Пирамидальная сортировка: преимущества и недостатки алгоритма

**Глухов С. А.**(1), студент

**Царёв М. М.**(1), студент

**Карышев Б. В.**(1), студент

**Казаков В. В.**(1), аспирант

(1)Кафедра ИУ4, МГТУ им. Н.Э. Баумана, г. Москва

**Аннотация:** Сортировка – одна из наиболее часто реализуемых в различных программах задач, которая имеет большое количество решений в виде различных алгоритмов сортировки, каждый из которых имеет свои преимущества и недостатки. В данной статье рассмотрена реализация алгоритма пирамидальной сортировки на основе создания из массива данных невозрастающей пирамиды и дальнейших операций с ней. Проведено сравнение его эффективности с другими алгоритмами сортировки на основе полученных данных о времени выполнения своей задачи для различных входных данных каждого алгоритма. На основе сравнения выявлены основные преимущества и недостатки, присущие данному алгоритму, и сделаны рекомендации о его применении при различных входных структурах данных.

**Ключевые слова:** сортировка, алгоритмы сортировки, пирамидальная сортировка, алгоритм пирамидальной сортировки.

**Пирамидальная сортировка**

В наше время всё чаще появляется необходимость в сборе, хранении и анализе больших объёмов данных различной природы для чего требуются значительные вычислительные мощности. Однако обработку данных можно значительно ускорить, если предварительно их упорядочить по определённым критериям, например, для численных значений по возрастанию. Алгоритм, выполняющий упорядочивание данных, называется алгоритмом сортировки.

На текущий момент существует большое количество различных алгоритмов сортировки, и у каждого из них есть свои достоинства и недостатки. Поэтому очень важно выбрать подходящий ещё на стадии разработки структуры программы.

В данной статье будет рассмотрен алгоритм пирамидальной сортировки, основанный структуре данных, которая называется пирамида или куча (heap).

Пирамида представляет собой массив , который можно рассматривать как почти полное бинарное дерево. Каждый узел соответствует элементу массива. Все уровни, за исключением низшего, заполнены. Корень дерева – . Тогда для узла под индексом можно легко найти индекс родительского узла– , а также индексы дочерних: левого – и правого – .

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

В основной реализации алгоритма 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.

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.

Алгоритм пирамидальной состоит из 2 этапов:

1. Построение пирамиды.

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

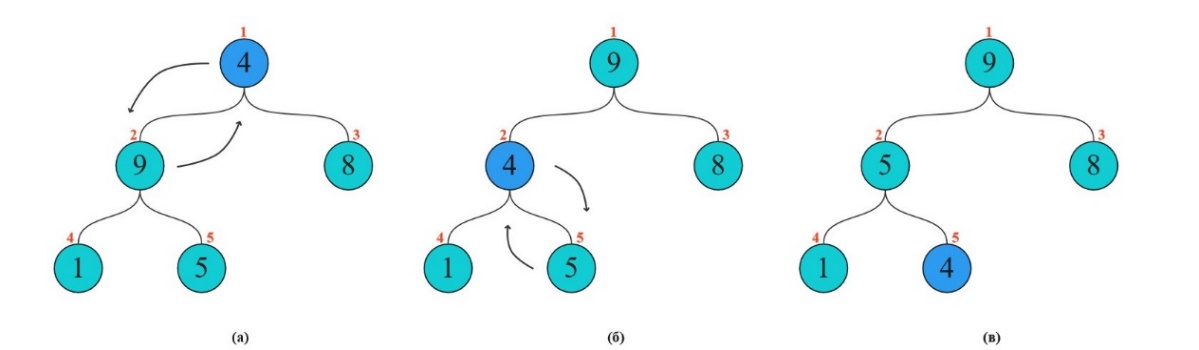


Рисунок 1. Построение пирамиды

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 (), функция перемещает текущий элемент на один уровень ниже. Далее, если положение элемента изменилось, функция рекурсивно вызывает саму себя, но передаёт новый индекс того же элемента, чтобы проверить, удовлетворяет ли его положение правилам построения пирамиды. В случае если элемент нарушает правила построения пирамиды, то функция вновь перемещает его и вновь вызывает саму себя.

Реализующая данный алгоритм 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);

}

}

Для построения неубывающей пирамиды, надо каждый элемент массива переместить на соответствующую позицию. Если последовательно применить функцию для всех элементов массива, то он будет преобразован в невозрастающую пирамиду, но количество действий может быть сокращено: когда цикл обратится к первому элементу, не имеющему дочерних, его дальнейшее выполнение не имеет смысла, так как все последующие элементы также не имеют дочерних, а это значит, что они уже находятся на требуемых уровнях пирамиды.

