k = 0;	O(1)	O(1)	O(1)	
4	for (int i = left; i < right; i++) {	O(n)	O(n)	O(n)	
5	if (list[i] > list[i + 1]) {	O(1)	O(1)		

6	int t = list[i + 1];
	list[i + 1] = list[i];
	list[i] = t;
	k = 1;
		O(1)			

	}				
	}				
7	right--;	O(1)	O(1)	O(1)	
8	for (int i = right; i > left; i--) {	O(n)	O(n)	O(n)	
9	if (list[i] < list[i - 1]) {	O(1)	O(1)		

10	int t = list[i - 1];
	list[i - 1] = list[i];
	list[i] = t;
	k = 1;
		O(1)			

	}				
	}				
11	left++;	O(1)	O(1)	O(1)	
12	if (k == 0) {	O(1)			
13	break;	O(1)			
	}				
	}				

Асимптотическая сложность алгоритма: квадратичная, так как в алгоритме используются два цикла. Т.е. $T(n)=\Theta(n^2)$.

Ёмкостная сложность: в данном случае ёмкостная сложность равна O(n), так как зависимость линейная (используется один одномерный массив).

Таблица 3

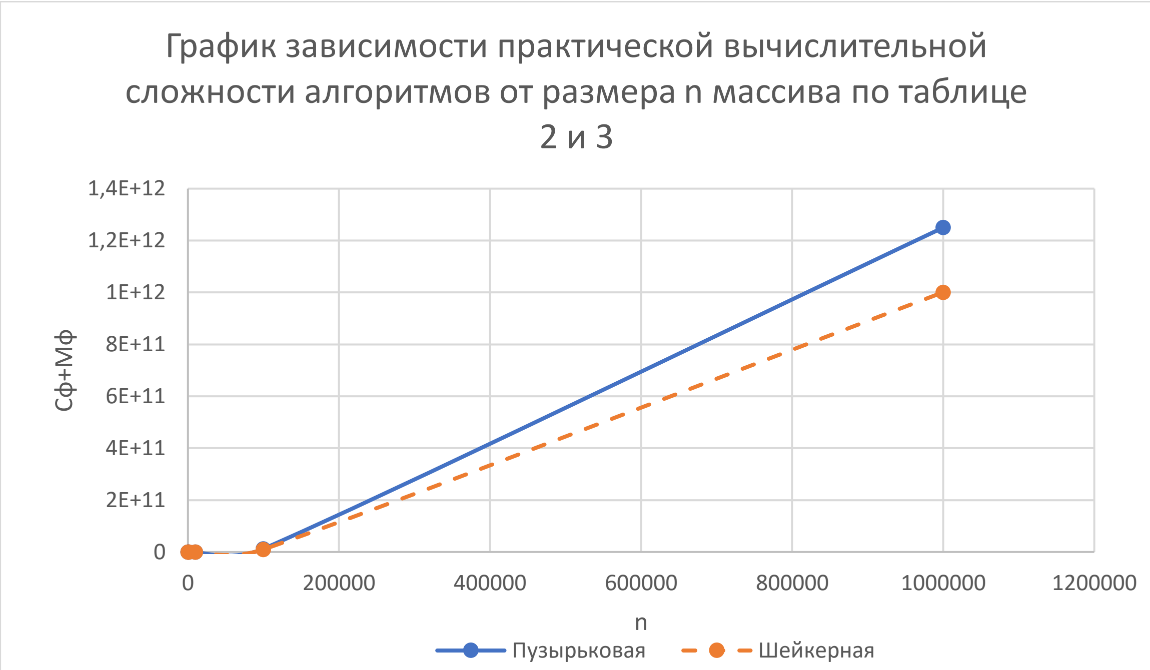
n	T(n), сек	T_T=f(C+M)	$T_п$=Сф+Мф
100	0.0059058	10000	7887+2360
1000	0.006915	1000000	757939+248145
10000	0.139417	100000000	75556867+24952888
100000	16.4861	10000000000	7519362367+2506044511
1000000	1777.74	1000000000000	750695515587+250123019598

1.3. Анализ полученных результатов по таблицам 2 и 3.

