Северо-Осетинский государственный университет имени К. Л. Хетагурова

Факультет математики и информационных технологий.
Прикладная математика и информатики.

Курсовая работа

По дисциплине: «Практикум на ПК»

Тема: «Пирамидальная сортировка»

(Сортировка двоичной кучей)

Выполнил студент первого курса

Гамосов Станислав

Владикавказ 2019 г.

Оглавление.

Пирамида и ее свойства.

Различают два вида бинарных пирамид: ***неубывающие и невозрастающие***. В пирамидах обоих видов значения, расположенные в узлах, удовлетворяют ***свойству пирамиды***, являющемуся отличительной чертой пирамиды того или иного вида.

Свойство невозрастающих пирамид заключается в том, что для каждого отличного от корневого узла с индексом i выполняется следующее неравенство:

$$A[i] \geq A[2i + 1] \,\&\, A[i] \geq A[2i + 2]$$

Другими словами, значение узла не превышает значение родительского по отношению к нему узла. Таким образом, в невозрастающей пирамиде самый большой элемент находится в корне дерева, а значения узлов поддерева, берущего начало в каком-то элементе, не превышают значения самого этого элемента.

Принцип организации ***неубывающей пирамиды*** прямо противоположный. ***Свойство неубывающих пирамид*** заключается в том, что для всех отличных от корневого узлов с индексом i выполняется такое неравенство:

$$A[i] \leq A[2i + 1] \,\&\, A[i] \leq A[2i + 2]$$

Таким образом, наименьший элемент такой пирамиды находится в ее корне. В алгоритме пирамидальной сортировки используются ***невозрастающие пирамиды***.

Неубывающие пирамиды часто применяются в приоритетных очередях. Для каждого приложения будет указано, с пирамидами какого вида мы будем иметь дело — с неубывающими или невозрастающими. При описании свойств как неубывающих, так и невозрастающих пирамид будет использоваться общий термин ***"пирамида".***

Рассматривая пирамиду как дерево, определим высоту ее узла как число ребер в самом длинном простом нисходящем пути от этого узла к какому-то из листьев дерева. Высота пирамиды определяется как высота его корня. Поскольку n-элементная пирамида строится по принципу полного бинарного дерева, то ее высота равна $\theta(\log n)$. Мы сможем убедиться, что время выполнения основных операций в пирамиде приблизительно пропорционально высоте дерева, и, таким образом, эти операции требуют для работы время $\theta(\log n)$.

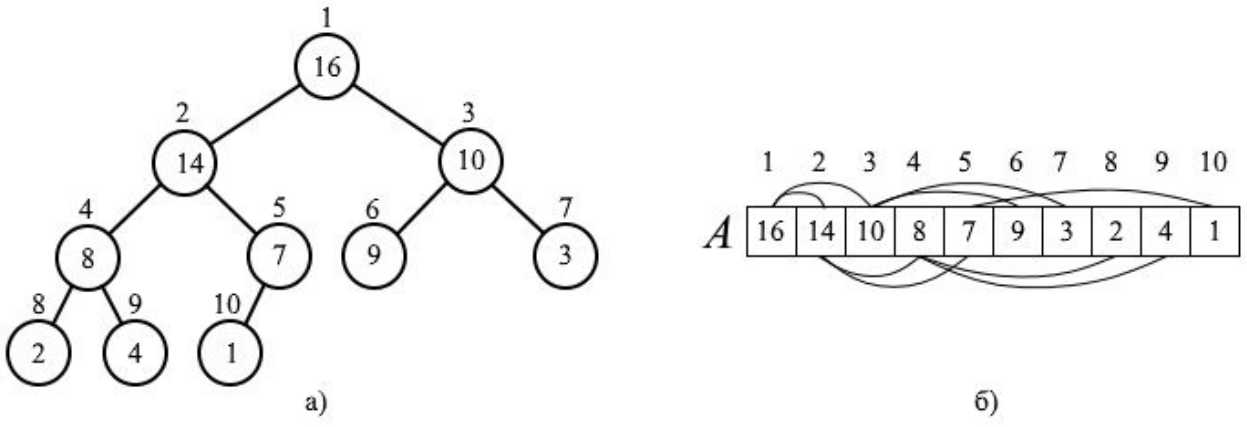


Рис. 1. Пирамида, представленная в виде а) *бинарного дерева* и б) *массива*

Поддержка свойства пирамиды.

