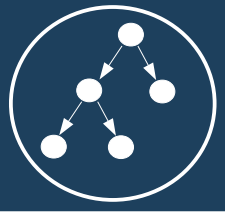
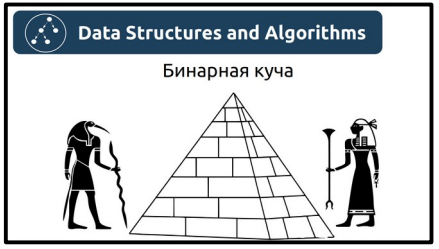


# Data Structures and Algorithms

## Пирамидальная сортировка

# Список лекций необходимых для занятия

Перед просмотром этого занятия нужно просмотреть следующие лекции.




Бинарная куча

# Сложность алгоритма

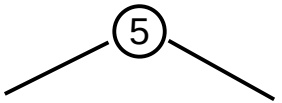
Вычислительная сложность $O(n \cdot \ln(n))$

# Алгоритм

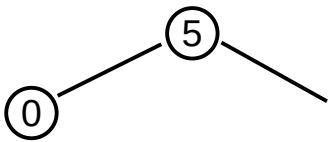
1) Создать бинарную кучу на основе массива. Для этого можно использовать метод восстановления свойств кучи для каждого элемента массива. Массив делиться на сортированную и не сортированную часть. Сортированная часть — правая. В начале алгоритма ее длина равна 0. Перейти к пункту **2**.
2) Провести обмен первого элемента (а он максимум) и последнего в не сортированной части. Увеличить отсортированную часть на единицу. Провести просеивание вниз на не сортированной части начиная с первого элемента. Перейти к пункту **3**.
3) Если длина отсортированной части равна длине массива, то **алгоритм закончен**, в противном случае перейти к пункту **2**.

# Графическая иллюстрация работы алгоритма

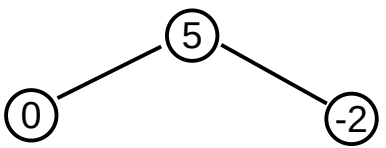
[5, 0, -2, 7, 3]



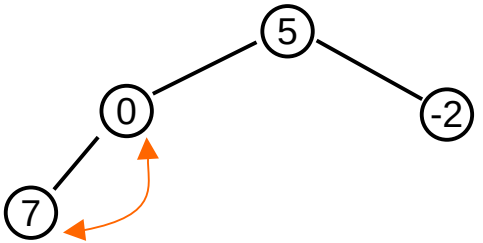

[5, 0, -2, 7, 3]



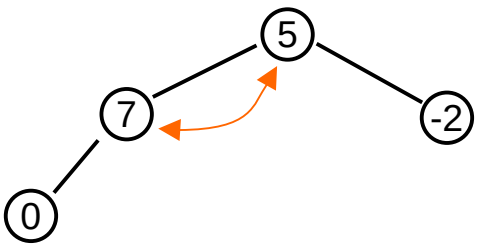

[5, 0, -2, 7, 3]




[5, 0, -2, 7, 3]