Последний элемент, который имеет дочерние, — это родительский элемент для последнего элемента массива, поэтому, чтобы ускорить работу алгоритма за счёт уменьшения действий по построению пирамиды, нужно выполнять цикл до элемента с индексом .

С описанного выше построения пирамиды начинается основная функция пирамидальной сортировки :

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);

}

}

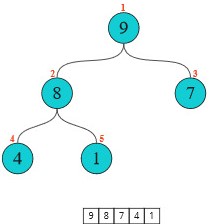
 Так как из обычного массива была получена невозрастающая пирамида (рис. 2), в корне оказался наибольший элемент. Для дальнейшей сортировки пирамиды можно создать дополнительный массив и переносить туда элементы или работать в изначальном. В данной работе рассматривалась реализация алгоритма без выделения дополнительной памяти.

Рисунок 2. Невозрастающая пирамида.

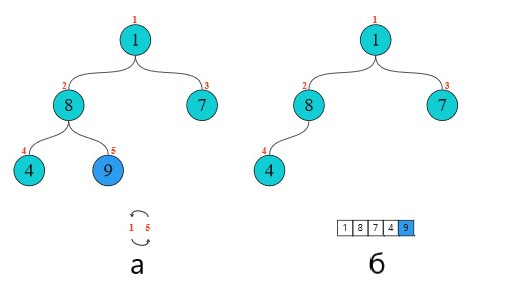


Рисунок 3. Последовательность сортировки.

Первый элемент меняется с последним (рис. 3, а). Таким образом массив разбивается на две части: неотсортированный массив размера и наибольший элемент (рис. 3, б). Далее часть, содержащая наибольший элемент, остаётся неизменной, а из второй части за счёт функции , применённой к первому элементу, строится новая невозрастающая пирамида размера , в которую не входит последний элемент.

Повторив вышеуказанные действия раз получим отсортированный массив(дальнейшие действия сортировки показаны на рис. 4).

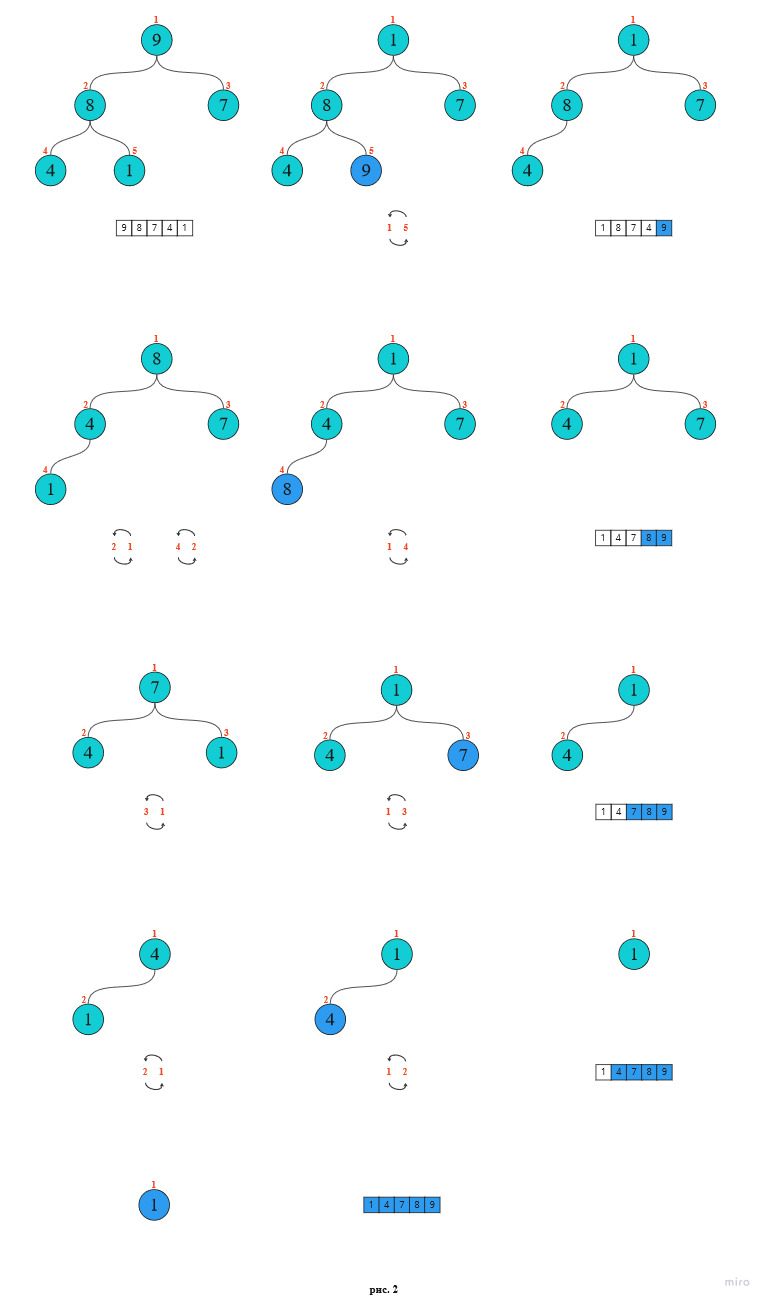


Рисунок 4. Последовательность сортировки.

