Алгоритмы сортировки. Сложность алгоритмов.

Семестр 1

Семинар 7

Задача сортировки массива

Вход: последовательность, состоящая из n чисел $\langle a_1, a_2, \ldots, a_n \rangle$.

Выход: перестановка (изменение порядка) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ входной последовательности таким образом, что $a'_1 \leqslant a'_2 \leqslant \cdots \leqslant a'_n$.

Фундаментальная задача:

- Частое применение
- История разработки алгоритмов
- Математическое доказательство оценок времени работы
- Выявление прикладных проблем

Сортировка пузырьком

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 10

void show_array (int * _a) {
    for (int i=0; i<N; i++)
        printf("%d ",_a[i]);
    putchar('\n');
}
void swap (int *x1, int *x2) {
    int temp = *x1;
    *x1=*x2;
    *x2=temp;
}
```

```
int main() {
    int array[N], i, j, temp;
    srand(time(NULL));
    for (i=0; i<N; i++)
        array[i]=rand()%100;
    show_array(array);

    for (i=0; i<N; i++)
        for (j=0; j<N-1; j++)
            if (array[j]>array[j+1])
                swap(array+j,array+j+1);

    show_array(array);
    system("pause");
    return 0;
}
```

Сортировка пузырьком: оценка времени работы

```
void swap (int *x1, int *x2) {
    int temp = *x1;
    *x1=*x2;
    *x2=temp;
}

for (i=0; i<N; i++)
   for (j=0; j<N-1; j++)
      if (array[j]>array[j+1])
         swap(array+j,array+j+1);
```

Лучший вариант
(массив уже отсортирован)

$c^1*N + c^2*N*(N-1) + c^3*N*(N-1) \sim N^2$

Худший вариант
(массив отсортирован наоборот)

$c^1*N + c^2*N*(N-1) +$
$c^3*N*(N-1) + c^4*N*(N-1) \sim N^2$

Средний вариант
(произвольный массив)

$c^1*N + c^2*N *(N-1) +$
$c^5*N *(N-1) + c^6*N *(N-1) \sim N^2$

Сортировка пузырьком: модификация

```
void swap (int *x1, int *x2) {
    int temp = *x1;
    *x1=*x2;
    *x2=temp;
}