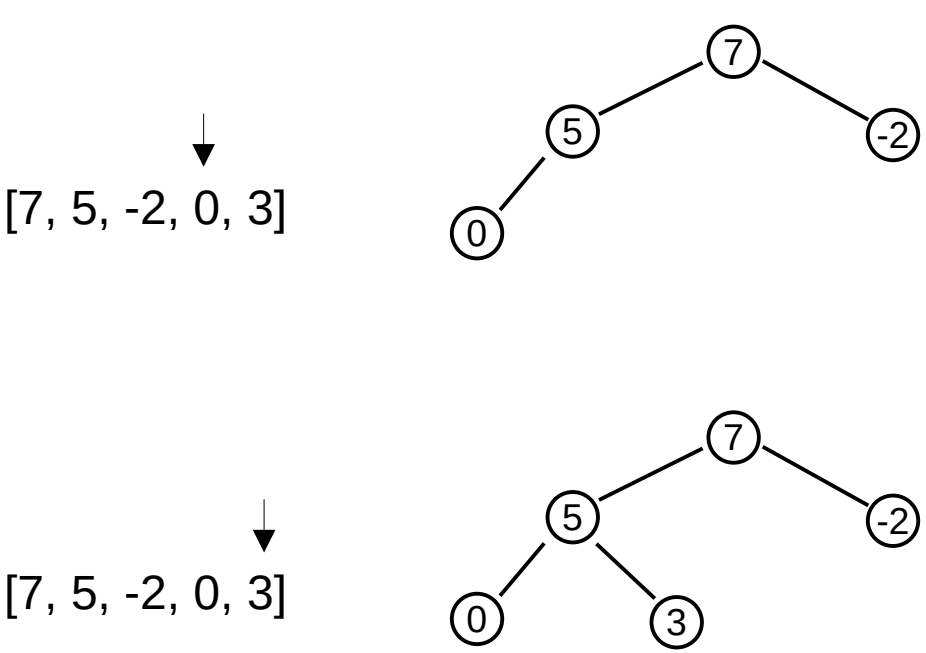



[5, 7, -2, 0, 3]




Сначала строим бинарную кучу на основании данных массива.

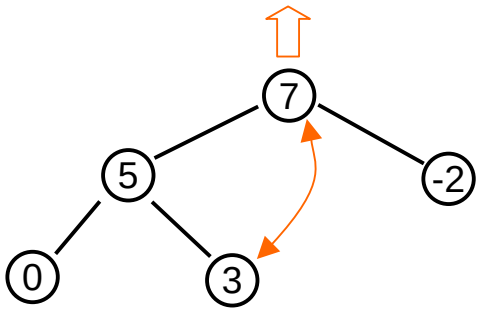
# Графическая иллюстрация работы алгоритма



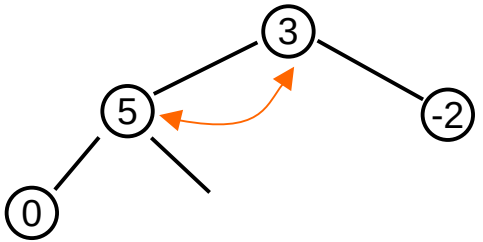

Сначала строим бинарную кучу на основании данных массива.

# Графическая иллюстрация работы алгоритма

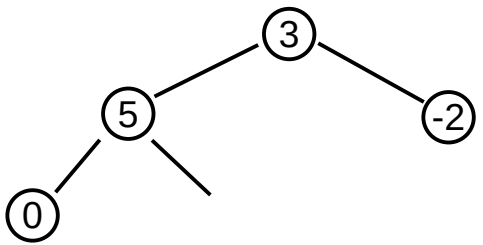
[7, 5, -2, 0, 3]



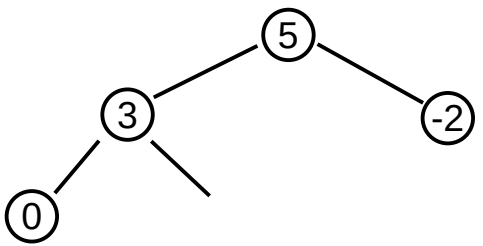

[3, 5, -2, 0, 7]




[3, 5, -2, 0, 7]



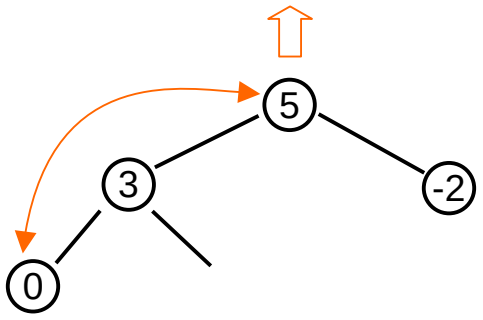

[5, 3, -2, 0, 7]



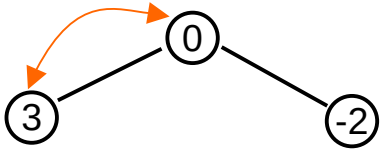

Переносим первый элемент в отсортированную часть и выполняем просеивание вниз на не сортированной части.

