**Пирамидальная сортировка**

Глухов Степан Андреевич, студент бакалавриата

Царёв Матвей Михайлович, студент бакалавриата

Московский государственный технический университет имени Николая Эрнестовича Баумана, г. Москва, Россия

***Аннотация:***

***Ключевые слова****: сортировка, алгоритмы сортировки, пирамидальная сортировка,*

**Пирамидальная сортировка**

Сортировка – фундаментальная задача из области вычислительной техники. Каждый человек, изучая тот или иной язык программирования, сталкивался с ней на начальных этапах обучения. Но алгоритмы сортировки используются не только в обучающих целях, но и в различных прикладных задачах, к примеру, для структуризации или визуализации информации.

Существует несколько алгоритмов сортировки, и у каждого из них есть свои достоинства и недостатки. Поэтому очень важно выбрать подходящий ещё на стадии составления алгоритма программы, чтобы не потратить лишнее время.

В данной статье будет рассмотрен алгоритм пирамидальной сортировки, основанный на такой структуре данных как пирамида или куча(heap).

Пирамида – массив A, который можно рассматривать как почти полное бинарное дерево. Каждый узел соответствует элементу массива. Все уровни, за исключением низшего, заполнены. Корень дерева – A[1]. Тогда для узла i можно легко найти родительский узел – , а также дочерние: левый – и правый – .

Во многих языках программирования нумерация элементов массива начинается с 0, поэтому вычисление дочерних узлов будет выглядеть иначе, но поиск родительского останется неизменным: , , .

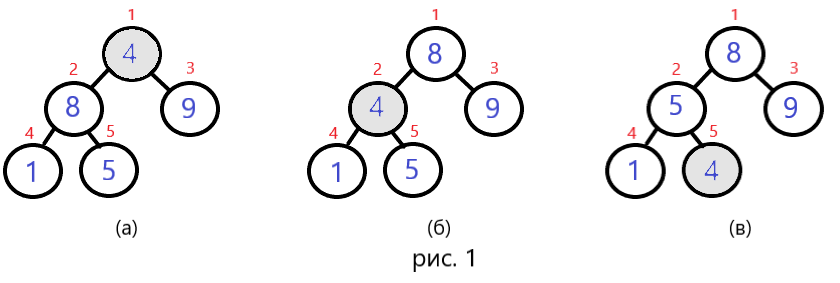
В алгоритме пирамидальной сортировки используется невозрастающая пирамида, то есть пирамида для каждого узла которой, не считая корневого, родительский узел будет иметь большее или равное значение: . Таким образом корневой узел будет наибольшим из всего массива.

Кроме невозрастающих существуют неубывающие пирамиды. В них корневой элемент будет наименьшим, а каждый родительский узел будет меньше или равен дочерним. Неубывающие пирамиды можно также использовать в пирамидальной сортировке, если требуется отсортировать массив по убыванию, а не по возрастанию.

Сам алгоритм пирамидальной сортировки делится на 2 этапа:

1. Построение пирамиды.

Для начала работы нужно сделать из неотсортированного массива невозрастающую пирамиду, для чего используется функция . В качестве аргументов она принимает указатель на массив, длину массива и номер выбранного элемента в массиве. Предназначение функции – опустить элемент на соответствующий ему уровень (рис.1).



Как можно увидеть из рис. 1, функция не строит пирамиду, а работает с конкретным элементом, сравнивая сначала с его левым, а затем правым дочерним элементом, и в случае, если не выполняется условие невозрастающей пирамиды (), опускает на один уровень ниже.

Далее, если элемент был передвинут, функция вызывает саму себя, но передаёт уже новый номер того же самого элемента. И так происходит до тех пор, пока он не встанет на своё место.

Функция, написанная на языке C:  
 **void** heapify(**int** \*arr, **int** n, **int** i) {

**int** largest = i;

**int** left = 2 \* i + 1;

**int** right = 2 \* i + 2;

**if** (left < n && arr[left] > arr[largest])

largest = left;

**if** (right < n && arr[right] > arr[largest])

largest = right;

**if** (largest != i){

SWAP(arr, i, largest);

heapify(arr, n, largest);

}

}

Но чтобы построить неубывающую пирамиду, надо каждый элемент массива поставить на соответствующее место. Если мы с помощью цикла начнём применять функцию с первого до последнего элемента, то сможем построить невозрастающую пирамиду. Но можно сократить количество действий: когда цикл дойдёт до первого элемента, не имеющего дочерних, его нужно будет остановить, т.к. все последующие также не имеют дочерних, а это значит, что их нельзя опустить на нужный уровень, ведь они уже на нём.