Теоретическая сложность алгоритма сортировки простого обмена с условием Айверсона и шейкерной сортировки с условием Айверсона одинаковы.

Однако практическая сложность по времени и по количеству операций перемещения и сравнения шейкерная сортировка является более эффективной.

1.4. График зависимости Сф+Мф для анализируемых алгоритмов.



1.5. Алгоритм ускоренной сортировки «Прямое слияние».

Код функции сортировки:

```
long long int compare = 0, swapping = 0;
void merge_sort(int* list, int n) {
	compare++;
	if (n == 0 || n == 1) {
		return;
	}
	int left_len = n / 2;
	int right_len = n - left_len;
	int* left = new int[left_len];
	int* right = new int[right_len];
	compare++;
	for (int i = 0; i < left_len; i++) {
		compare++;
		swapping++;
		left[i] = list[i];
```

```
		}
		compare++;
		for (int i = 0; i < right_len; i++) {
			compare++;
			swapping++;
			right[i] = list[n / 2 + i];
		}
		merge_sort(left, n / 2);
		merge_sort(right, n - n / 2);
		int x = 0, y = 0, k = 0;
		int* new_list = new int[n];
		compare++;
		while (x < left_len && y < right_len) {
			compare++;
			compare++;
			if (left[x] <= right[y]) {
				swapping++;
				new_list[k] = left[x];
				x++;
			}
			else {
				compare++;
				swapping++;
				new_list[k] = right[y];
				y++;
			}
			k++;
		}
		compare++;
		while (x < left_len) {
			compare++;
			swapping++;
			new_list[k] = left[x];
			x++;
			k++;
		}
		compare++;
		while (y < right_len) {
			swapping++;
			compare++;
			new_list[k] = right[y];
			y++;
			k++;
		}
		compare++;
		for (int i = 0; i < n; i++) {
			compare++;
			swapping++;
			list[i] = new_list[i];
		}
		delete new_list;
	}
```

Асимптотическая сложность алгоритма: $\Theta(n)$, т.к. вложенные циклы не используются, а рекурсия только разбивает сортировку на более простые задачи.

Ёмкостная сложность: в данном случае ёмкостная сложность равна $O(n)$, так как зависимость линейная (используются только одномерные массивы).