# Графическая иллюстрация работы алгоритма

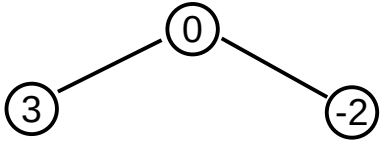
[5, 3, -2, 0, 7]



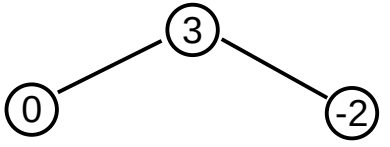

[0, 3, -2, 5, 7]




[0, 3, -2, 5, 7]



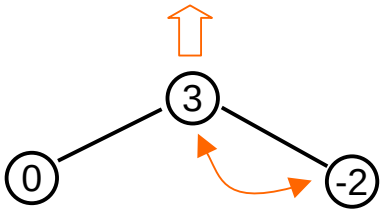

[3, 0, -2, 5, 7]



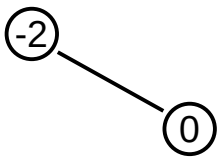

Переносим первый элемент в отсортированную часть и выполняем просеивание вниз на не сортированной части.

# Графическая иллюстрация работы алгоритма

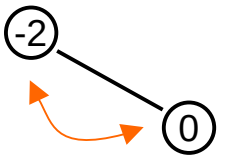
[3, 0, -2, 5, 7]



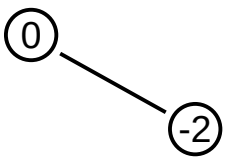

[-2, 0, 3, 5, 7]




[-2, 0, 3, 5, 7]