Процедура ***shiftDown*** является важной подпрограммой, предназначенной для работы с элементами невозрастающих пирамид. На ее вход подается массив A и индекс i этого массива. При вызове процедуры ***shiftDown*** предполагается, что бинарные деревья, корнями которых являются элементы $left$ и $right$, являются ***невозрастающими пирамидами***, но сам элемент $A[i]$ может быть меньше своих дочерних элементов, нарушая тем самым свойство невозрастающей пирамиды. Функция ***shiftDown*** опускает значение элемента $A[i]$ вниз по пирамиде до тех пор, пока поддерево с корнем, отвечающем индексу i, не становится ***невозрастающей пирамидой***.

Действие процедуры ***shiftDown*** проиллюстрировано на рис. 2. На каждом этапе ее работы определяется, какой из элементов — $A[i]$, $A[left]$ или $A[rigth]$ — является максимальным, и его индекс присваивается переменной $size$. Если наибольший элемент — $A[i]$, то поддерево, корень которого находится в узле с индексом i, — невозрастающая пирамида, и процедура завершает свою работу. В противном случае максимальное значение имеет один из дочерних элементов, и элемент $A[i]$ меняется местами с элементом $A[size]$. После этого узел с индексом i и его дочерние узлы станут удовлетворять свойству невозрастающей пирамиды. Однако теперь исходное значение элемента $A[i]$ присвоено элементу с индексом $size$, и свойство невозрастающей пирамиды может нарушиться в поддереве с этим корнем. Для устранения нарушения для этого дерева необходимо рекурсивно вызвать процедуру ***shiftDown***.

На рис. 2 показана работа процедуры ***shiftDown***. В части а этого рисунка показана начальная конфигурация, в которой элемент $A[2]$ нарушает свойство невозрастающей пирамиды, поскольку он меньше, чем оба дочерних узла. Поменяв местами элементы $A[2]$ и $A[4]$, мы восстанавливаем это свойство в узле 2, но нарушаем его в узле 4 (часть б рисунка). Теперь в рекурсивном вызове процедуры $shiftDown(A, size, 4)$ в качестве параметра выступает значение $i = 4$. После перестановки элементов A[4] и A[9] (часть в рисунка) ситуация в узле 4 исправляется, а рекурсивный вызов процедуры $shiftDown(A, size, 9)$ не вносит никаких изменений в рассматриваемую структуру данных.

