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

Отчет по заданию №1

«Методы сортировки»

Вариант 2 / 1 / 2 / 5

Выполнила:
студентка 102 группы
Анисимова Д. В.

Преподаватель:
Сенюкова О. В.

Москва
2020

Содержание

Постановка задачи

Требуется реализовать два метода сортировки: метод простого выбора и пирамидальную сортировку. Необходимо провести их экспериментальное сравнение и привести теоретические оценки. Сравение методов производится по количеству сравнений и обменов элементов сортируемого массива. Массив состоит из 64-разрядных целых чисел, их нужно упорядочить по неубыванию.

Результаты экспериментов

Сортировка методом простого выбора

n	**Параметр**	**Номер сгенерированного массива**				**Среднее значение**
		1	2	3	4	
10	Сравнения	45	45	45	45	45
	Перемещения	0	5	7	7	4,75
100	Сравнения	4950	4950	4950	4950	4950
	Перемещения	0	50	92	92	58,5
1000	Сравнения	499500	499500	499500	499500	499500
	Перемещения	0	500	994	992	621,5
10000	Сравнения	49995000	49995000	49995000	49995000	49995000
	Перемещения	0	5000	9989	9987	6244

Таблица 1: Результаты работы сортировки методом простого выбора

Оценка работы сортировки методом простого выбора

Количество сравнений не зависит от начального порядка элементов. На первой итерации внешнего цикла производятся $n-1$ сравнений, на второй - $n-2$ сравнений и т.д.; все элементы сравниваются попарно, следовательно, количество сравнений равно $\frac{n*(n-1)}{2}$.

Оценки количества обменов взяты из [1]. Обмен элементов происходит только во внешнем цикле, следовательно, число обменов не превосходит $n-1$ (худший случай). В лучшем случае (упорядоченный массив) количество обменов равно 0. Чтобы определить M_{avg} , будем рассуждать так: алгоритм просматривает массив, сравнивая каждый элемент с наименьшим значением, найденным до сих пор, и если элемент оказывается меньше, чем этот минимум, выполняется присваивание. Вероятность, что второй элемент меньше, чем первый, равна 1/2; такова же и вероятность присваивания минимуму нового значения. Вероятность того, что третий элемент окажется меньше, чем первые два, равна 1/3, а вероятность того, что четвертый окажется наименьшим, равна 1/4, и т. д. Поэтому полное среднее число пересылок равно H_{n-1} , где H_n – n-ое гармоническое число

$$H_n = 1 + 1/2 + 1/3 + ... + 1/n \tag{1}$$

H_n можно представить как

$$H_n = ln(n) + g + 1/2n - 1/12n^2 + ... \tag{2}$$

где $g = 0.577216...$ – константа Эйлера. Для достаточно больших n можно отбросить дробные члены и получить приближенное среднее число присваиваний на i-м проходе в виде

$$F_i = ln(i) + g + 1 \tag{3}$$

Тогда среднее число пересылок M_{avg} в сортировке выбором равно сумме величин F_i с i , пробегающим от 1 до n :

$$M_{avg} = n * (g + 1) + \sum_{i=1}^{n} ln(i) \tag{4}$$

Аппроксимируя дискретную сумму интегралом

$$\int_1^n ln(x)dx = n * ln(n) - n + 1 \tag{5}$$

получаем приближенное выражение

$$M_{avg} = n * (ln(n) + g) \tag{6}$$

Пирамидальная сортировка

n	Параметр	Номер сгенерированного массива				Среднее значение
		1	2	3	4	
10	Сравнения	39	28	32	35	33,5
	Перемещения	30	21	25	27	25,75
100	Сравнения	1075	883	1013	985	989
	Перемещения	640	516	594	578	582
1000	Сравнения	17479	15291	16499	16545	16453,5
	Перемещения	9708	8316	9086	9108	9054,5
10000	Сравнения	244129	220019	231941	231955	232011
	Перемещения	131956	116696	124223	124251	124281,5

Таблица 2: Таблица 2: Результаты работы пирамидальной сортировки

Оценка работы пирамидальной сортировки

1.Оценка количества сравнений.

Первый шаг: построение пирамиды. $\frac{n}{2}$ сравнений позволяют определить меньший ключ в каждой паре элементов, еще $\frac{n}{4}$ сравнений дадут меньший ключ в каждой паре из уже найденных меньших ключей, и т. д. Имея только $n-1$ сравнений, мы можем построить пирамиду, причем в корне будет искомый наименьший ключ.

Второй шаг: спуск по пути, соответствующему наименьшему ключу, и его удаление. Теперь элементом в корне дерева будет второй наименьший ключ, и его можно удалить. После n таких шагов выбора дерево станет пустым (то есть будет заполнено дырками), и процесс сортировки прекращается. Следует заметить, что на каждом из n шагов выбора требуется только $log(n)$ сравнений. Поэтому вся процедура требует лишь порядка $n*log(n)$ элементарных операций в дополнение к тем n шагам, которые нужны для построения дерева.