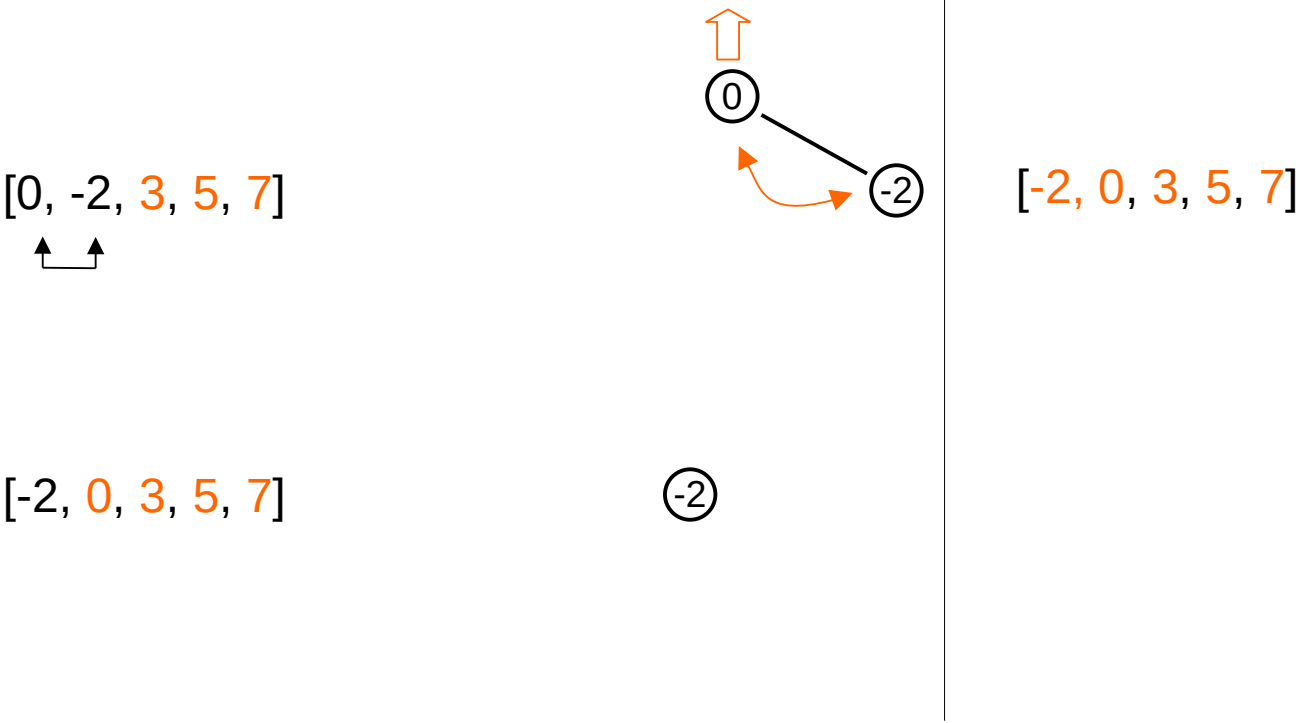



[0, -2, 3, 5, 7]




Переносим первый элемент в отсортированную часть и выполняем просеивание вниз на не сортированной части.

# Графическая иллюстрация работы алгоритма




Переносим первый элемент в отсортированную часть и выполняем просеивание вниз на не сортированной части.

# Реализация на Python

# Функции просеивания вверх и вниз

```
def sift_up(arr, i):
    while i > 0:
        j = (i-1)//2
        if arr[i] > arr[j]:
            arr[i], arr[j] = arr[j], arr[i]
        else:
            break
        i = j


def sift_down(arr, i, last_index):
    while True:
        left_j = 2 * i + 1
        right_j = 2 * i + 2
        j = i
        if left_j < last_index and arr[left_j] > arr[j]:
            j = left_j
        if right_j < last_index and arr[right_j] > arr[j]:
            j = right_j
        if j != i:
            arr[i], arr[j] = arr[j], arr[i]
            i = j
        else:
            break
```

Python

# Функция для создания кучи

```
def heapify(arr):
    for i in range(len(arr)):
        sift_up(arr, i)
```

Python

# Функция сортировки кучей

```
def heap_sort(arr):
    heapify(arr)
    last_index = len(arr) - 1
    while last_index > 0:
        arr[0], arr[last_index] = arr[last_index], arr[0]
        sift_down(arr, 0, last_index)
        last_index -= 1
```

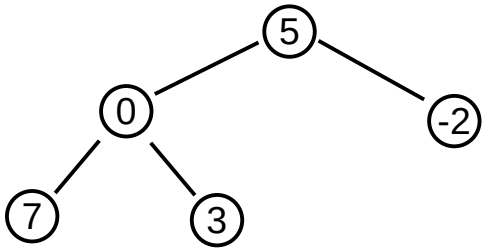
# Модификация алгоритма

Можно модифицировать этот алгоритм упростив реализацию. Стоит заметить, что применяя просеивание вниз кучи начиная с ее первой половины, можно также привести ее нужному виду. В таком случае просеивание вверх уже не нужно.

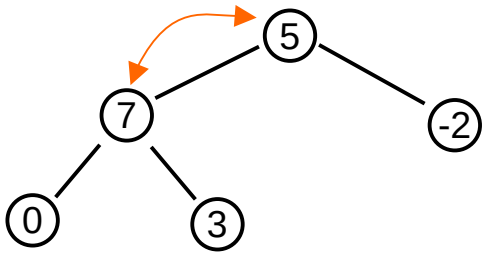
Таким образом алгоритм приобретает следующий вид. Начиная от середины массива в сторону начала проводим просеивание вниз. Дальнейший ход алгоритма не отличается от описанного ранее.

# Графическая иллюстрация работы алгоритма

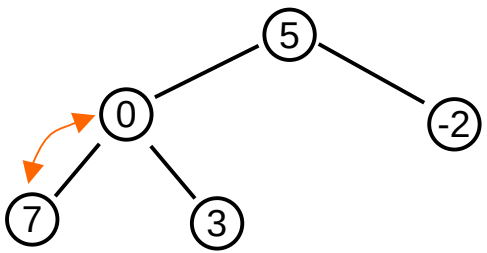
[5, 0, -2, 7, 3]



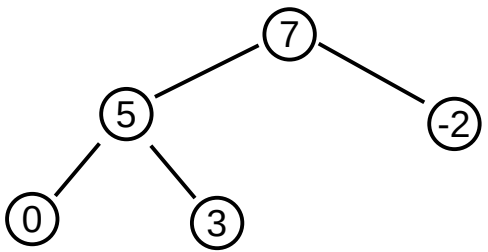

[5, 7, -2, 0, 3]