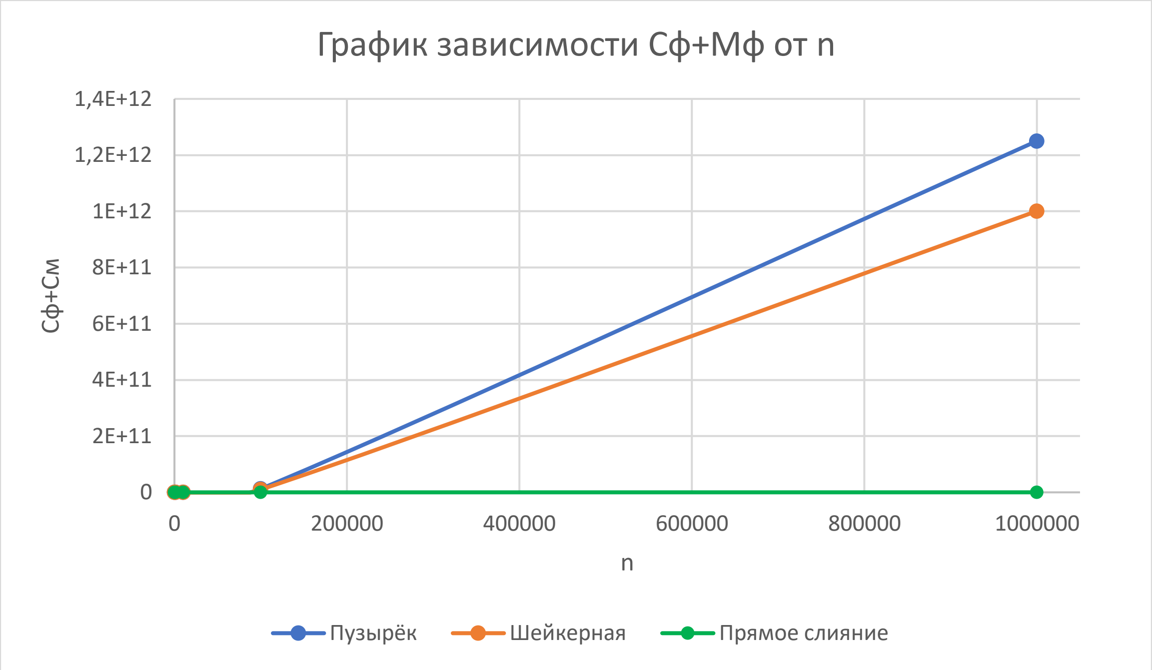
Таблица 4

n	T(n), сек	T_T=f(C+M)	$T_п$=Сф+Мф
100	0.0036712	100	3623+2016
1000	0.0079264	1000	50970+29928
10000	0.020344	10000	662768+400848
100000	0.0827153	100000	8119864+5006784
1000000	0.942694	1000000	95923537+59854272

1.6. Анализ полученных результатов по таблицам 3 и 4.

Из таблиц 3 и 4 видно, что алгоритм ускоренной сортировки «Прямое слияние» гораздо эффективнее по временной сложности, чем алгоритм Шейкерной сортировки с условием Айверсона. Особенно видна разница в тестировании на массиве из 1000000 чисел.

1.7. График зависимости Сф+Мф для анализируемых алгоритмов.



Задание 2. Определение эффективного из алгоритмов для наихудшего и наилучшего случаев

2.1. Таблицы с результатами прогонов на упорядоченных массивах.

Алгоритм сортировки Простого обмена (пузырёк) с условием Айверсона

- Исходный массив отсортирован в строго убывающем порядке значений элементов

Таблица 5

n	**T(n), сек**	**T_T=f(C+M)**	**$T_п$=Сф+Мф**
100	0.0049422	10000	10102+4950
1000	0.0086293	1000000	1001002+499500
10000	0.212322	100000000	100010002+49995000
100000	19.6431	10000000000	10000100002+4999950000
1000000	2876.52	1000000000000	1000001000002+499999500000

- Исходный массив отсортирован в строго возрастающем порядке значений элементов

Таблица 6

n	**T(n), сек**	**T_T=f(C+M)**	**$T_п$=Сф+Мф**
100	0.0045965	10000	202+0
1000	0.0044447	1000000	2002+0
10000	0.0060208	100000000	20002+0
100000	0.003865	10000000000	200002+0
1000000	0.0097565	1000000000000	2000002+0

Алгоритм Шейкерной сортировки с условием Айверсона

- Исходный массив отсортирован в строго убывающем порядке значений элементов

Таблица 7

n	**T(n), сек**	**T_T=f(C+M)**	**$T_п$=Сф+Мф**
100	0.0030794	10000	10003+4950
1000	0.0074335	1000000	1000003+499500
10000	0.225526	100000000	100000003+49995000
100000	20.8373	10000000000	10000000003+4999950000
1000000	3254.73	1000000000000	1000000000003+499999500000

- Исходный массив отсортирован в строго возрастающем порядке значений элементов

Таблица 8

n	T(n), сек	T_T=f(C+M)	$T_п$=Сф+Мф
100	0.0057376	10000	399+0
1000	0.0049854	1000000	3999+0
10000	0.006266	100000000	39999+0
100000	0.0068346	10000000000	399999+0
1000000	0.0113833	1000000000000	3999999+0

Алгоритм сортировки «Простое слияние»

- Исходный массив отсортирован в строго убывающем порядке значений элементов

Таблица 9

n	T(n), сек	T_T=f(C+M)	$T_п$=Сф+Мф
100	0.0036753	100	3521+2016
1000	0.0131032	1000	48009+29928
10000	0.0218556	10000	618857+400848
100000	0.084654	100000	7514585+5006784
1000000	0.783685	1000000	87987129+59854272

- Исходный массив отсортирован в строго возрастающем порядке значений элементов

Таблица 10

n	T(n), сек	T_T=f(C+M)	$T_п$=Сф+Мф
100	0.011503	100	3125+2016
1000	0.0070532	1000	48009+29928
10000	0.0185919	10000	618857+400848
100000	0.0919446	100000	7514585+5006784
1000000	0.811204	1000000	87987129+59854272

2.2. Асимптотическая вычислительная сложность для каждого алгоритма для лучшего и худшего случаев.