2. Оценка количества обменов

Фаза сортировки требует $n-1$ просеиваний, с не более чем $log(n-1)$, $log(n-2)$, ..., 1 пересылок. Кроме того, нужно $n-1$ пересылок для «складирования» элементов с вершины пирамиды в правом конце массива. Эти рассуждения показывают, что Heapsort требует порядка $n*log(n)$ пересылок даже в наихудшем случае. Отнюдь не ясно, в каких случаях следует ожидать наихудшей (или наилучшей) производительности. Фаза создания пирамиды не требует пересылок, если элементы изначально стоят в обратном порядке. Среднее число пересылок примерно равно $\frac{n*log(n)}{2}$, а отклонения от этого значения сравнительно малы [1].

Структура программы и спецификация функций

- `void make_mas(long long n, int p, long long *arr)`
  long long n - длина массива;
  int p - параметр, по которому будет строиться массив (при p == 1 происходит генерация упорядоченного массива, при p == 2 - обратно упорядоченного массива, при p == 3 - случайного массива);
  long long *arr - указатель на начало генерируемого массива.
  Ничего не возвращает.

- `void selection_sort(long long n, long long *buf)`
  long long n - длина массива;
  long long *buf - указатель на начало сортируемого массива.
  Функция сортирует переданный массив методом простого выбора. Ничего не возвращает.

- `void sift(long long left, long long right, long long *arr)`
  long long left - левая граница рассматриваемого подмассива;
  long long right - правая граница рассматриваемого подмассива;
  long long *arr - указатель на начало массива.
  Функция "просеивания". Рассматривает подмассив как пирамиду и "просеивает"через пирамиду ее корень, то есть из подмассива строится невозрастающая пирамида. Функция ничего не возвращает.

- `void heap_sort(long long n, long long *arr)`
  long long n - длина массива;
  long long *arr - указатель на начало сортируемого массива.
  Функция выполняет пирамидальную сортировку. Ничего не возвращает.

- `void print_mas(long long n, long long *arr)`
  long long n - длина массива;
  long long *arr - указатель на начало массива.
  Функция выводит элементы массива на стандартный поток вывода. Ничего не возвращает.

Отладка программы, тестирование функций

Тестирование сортировки методом простого выбора

	buf[0]	**buf[1]**	**buf[2]**	**buf[3]**	**buf[4]**	**buf[5]**	**buf[6]**	**buf[7]**	**buf[8]**	**buf[9]**
Ввод	0	1	2	3	4	5	6	7	8	9
Вывод	0	1	2	3	4	5	6	7	8	9

Таблица 3: Тест 1

	buf[0]	**buf[1]**	**buf[2]**	**buf[3]**	**buf[4]**	**buf[5]**	**buf[6]**	**buf[7]**	**buf[8]**	**buf[9]**
Ввод	10	9	8	7	6	5	4	3	2	1
Вывод	1	2	3	4	5	6	7	8	9	10

Таблица 4: Тест 2

	buf[0]	**buf[1]**	**buf[2]**	**buf[3]**	**buf[4]**	**buf[5]**	**buf[6]**	**buf[7]**	**buf[8]**	**buf[9]**
Ввод	361	-126	158	911	-132	63	787	828	88	637
Вывод	-132	-126	63	88	158	361	637	787	828	911

Таблица 5: Тест 3

Тестирование пирамидальной сортировки

	arr[0]	**arr[1]**	**arr[2]**	**arr[3]**	**arr[4]**	**arr[5]**	**arr[6]**	**arr[7]**	**arr[8]**	**arr[9]**
Ввод	0	1	2	3	4	5	6	7	8	9
Вывод	0	1	2	3	4	5	6	7	8	9

Таблица 6: Тест 1

	arr[0]	**arr[1]**	**arr[2]**	**arr[3]**	**arr[4]**	**arr[5]**	**arr[6]**	**arr[7]**	**arr[8]**	**arr[9]**
Ввод	10	9	8	7	6	5	4	3	2	1
Вывод	1	2	3	4	5	6	7	8	9	10

Таблица 7: Тест 2

	arr[0]	**arr[1]**	**arr[2]**	**arr[3]**	**arr[4]**	**arr[5]**	**arr[6]**	**arr[7]**	**arr[8]**	**arr[9]**
Ввод	361	-126	158	911	-132	63	787	828	88	637
Вывод	-132	-126	63	88	158	361	637	787	828	911

Таблица 8: Тест 3

Анализ допущенных ошибок

Ошибок не было.

Список литературы

[1] Вирт Н. Алгоритмы и структуры данных. — М.: Мир, 1989