**Временная и пространственная сложность**

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

Функция – “на каждом шаге требуется воспроизвести θ(1) действий, не считая рекурсивного вызова”.[2] 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 составляет .

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

Время работы функции будет складываться из времени построения пирамиды , а также из времени вызовов функции , то есть общая временная сложность алгоритма будет составлять. При этом временная сложность для худшего и наилучшего случая одинакова, как и для среднего, и равна . Однако если в лучшем случае будут представлены одинаковые ключи-значения, то временная сложность составляет .

На рис. 5 представлено сравнение теоретической и практической времени работы пирамидальной сортировки:

Рисунок 5. Временная сложность алгоритма пирамидальной сортировки.

Пространственная сложность зависит выбора одного из двух варианта реализации алгоритма сортировки: если для отсортированной последовательности создаётся дополнительный массив, то она составляет , а если всё происходит внутри одной структуры данных, то .

**Преимущества и недостатки**

Лучше всего можно выявить преимущества и недостатки алгоритма пирамидальной сортировки при сравнении его с другими алгоритмами. В данной работе проводилось его сравнение с алгоритмами быстрой сортировки, сортировки Шелла и пузырьковой (обменной) сортировки.

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

Рассмотрим эффективность вышеуказанных алгоритмов сортировки при различных вариантах изначального расположения и характера элементов: расположенных в случайном порядке, упорядоченных в обратном порядке, наполовину упорядоченных, равных и однообразных элементов.

1. Определим время работы выбранных алгоритмов сортировки для массива случайно расположенных разнообразных чисел (рис. 6).

Рисунок 6. Время сортировки массива случайных чисел.

Из рис. 6 видно, что пузырьковая (обменная) сортировка является наименее эффективной при большом количестве элементов, что объясняется тем, что её теоретическая временная сложность равна . Чтобы графики других алгоритмов были различимы, график пузырьковой сортировки изображён только до количества элементов равного 1500. Средняя теоретическая временная сложность остальных алгоритмов составляет . При этом пирамидальная сортировка обладает наименьшей эффективностью, а быстрая сортировка – наибольшей. Однако, пирамидальная сортировка является более стабильной, чем сортировка Шелла, время которой может меняться в зависимости от исходного массива и, местами, превышать время пирамидальной сортировки.

1. Рассмотрим эффективность алгоритмов при сортировке массивов, упорядоченных в обратном порядке (рис. 7). Такую структуру данных эффективнее развернуть, то есть менять соответствующие элементы местами, а не применять алгоритмы сортировки, но для её упорядочивания их использование обосновано, если структура заранее не известна.

Рисунок 7. Время сортировки массива, упорядоченного в обратном порядке.

По рис. 7 видно, что пирамидальная сортировка сортирует упорядоченный в обратном порядке массив за меньшее время, чем быстрая и обменная сортировки. При сортировке данной структуры массива алгоритм Шелла обладает наибольшей эффективностью среди исследованных алгоритмов. Также видно, что в данном случае сортировка Шелла ведет себя стабильнее, чем Пирамидальная сортировка, которая имеет 2 аномальные временные вершины на графике.

1. Рассмотрим эффективность алгоритмов сортировки, упорядочивающих массив, состоящий из равных друг другу элементов (рис. 8).

Рисунок 8. Время сортировки массива, состоящего из равных элементов для всех представленных сортировок.

Как видно по рис. 8 по эффективности быстрая сортировка значительно уступает другим алгоритмам. Чтобы лучше увидеть, как они отличаются друг от друга рассмотрим график зависимости времени от величины массива для сортировок, исключая быструю сортировку (рис. 9):

Рисунок 9. Время сортировки массива, состоящего из равных элементов для всех представленных сортировок, кроме быстрой сортировки.