[5, 0, -2, 7, 3]




[7, 5, -2, 0, 3]




Построение кучи с помощью просеивания вниз

# Реализация на Java

# Методы для просеивания вниз

```
public static void siftDOwn(int[] array, int i, int lasIndex) {
     for (;;) {
          int leftIndex = 2 * i + 1;
          int rightIndex = 2 * i + 2;
          int j = i;
          if (leftIndex < lasIndex && array[leftIndex] > array[j]) {
               j = leftIndex;
          }
          if (rightIndex < lasIndex && array[rightIndex] > array[j]) {
               j = rightIndex;
          }
          if (i != j) {
               swap(array, i, j);
               i = j;
          } else {
               break;
          }
     }
}

public static void swap(int[] array, int i, int j) {
     int temp = array[i];
     array[i] = array[j];
     array[j] = temp;
}
```

# Метод реализующий сортировку кучей

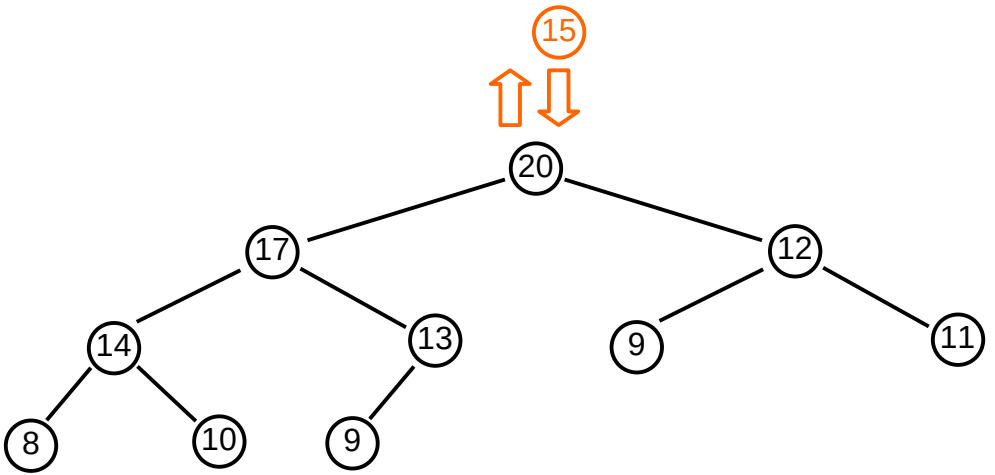
```
public static void heapSort(int[] array) {
    int n = array.length / 2;
    int lastIndex = array.length;
    for (int i = n; i >= 0; i--) {
        siftDOwn(array, i, lastIndex);
    }
    lastIndex -= 1;
    for (; lastIndex > 0;) {
        swap(array, 0, lastIndex);
        siftDOwn(array, 0, lastIndex);
        lastIndex -= 1;
    }
}
```