Чтобы остановить цикл, нужно найти родительский элемент для последнего элемента массива, т.к. он и будет последним элементом, у которого есть дочерние.

С такого построения пирамиды начинается основная функция пирамидальной сортировки . В качестве аргументов она принимает указатель на массив, а также его длину:

**void** heapSort(**int** \*arr, **int** n) {

**for** (**int** i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

**for** (**int** i = n - 1; i >= 0; i--) {

SWAP(arr, 0, i);

heapify(arr, i, 0);

}

}

Так как из обычного массива была получена невозрастающая пирамида, в корне, то есть на первом месте, оказался наибольший элемент. Дальше есть два пути: создать дополнительный массив и переносить элементы туда или работать в изначальном. В статье был выбран второй способ, поэтому обе функции принимают в качестве аргумента длину массива.

Первый(наибольший) элемент меняется с последним. Таким образом массив разбивается на две части: слева неотсортированный массив размера , а справа наибольший элемент.

Далее правая часть уже не трогается, а из левой за счёт функции , применённой к первому элементу, строится новая невозрастающая пирамида размера , в которую не входит последний элемент, т.к. в функцию в качестве размера массива было передано значение .

После того, как левая часть была переделана в невозрастающую пирамиду, на первом месте снова оказался наибольший элемент. Теперь он меняется местами с последним элементом массива, то есть встаёт на предпоследнее место. Таким образом в левой части остаётся элемента, а в правая, состоящая уже из 2 чисел, является частью отсортированной структуры данных.

Выполняя такое повторение раз, можно отсортировать массив по возрастанию. Чтобы лучше понять процесс рассмотрите рис.2:

**Временная и пространственная сложность**

Чтобы теоретически оценить временную сложность алгоритма пирамидальной сортировки, рассмотрим каждое действие, совершаемое им:

Функция – “на каждом шаге требуется воспроизвести θ(1) действий, не считая рекурсивного вызова”. T(n) – время работы функции для поддерева. В наихудшем случае, если нижний уровень заполнен только на половину, поддерево может содержать не более от общего количества элементов. То есть:

По основной теореме о рекуррентных соотношения . Соответственно временная сложность функции .

Время построения пирамиды менее , т.к. функция вызывается n раз. Но время работы зависит от высоты узла, поэтому реальная временная сложность - .

Время работы функции будет складываться из времени построения пирамиды , а также из времени вызова функции , то есть общая временная сложность алгоритма будет составлять.

На рис. 3 представлено сравнение теоретической и практической времени работы пирамидальной сортировки:

Пространственная сложность зависит выбора одного из двух путей, описанных выше: если для отсортированной последовательности чисел создаётся дополнительный массив, то она будет , а если всё происходит внутри одной структуры данных, то .

**Преимущества и недостатки**

Лучше всего можно понять плюсы и минусы алгоритма пирамидальной сортировки на сравнении его с другими алгоритмами. В данной статье он будет сравниваться с алгоритмами: быстрой сортировки, сортировки Шелла, обменной сортировки.

Для каждого графика время работы было выбрано как среднее из 50 проведённых измерений для каждой точки, чтобы уменьшить случайную погрешность.

Для начала рассмотрим график работы выбранных алгоритмов сортировки на примере массива из случайно выбранных чисел.