На основе таблиц 5-10 можно сделать вывод, что время выполнения алгоритмов Шейкерной сортировки с условием Айверсона и сортировки Простого обмена (пузырёк) с условием Айверсона зависит от исходной упорядоченности массива. Однако время выполнения алгоритма сортировки «Простое слияние» примерно одинаково независимо от упорядоченности исходного массива.

- Алгоритм сортировки Простого обмена (пузырёк) с условием Айверсона

В наилучшем случае (когда массив уже отсортирован по рассматриваемому в алгоритме правилу) вложенный цикл выполнится n-1 раз, а внешний – 1 раз. Таким образом, порядок роста для алгоритма в лучшем случае линейный. Т.е. $T(n)= \Omega(n)$.

В наихудшем случае массив упорядочен, но по правилу, обратному рассматриваемому в алгоритме. И вложенный, и внешний циклы будут задействованы. Т.е. $T(n)=O(n^2)$.

- Алгоритм Шейкерной сортировки с условием Айверсона

В наилучшем случае (когда массив уже отсортирован по рассматриваемому в алгоритме правилу) вложенные циклы выполнятся по n-1 раз, а внешний – 1 раз. Таким образом, порядок роста для алгоритма в лучшем случае линейный. Т.е. $T(n)= \Omega(n)$.

В наихудшем случае массив упорядочен, но по правилу, обратному рассматриваемому в алгоритме. И вложенный, и внешний циклы будут задействованы. Т.е. $T(n)=O(n^2)$.

- Алгоритм сортировки «Простое слияние»

В наилучшем и наихудшем случаях зависимость линейная, время выполнения программы не зависит от упорядоченности массива. Т.е. $T(n)=O(n)= \Omega(n)$.

Выводы:

Для лучшего случая алгоритм Шейкерной сортировки с условием Айверсона является наиболее эффективным.

Для худшего случая алгоритм сортировки «Прямое слияние» является наиболее эффективным.

Если рассматривать все три случая, то наиболее эффективным будет алгоритм сортировки «Прямое слияние».

2.3. Таблица для рассматриваемых в задании алгоритмов

Алгоритм	Асимптотическая сложность алгоритма			
	Наихудший случай	Наилучший случай	Средний случай	Емкостная сложность
«Пузырёк» с условием Айверсона	$O(n^2)$	O(n)	$O(n^2)$	O(n)
Шейкерная сортировка с условием Айверсона	$O(n^2)$	O(n)	$O(n^2)$	O(n)
«Прямое слияние»	O(n)	O(n)	O(n)	2* O(n)

ВЫВОДЫ

В ходе выполнения задания получены практические навыки в:

1. Разработке алгоритмов простой сортировки, усовершенствованной сортировки и слияния.
2. Оценке зависимости времени выполнения алгоритма от размера массива;
3. Оценке вычислительной сложности алгоритма сортировки в наихудшем и наилучшем случаях;
4. Проведении эмпирической (практической) оценки вычислительной сложности алгоритма;
5. Определении емкостной сложности алгоритма от n;
6. Оценке эффективности алгоритмов простых сортировок;
7. Поиске наиболее эффективного алгоритма.

Тестирования всех операций пройдены успешно.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Процедурное программирование Языки программирования – Сайт lizochekk! [Электронный ресурс]: URL: https://lizochekk.jimdofree.com/программирование/
2. Документация по Microsoft C/C++ | Microsoft Docs – [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/cpp/?view=msvc-160
3. C++ – Типизированный язык программирования / Хабр – [Электронный ресурс] URL: https://habr.com/ru/hub/cpp/
4. Сортировка прямым обменом (метод «пузырька») – [Электронный ресурс] URL: https://prog-cpp.ru/sort-bubble/
5. Сортировка слиянием – [Электронный ресурс] URL: https://prog-cpp.ru/sort-merge/