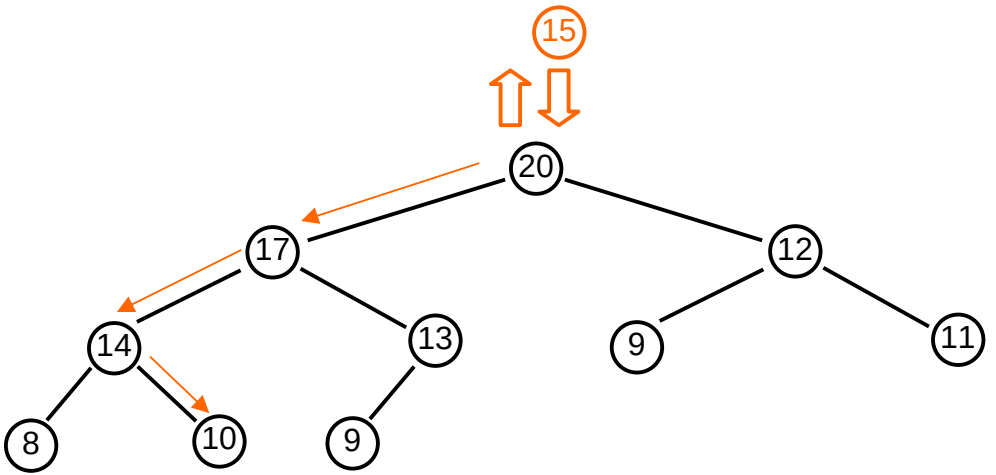
# Сортировка кучей снизу-вверх

Сортировка кучей снизу-вверх (**bottom-up heap sort**) — модификация сортировки кучей которая уменьшает количество сравнений на этапе просеивания вниз. Эта модификация является выигрышной в случае высоких затрат на сравнение двух элементов массива (длинные строки и т. д.). В случае если сравнение происходит быстро (ключи являются базовыми числовыми типами) то ее использование практически не дает никакого выигрыша.

Модифицированное просеивание вниз выглядит следующим образом. Находим максимальный лист кучи (узел на последнем слое у которого нет дочерних) от текущего элемента (чаще всего от корня). После этого двигаемся от этого элемента вверх до нахождения элемента большего текущего. Ставим текущий элемент на найденное место, остальные элементы сдвигаем на один элементы выше по куче.

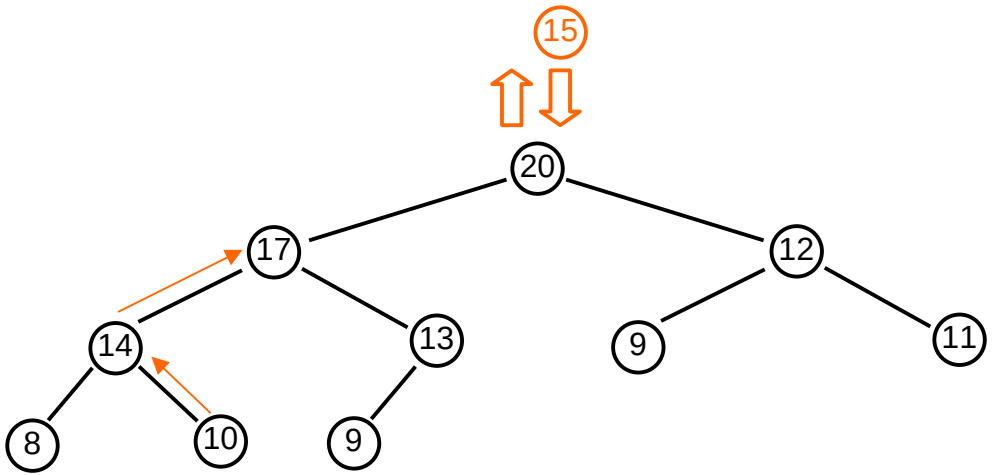
# Графическая иллюстрация модифицированного просеивания вниз

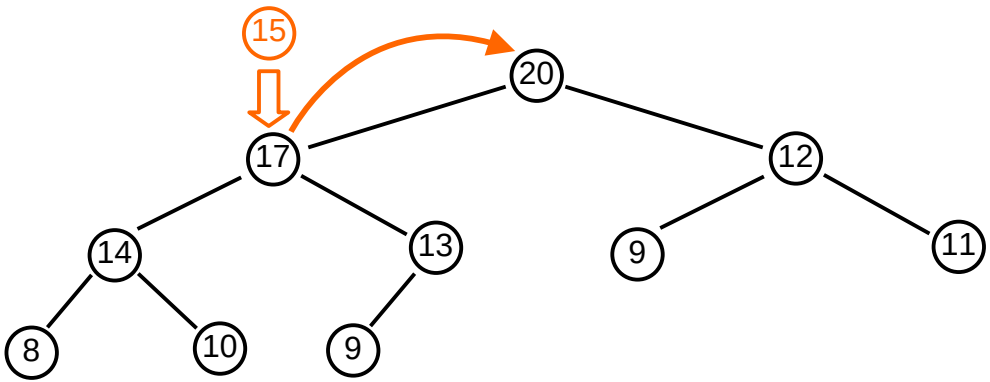



Предположим что выполняется просеивание от вершины кучи (такая операция популярна в операции получения элемента со вставкой). Ищем максимальный лист кучи (в примере это узел с ключом 10).