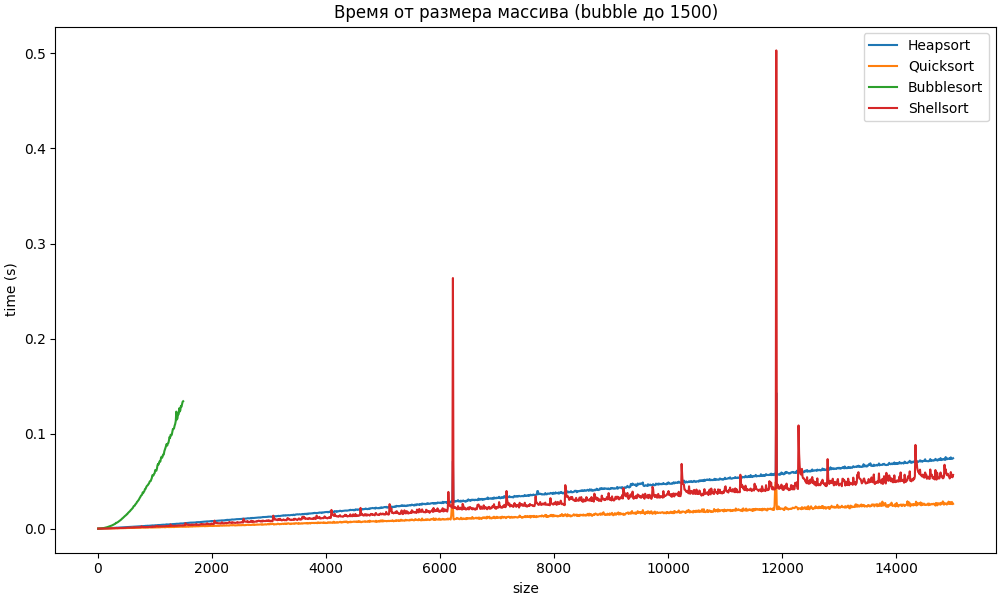


Рис. 1

Как видно из рис. 1 видно, что пузырьковая(обменная) сортировка является наиболее долгой, теоретическое время работы данного алгоритма . Чтобы другие алгоритмы не сливались в сплошную линию, график изображён только да значения n=1500.

Теоретическое время работы оставшихся алгоритмов . Чтобы лучше продемонстрировать различие во времени между ними, рассмотрим рис. 2:

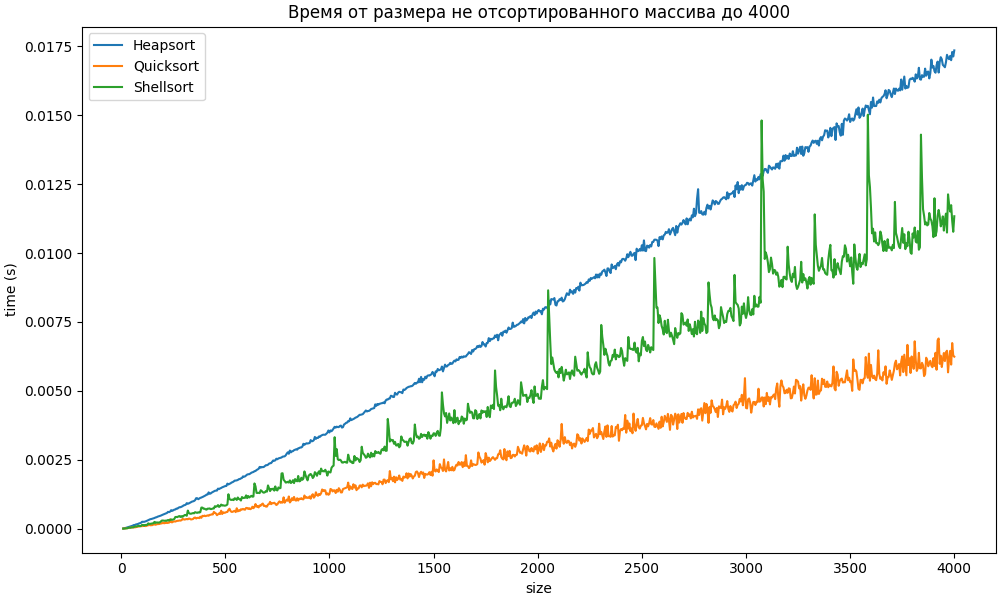


Рис. 2

По нему можно определить, что наименее быстрой сортировка является пирамидальная, а наибольшая скорость у быстрой. Но также можно заметить, что пирамидальная является наиболее стабильной на всём своём графике.

Если же взглянуть на рис. 1, то можно увидеть, что сортировка Шелла наименее стабильна из всех, поэтому хоть в большинстве значений она быстрее пирамидальной, но на некоторых вариантах случайных массивов время её работы будет в разы превышать время работы пирамидального алгоритма.