Из рис. 9 видно, что при равных элементах пузырьковая сортировка справляется значительно лучше остальных рассмотренных алгоритмов, в то время как пирамидальная сортировка немного ей уступает. Данный результат можно объяснить количеством проводимых операций. Так пузырьковой сортировке, использовавшейся в данной работе, нужно всего лишь один раз пройти весь массив и провести сравнения соседних элементов. После одного прохода она возвращает исходный массив без изменений. В таком случае все временная сложность пузырьковой сортировки составляет . В то время как сортировки Шелла и Пирамидальная имеют больше проводимых операций, в том числе и замена равных элементов. Сравнивая же сортировку Шелла и Пирамидальную сортировку между собой, первая показывает себя хуже, чем вторая. Сортировка Шелла имеет большее время сортировки и больше подвержена факторам, связанным с работой компьютера., то есть временные скачки, наблюдавшиеся одновременно у сортировок Шелла и Пирамидальной, имеют большую амплитуду.

1. Рассмотрим структуру массива, состоящего из 3 повторяющихся элементов. (рис. 10)

Рисунок 10. Время сортировки массива, состоящего из трёх повторяющихся элементов.

В данном случае самой медленной закономерно становится пузырьковая сортировка, а пирамидальную превосходит сортировка Шелла.

Также на примере наборов данных, состоящих из одного и трех элементов, можно увидеть один из главных недостатков алгоритма пирамидальной сортировки – изменение относительного расположения равных элементов. Так как алгоритм строит бинарное дерево для вычисления расположения каждого элемента в отсортированном массиве, равные элементы перемешиваются и их положение относительно друг друга в итоговой структуре данных не будет совпадать с первоначальным. При работе с числами это не имеет никаких последствий, но в случае более сложных структур эта особенность может быть причиной для отказа от применения алгоритма пирамидальной сортировки.

1. Рассмотрим случай, когда к отсортированному массиву добавили произвольные данные. Назовём такую структуру – частично упорядоченный массив (рис. 11).

Рисунок 11. Время сортировки частично упорядоченного массива.

**Заключение**

На основе проведённых экспериментов определить основные преимущества и недостатки пирамидального алгоритма сортировки:

Стабильность. Пузырьковая сортировка одинаково хорошо справляется с упорядочиванием любых вариантов расположения исходных данных. Не было выявлено структуры, на которой временная сложность пирамидальной сортировки значительно изменяется, как, например, у пузырьковой сортировки, имеющей в общем случае временную сложность , которая при сортировке упорядоченного массива падает до , или Быструю сортировку, которая имеет в худших вариантах.

Скорость. При том, что ни в одном из рассмотренных случаях пирамидальная сортировка не являлась самой быстрой, её эффективность не сильно отставала от первого места. Поэтому, учитывая стабильность алгоритма, для многих проектов пирамидальную сортировку можно считать оптимальной.

Перемешивание равных элементов. Пирамидальная сортировка меняет относительное расположение равных между собой элементов. Данный недостаток проявляет себя только при сортировке элементов сложной природы, однако он может оказать решающее значение и привести к невозможности применения данного алгоритма.

Сложность. Пирамидальная сортировка относительно сложна в реализации и понимании, поэтому в случае необходимости самостоятельной реализации алгоритма, выбор пирамидальной сортировки будет обоснован исключительно в случае наличия потребности в высокой скорости выполнения, которая обычно возникает при большом количестве элементов.

Алгоритм пирамидальной сортировки является одним из наиболее стабильных и быстрых алгоритмов, который может использоваться в проектах, в которых требуется сортировка различных видов и частных случаев массивов.

Список литературы

1. Искусство программирования для ЭВМ / Кнут Д. Т. 3 : Сортировка и поиск / пер. с англ. Вьюкова Н. И., Галатенко В. А., Ходулев А. Б. ; ред. пер. с англ. Баяковский Ю. М., Штаркман В. С. - 1978. - 843 с.
2. [Левитин А. В.](https://www.wikidata.org/wiki/Q21694518) / Алгоритмы: введение в разработку и анализ  — М.: [Вильямс](https://www.wikidata.org/wiki/Q21694521), 2006. — С. 275—284.
3. Седжвик Р. Фундаментальные алгоритмы на C. Ч. 1-4. Анализ. Структуры данных. Сортировка. Поиск. - СПб.: ДиаСофтЮП, 2003. - 672 с.
4. Кормен Т., Лайзерсон Ч., Ривест Р. Сортировка с помощью кучи // Кормен Т., Лайзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ: пер. с англ. - М.: МЦНМО, 2002. - Гл. 7. - С. 138-150.
5. Иванов Н. О. АСИНХРОННАЯ ПИРАМИДАЛЬНАЯ СОРТИРОВКА / Н. О. Иванов, Д. О. Тингайкин. — Текст : непосредственный // Сборник научных трудов Новосибирского государственного технического университета — Новосибирск : Новосибирский государственный технический университет, 2016. — С. 98-106.