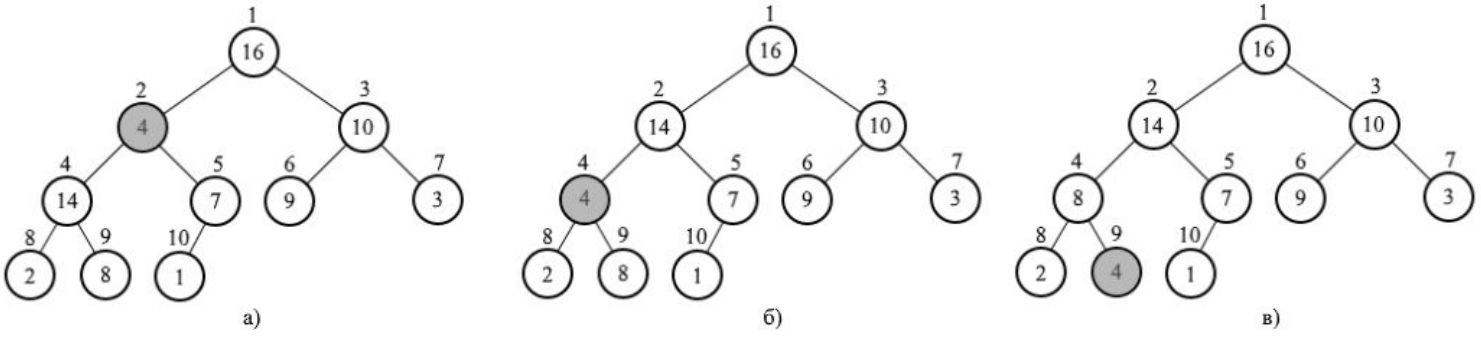
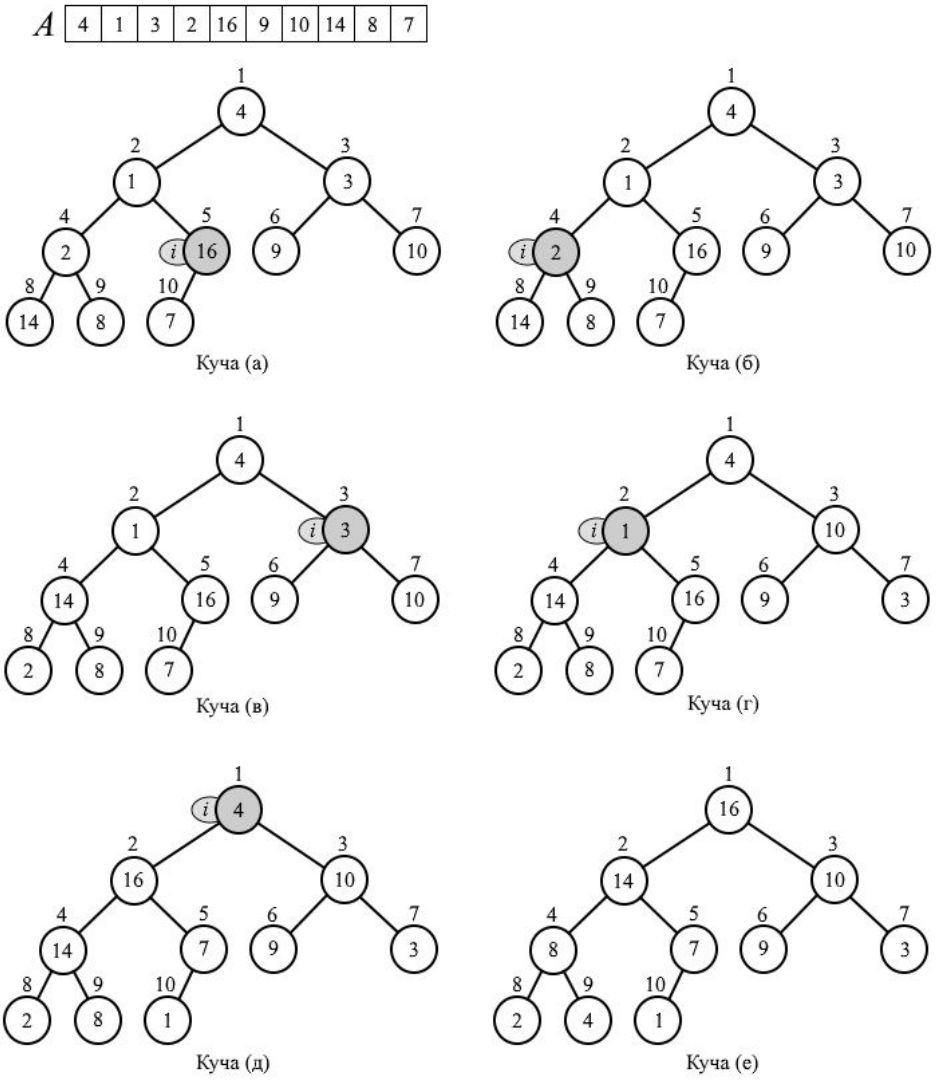


Рис. 2. Работа процедуры s*hiftDown*(A, $size$, 2) при size = 10

Создание пирамиды.