for (i=0; i<N; i++) {
   check = 0;
   for (j=0; j<N; j++)
      if (array[j]>array[j+1]) {
        check = 1;
        swap(array+j,array+j+1);
      }
   if (!check) break;
}
```

Лучший вариант
(массив уже отсортирован)
$c^1 + c^2*(N-1) + c^3*(N-1) \sim N$

Худший вариант
(массив отсортирован наоборот)
$c^1*N + c^2*N*(N-1) +$
$c^3*N*(N-1) + c^4*N*(N-1) \sim N^2$

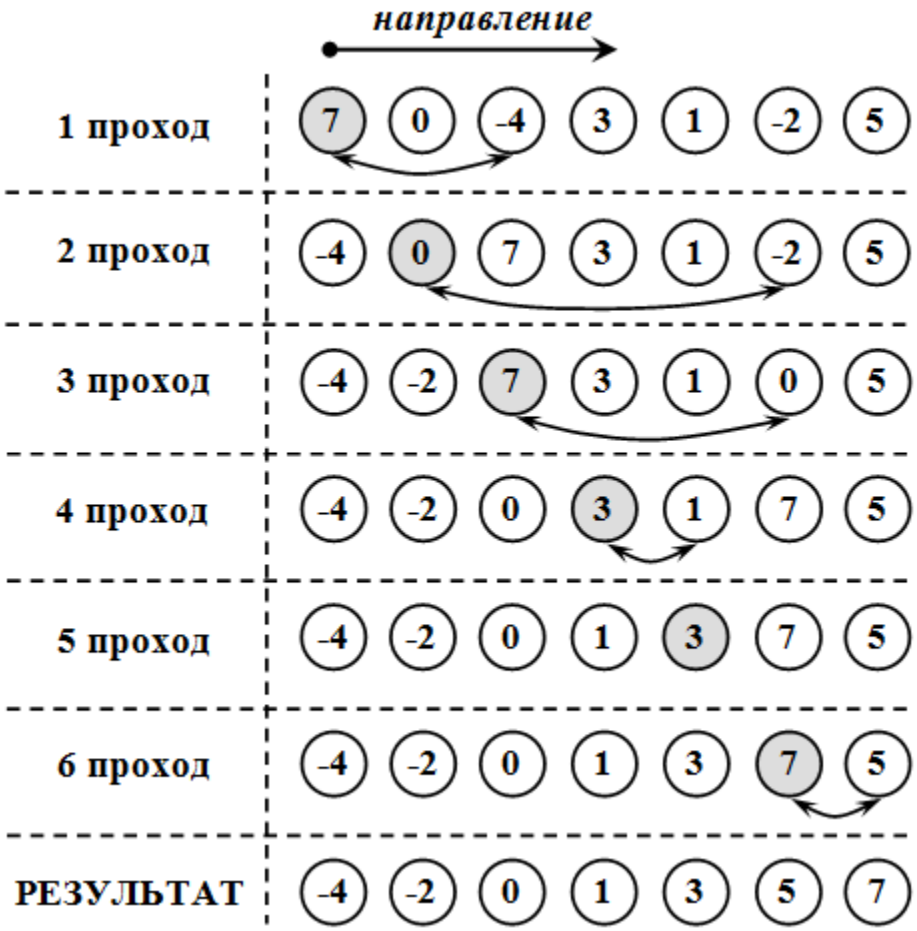
Средний вариант
(произвольный массив)
$c^1*N + c^2*N*(N-1) +$
$c^5*N*(N-1) + c^6*N*(N-1) \sim N^2$

Сортировка пузырьком: шейкерная сортировка

- Идем поочередно слева направо и справа налево
- Границы рабочей части массива устанавливаются в месте последнего обмена
- Лучшая сложность O(N)
- Средняя сложность $O(N^2)$
- Худшая сложность $O(N^2)$

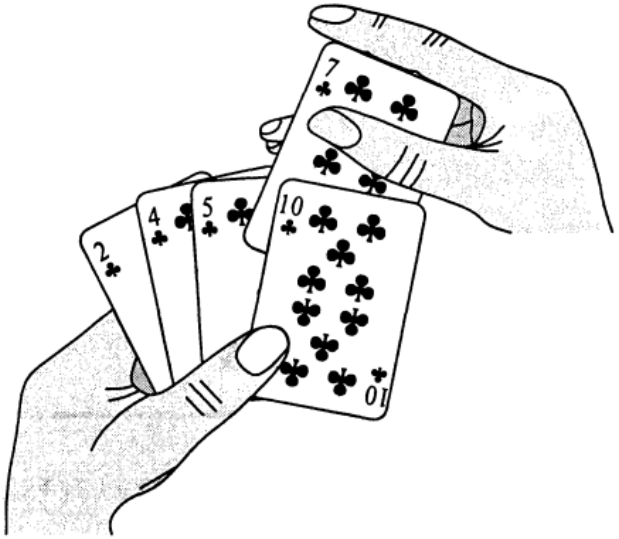
3 1 5 8 1 0 4 6 6 7
3 1 5 8 0 1 4 6 6 7
3 1 5 0 8 1 4 6 6 7
3 1 0 5 8 1 4 6 6 7
3 0 1 5 8 1 4 6 6 7
0 3 1 5 8 1 4 6 6 7 Left=1
0 1 3 5 8 1 4 6 6 7
0 1 3 5 1 8 4 6 6 7
0 1 3 5 1 4 8 6 6 7
0 1 3 5 1 4 6 8 6 7
0 1 3 5 1 4 6 6 8 7
0 1 3 5 1 4 6 6 7 8 Right=8
0 1 3 1 5 4 6 6 7 8
0 1 1 3 5 4 6 6 7 8 Left=3
0 1 1 3 4 5 6 6 7 8

Сортировка методом выбора

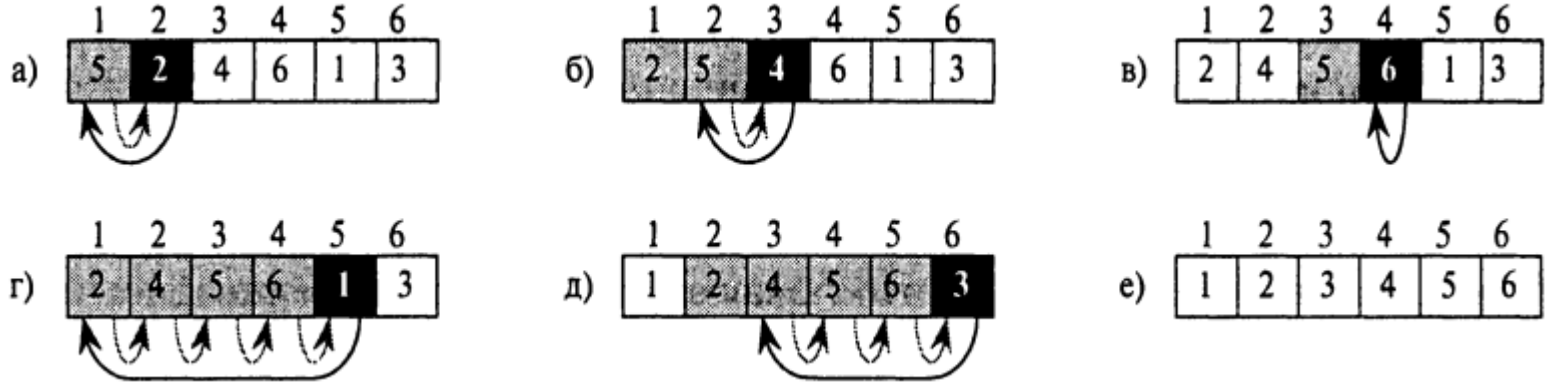


- Лучшая сложность $O(N^2)$
- Средняя сложность $O(N^2)$
- Худшая сложность $O(N^2)$

Сортировка вставкой



- Лучшая сложность O(N)
- Средняя сложность $O(N^2)$
- Худшая сложность $O(N^2)$



Сравнение алгоритмов

Name	Best	Average	Worst	Memory	Stable
Binary tree sort	n	$n \log n$	$n \log n$	n	Yes
Bubble sort	n	n^2	n^2	1	Yes
Cocktail sort	n	n^2	n^2	1	Yes
Comb sort	n	$n \log n$	n^2	1	No
Cycle sort	—	n^2	n^2	1	No
Franceschini's method[5]	—	$n \log n$	$n \log n$	1	No
Gnome sort	n	n^2	n^2	1	Yes
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No
In-place merge sort	—	—	$n (\log n)^2$	1	Yes
Insertion sort	n	n^2	n^2	1	Yes
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No
Library sort	—	$n \log n$	n^2	n	Yes
Merge sort	$n \log n$	$n \log n$	$n \log n$	Depends[*further explanation needed*]; worst case is n	Yes

Сравнение алгоритмов

Patience sorting	—	—	$n \log n$	n	No
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$ on average, worst case is n	typical in-place sort is not stable; stable versions exist