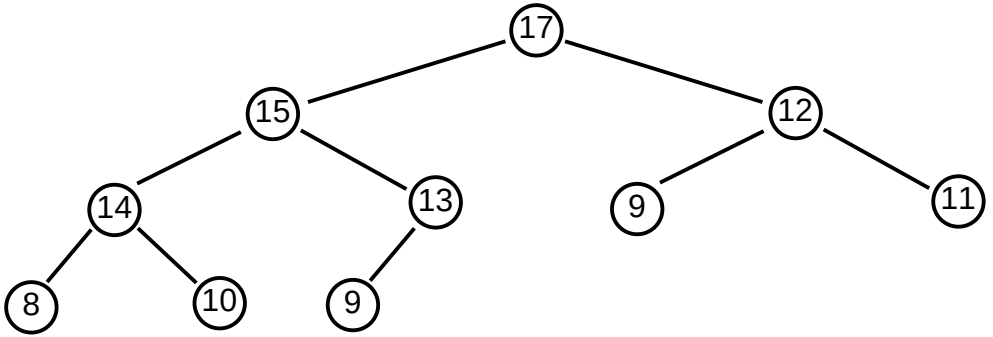
# Графическая иллюстрация модифицированного просеивания вниз




Начиная от найденного узла идем вверх до узла значение которого больше чем вставляемый (17 больше 15). От найденного узла сдвигаем все узлы на один шаг вверх. После чего вставляемый элемент заменяет найденный узел.

# Графическая иллюстрация модифицированного просеивания вниз



# Реализация на Fortran

# Функция для поиска максимального узла

```
function leafSearch(arr, i, last_index)
    implicit none
    integer,intent(in)::arr(:)
    integer,intent(in)::last_index
    integer,intent(in)::i
    integer::j
    integer::leafSearch
    j = i
    do
        if(2 * j + 1 >= last_index) then
            exit
        end if
        if (arr(2 * j) > arr (2 * j + 1)) then
            j = 2 * j
        else
            j = 2 * j + 1
        end if
    end do
    if (2 * j < last_index) then
        j = 2 * j
    end if
    leafSearch = j
end function
```

# Модифицированная процедура просеивания вниз

```
subroutine silf_down(arr,i, last_index)
    implicit none
    integer, intent(inout)::arr(:)
    integer, intent(in)::i, last_index
    integer::j
    j = leafSearch(arr, i, last_index)
    do
        if(arr(i) <= arr(j)) then
            exit
        end if
        j = j / 2
    end do

    do
        if(j <= i) then
            exit
        end if
        call swap(arr(i), arr(j))
        j=j/2
    end do

end subroutine silf_down
```

# Процедура обмена значений двух переменных

```
subroutine swap(first_val, second_val)
    implicit none
    integer, intent(inout)::first_val, second_val
    integer::temp
    temp = first_val
    first_val = second_val
    second_val = temp
end subroutine swap
```

# Процедура сортировки кучей

```
subroutine heap_sort(arr)
    implicit none
    integer,intent(inout)::arr(:)
    integer::i, last_index, n
    n = size(arr, dim = 1)
    last_index = n + 1
    do i = n/2,1,-1
        call silf_down(arr,i, last_index)
    end do
    last_index = n
    do
        if(last_index == 1) then
            exit
        end if
        call swap(arr(1),arr(last_index))
        call silf_down(arr,1, last_index)
        last_index = last_index - 1
    end do
end subroutine heap_sort
```

# Список литературы

1) Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн // Алгоритмы: построение и анализ 3-е издание. — М.: «Вильямс», 2013. — С. 1328. ISBN 978-5-8459-1794-2
2) Роберт Седжвик, Кевин Уэйн «Алгоритмы на java 4-е издание» Пер. с англ. - М. : ООО "И.Д. Вильямс", 2013. ISBN 978-5-8459-1781-2.