С помощью процедуры ***bildHeap*** можно преобразовать массив $A[1...n]$, где n - размер массива A, в ***невозрастающую пирамиду*** в направлении снизу вверх. Известно, что все элементы подмассива $A[(n/2 + 1) ... n]$ являются ***листьями дерева***, поэтому каждый из них можно считать ***одноэлементной пирамидой***, с которой можно начать процесс построения. Процедура ***bildHeap*** проходит по остальным узлам и для каждого из них выполняет процедуру ***siftDown***:

Пример работы процедуры ***bildHeap*** показан на рис. 3: В ***части а*** этого рисунка изображен 10-элементный входной массив A и представляющее его ***бинарное дерево***. Из этого рисунка видно, что перед вызовом процедуры ***siftDown*** индекс цикла i указывает на 5-й узел. Получившаяся в результате структура данных показана в ***части б***. В следующей итерации индекс цикла i указывает на узел 4. Последующие итерации цикла for в процедуре ***bildHeap*** показаны в ***частях от в до д*** рисунка. Обратите внимание, что при вызове процедуры ***siftDown*** для любого узла поддеревья с корнями в его дочерних узлах являются невозрастающими пирамидами. В



части е показана невозрастающая пирамида, полученная в результате работы процедуры ***bildHeap***.

Рис. 3. Схема работы процедуры *bildHeap*

Простую верхнюю оценку времени работы процедуры ***bildHeap*** можно получить следующим простым способом. Каждый вызов процедуры ***siftDown*** занимает время $\theta(log\ n)$, и всего имеется $\theta(n)$ таких вызовов. Таким образом, время работы алгоритма равно $\theta(n\ log\ n)$. Эта верхняя граница вполне корректна, однако не является асимптотически точной.

Словесное описание алгоритма Пирамидальной сортировки.

Работа алгоритма пирамидальной сортировки начинается с вызова процедуры ***bildHeap***, с помощью которой из входного массива $A[1 \ldots n]$, где n - размер массива, создается ***невозрастающая пирамида***. Поскольку наибольший элемент массива находится в ***корне***, т.е. в элементе $A[1]$, его можно поместить в окончательную позицию в отсортированном массиве, поменяв его местами с элементом $A[n]$. Выбросив из пирамиды узел n (*путем уменьшения на единицу величины размера пирамиды*), мы обнаружим, что подмассив $A[1 \ldots n]$ легко преобразуется в ***невозрастающую пирамиду***. Пирамиды, дочерние по отношению к корневому узлу, после обмена элементов $A[1]$ и $A[n]$ и уменьшения размера пирамиды остаются ***невозрастающими***, однако новый корневой элемент может нарушить свойство невозрастания пирамиды. Для восстановления этого свойства достаточно вызвать процедуру ***shiftDown***, после чего подмассив $A[1 \ldots n]$ превратится в ***невозрастающую пирамиду***. Затем алгоритм пирамидальной сортировки повторяет описанный процесс для невозрастающих пирамид размера $n - 1, n - 2, \ldots, 2$. *Пусть дана последовательность из 5 элементов 3,2,4,1,5*