Selection sort	n^2	n^2	n^2	1	No
Shell sort	n	$n(\log n)^2$ or $n^{3/2}$	Depends on gap sequence; best known is $n(\log n)^2$	1	No
Smoothsort	n	$n \log n$	$n \log n$	1	No
Strand sort	n	n^2	n^2	n	Yes
Timsort	n	$n \log n$	$n \log n$	n	Yes
Tournament sort	—	$n \log n$	$n \log n$	n[4]	

Сортировка слиянием

(merge sort)

Джон фон Нейман, 1945 г.

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

6 5 3 1 8 7 2 4

Сортировка слиянием
(merge sort)

Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

Основная идея слияния двух отсортированных массивов:

1. Исходное положение. Пусть мы имеем два подмассива. Элементы подмассивов в каждом из этих подмассивов отсортированы по возрастанию.
2. Слияние двух подмассивов в третий результирующий массив.
   На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив.
3. Счетчики номеров элементов результирующего массива и подмассива из которого был взят элемент увеличиваем на 1.
4. "Прицепление" остатка.
   Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

Сортировка слиянием (merge sort)

merge.cpp

- Лучшая сложность O(N log N)
- Средняя сложность O(N log N)
- Худшая сложность O(N log N)

- Память O(N)

Наглядное сравнение

Пузырьком

O(n^2)
«Быстрый» компьютер
1 000 000 000 операций/сек
«Хороший» код
2 n^2

Слияния

O(n log n)
«Медленный» компьютер
10 000 000 операций/сек
«Плохой» код
50 n log n

1 000 000 чисел

$$\frac{2 \cdot (10^6)^2 \text{команд}}{10^9 \text{команд / с}} = 2000 \text{ с}$$

$$\frac{50 \cdot 10^6 \cdot \lg 10^6 \text{команд}}{10^7 \text{команд / с}} \approx 100 \text{ с}$$

Сортировка Хоара
(quick sort)

Чарльз Хоар, 1960

1. Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива.
2. Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на два непрерывных отрезка, следующие друг за другом — «меньшие опорного», «равные и большие».
3. Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

- Лучшая сложность $O(N \log N)$
- Средняя сложность $O(N \log N)$
- Худшая сложность $O(N^2)$

Указатели на функции

- Переменная, содержащая адрес в памяти, по которому расположена функция.
- Аналог - адрес начала массива.

Объявление переменной compare