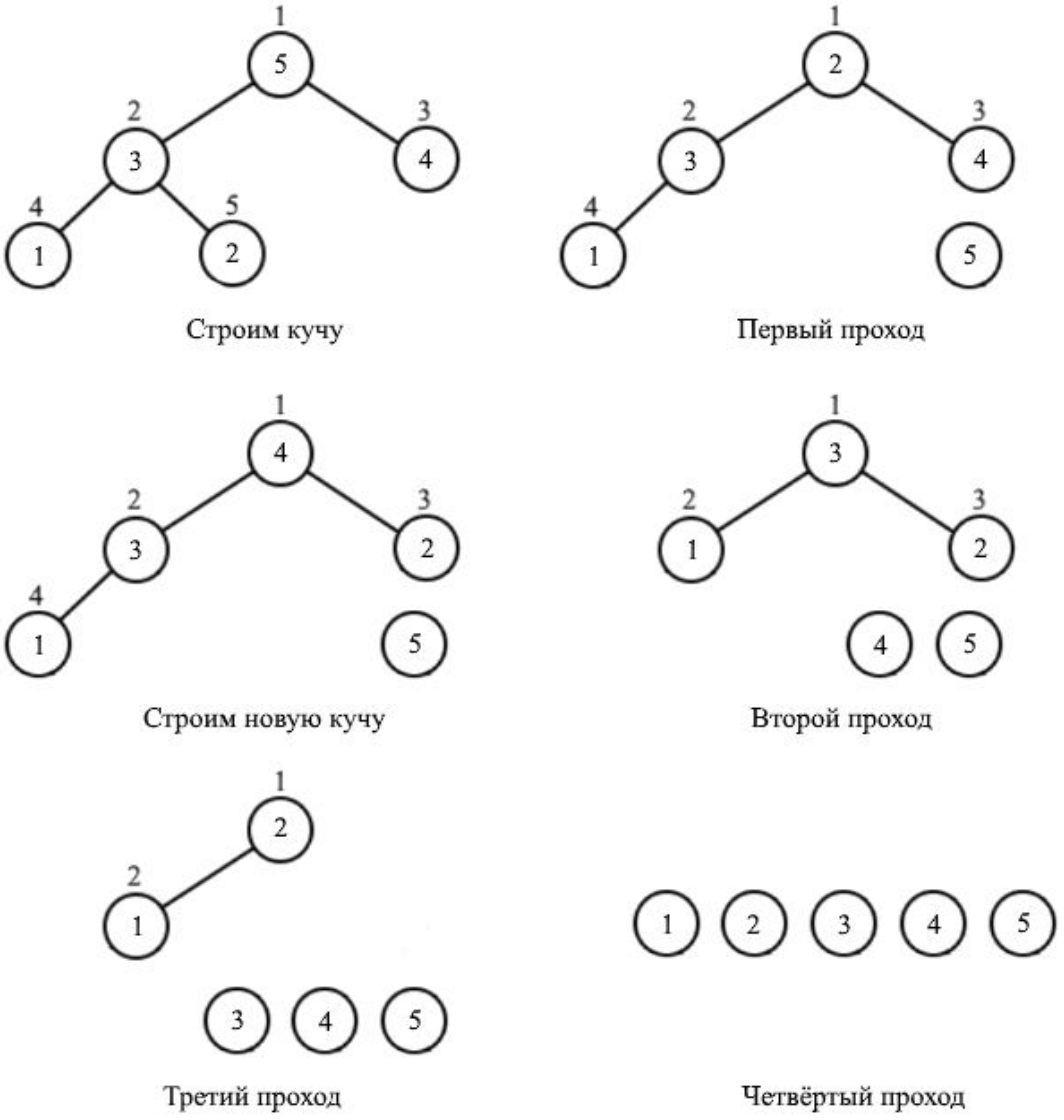
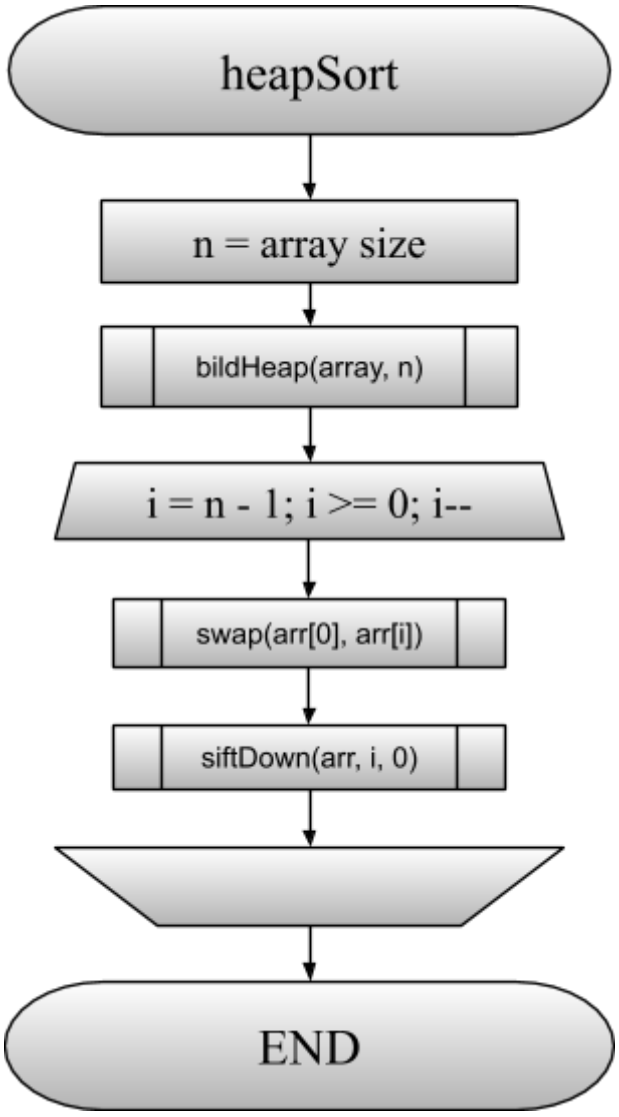
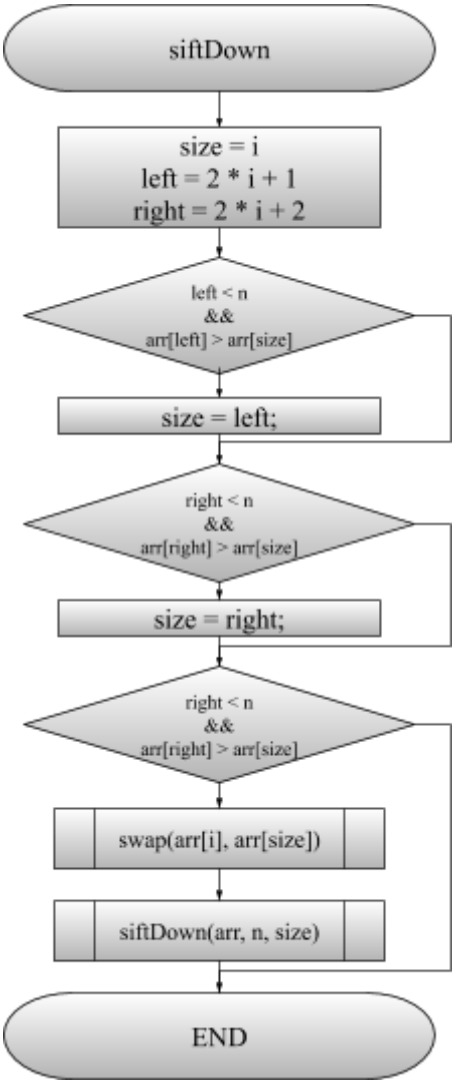
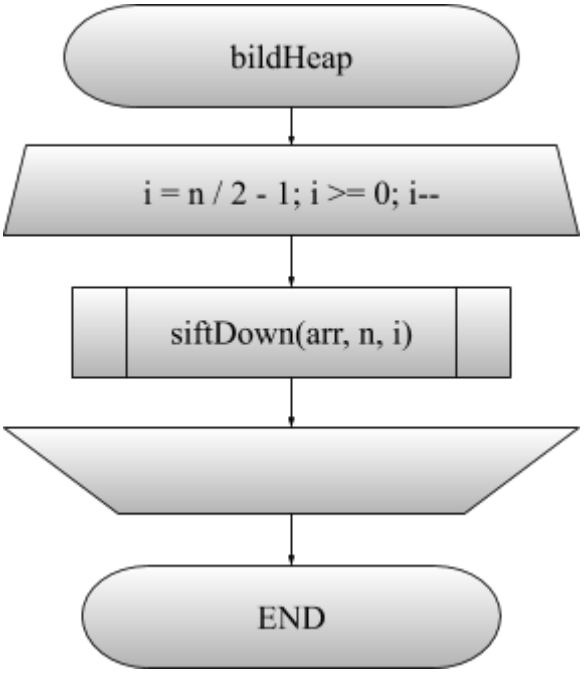


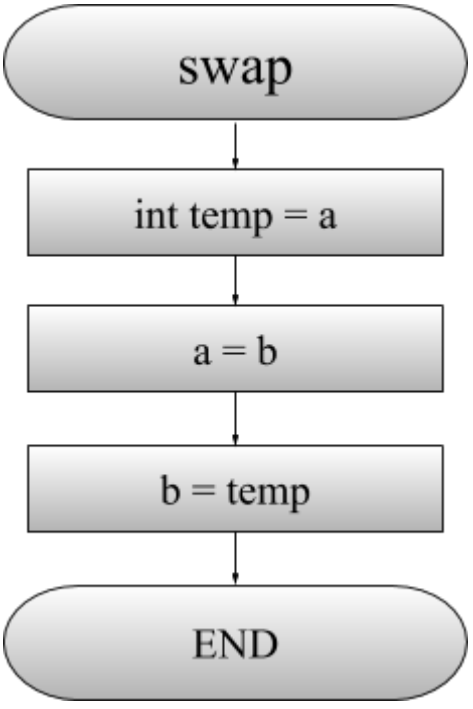
Рис. 4. Графическое представление сортировки.

Описание алгоритма сортировки в виде блок-схемы.









Сложность.

Операция siftDown работает за $\theta(log\ n)$. Всего цикл выполняется $(n - 1)$(раз. Таким образом сложность сортировки кучей является $\theta(n\ log\ n)$

Достоинства:

- худшее время работы — $\theta(n\ log\ n)$.
- требует $\theta(1)$ дополнительной памяти.

Недостатки:

- Неустойчив.
- На почти отсортированных массивах работает столь же долго, как и на хаотических данных.

Применение.

Алгоритм «сортировка кучей» активно применяется в ядре Linux.

Описание работы алгоритма на примере.

(Таблица трассировки)

Пусть дана последовательность из *5* элементов *3,2,4,1,5.*

Массив	Описание шага
5 3 4 1 2	Строим кучу из исходного массива
Первый проход	
2 3 4 1 **5**	Меняем местами первый и последний элементы
4 3 2 1 5	Строим кучу из первых четырёх элементов
Второй проход	
1 3 2 **4** 5	Меняем местами первый и четвертый элементы
3 1 2 4 5	Строим кучу из первых трёх элементов
Третий проход	
2 1 **3** 4 5	Меняем местами первый и третий элементы
2 1 3 4 5	Строим кучу из двух элементов
Четвертый проход	
1 **2** 3 4 5	Меняем местами первый и второй элементы
1 2 3 4 5	Массив отсортирован

Рис. 5. Таблица трассировки.

Входные данные.

Последовательность из целых чисел.

Выходные данные.

Отсортированная последовательность по неубыванию.

Код сортировки.

```
static void heapSort(ref int[] arr)
{
	int n = arr.Length;
	bildHeap(ref arr, n);

	for (int i = n - 1; i >= 0; i--)
	{
		swap(ref arr[0], ref arr[i]);
		siftDown(arr, i, 0);
	}
}

static void siftDown(int[] arr, int n, int i)
{
	int size = i;
	int left = 2 * i + 1;
	int right = 2 * i + 2;

	if (left < n && arr[left] > arr[size])
		size = left;
	if (right < n && arr[right] > arr[size])
		size = right;

	if (size != i)
	{
		swap(ref arr[i], ref arr[size]);
		siftDown(arr, n, size);
	}
}

static void bildHeap(ref int[] arr, int n)
{
	for (int i = n / 2 - 1; i >= 0; i--)
		siftDown(arr, n, i);
}
```