```
int (* compare) (double, char);
int foo (double x, char c) { … }
…
compare = foo;
int z = foo (3.5, 'a');
int k = compare (-0.7, 'w');
```

Использование qsort

```
void qsort (
    void * arr,   size_t count,   size_t cell_size,
    int (*compare)(void *, void*)                 );

int numcmp (void * _p1, void * _p2) {
        int * p1 = (int*) _p1;
        int * p2 = (int*) _p2;
        int x1 = *p1;
        int x2 = *p2;
        return x2-x1;
}

int arr [10];
qsort (arr, 10, sizeof (a[0]), numcmp );
```

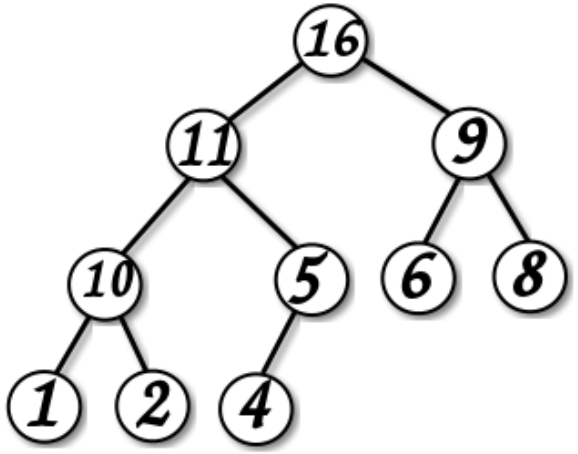
Время работы алгоритма

в миллисекундах:

```
#include <time.h>
...
   int start = clock();
...
   int finish = clock();
   printf("%d\n", finish-start);
```

Сортировка кучей

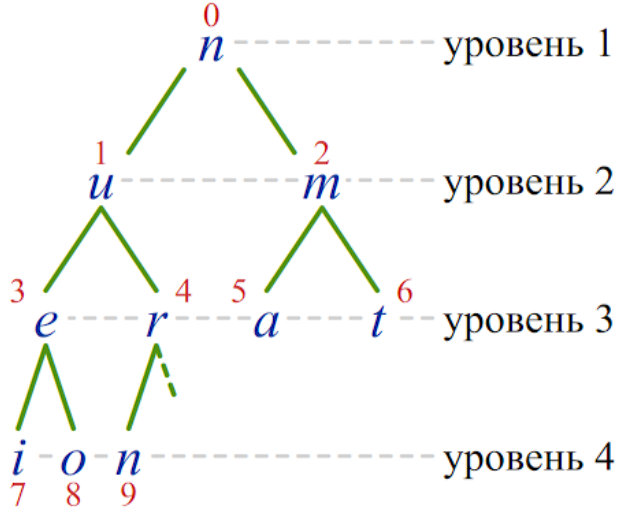
(heapsort)



16	11	9	10	5	6	8	1	2	4

Сортирующее дерево:

1. все листовые элементы находятся в нижнем уровне, либо в нижних двух уровнях;
2. все листья в уровне заполняют уровень слева;
3. все уровни (кроме, быть может, последнего уровня) полностью заполнены элементами.

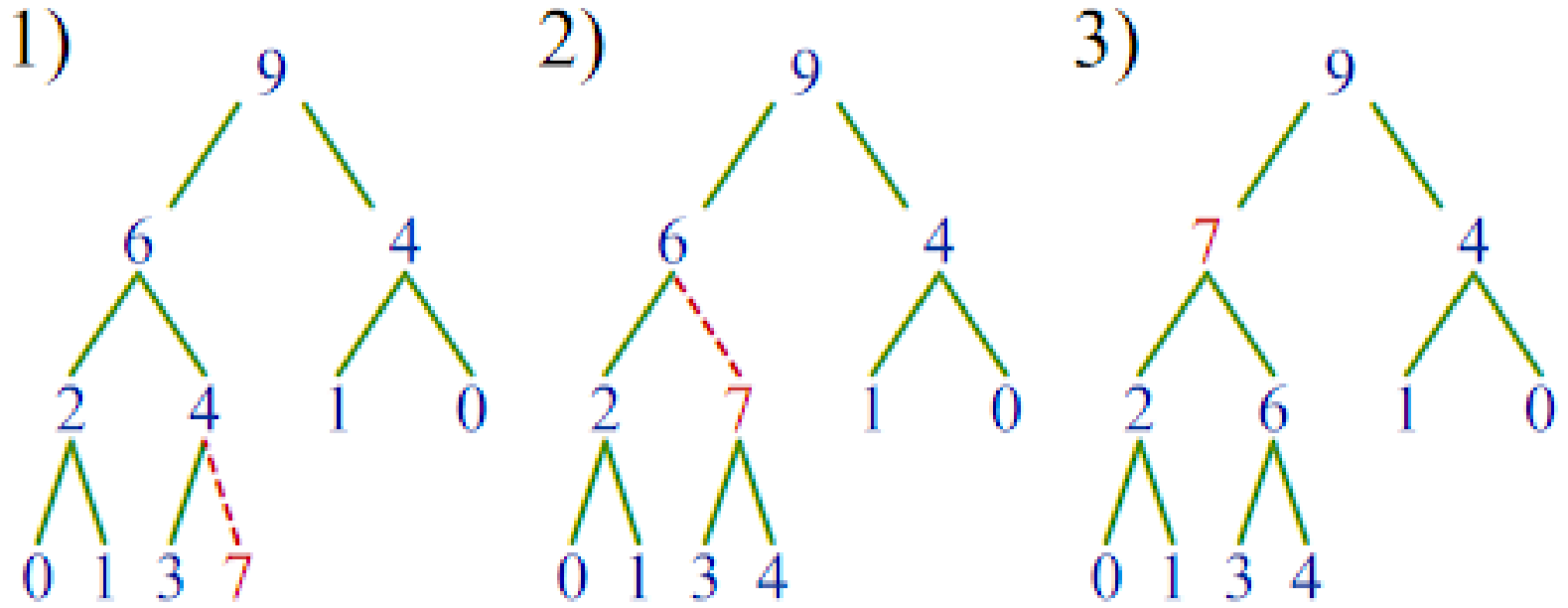


Потомки узла i имеют индексы 2i+1 и 2i+2, родитель - индекс (i-1)/2.

Сортировка кучей
(heapsort)

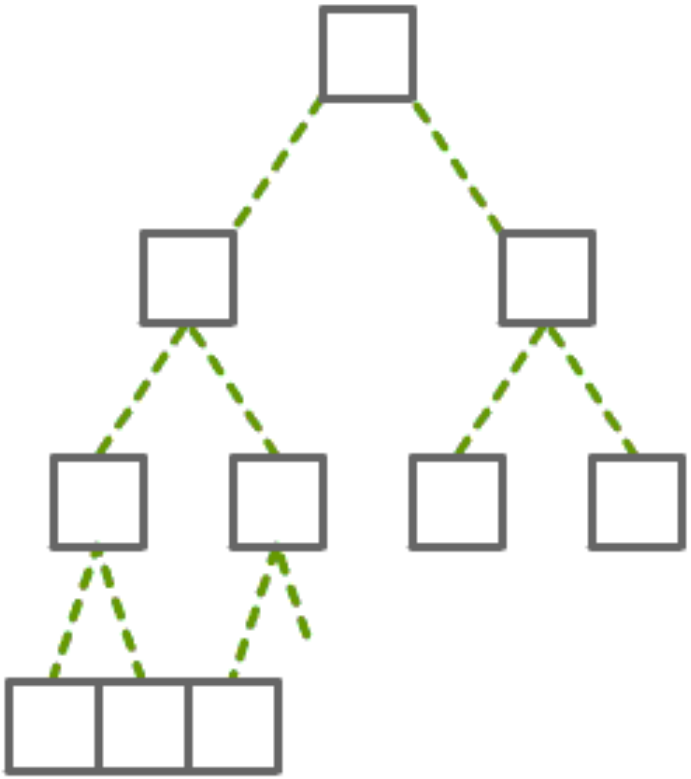
Просеивание вверх:

1. если элемент корневой, или его значение меньше/равно значению родителя, то конец;
2. меняем местами значения элемента и его родителя;
3. переходим к родителю, и выполняем для него этот же алгоритм, начиная с пункта 1.



Сортировка кучей

(heapsort)

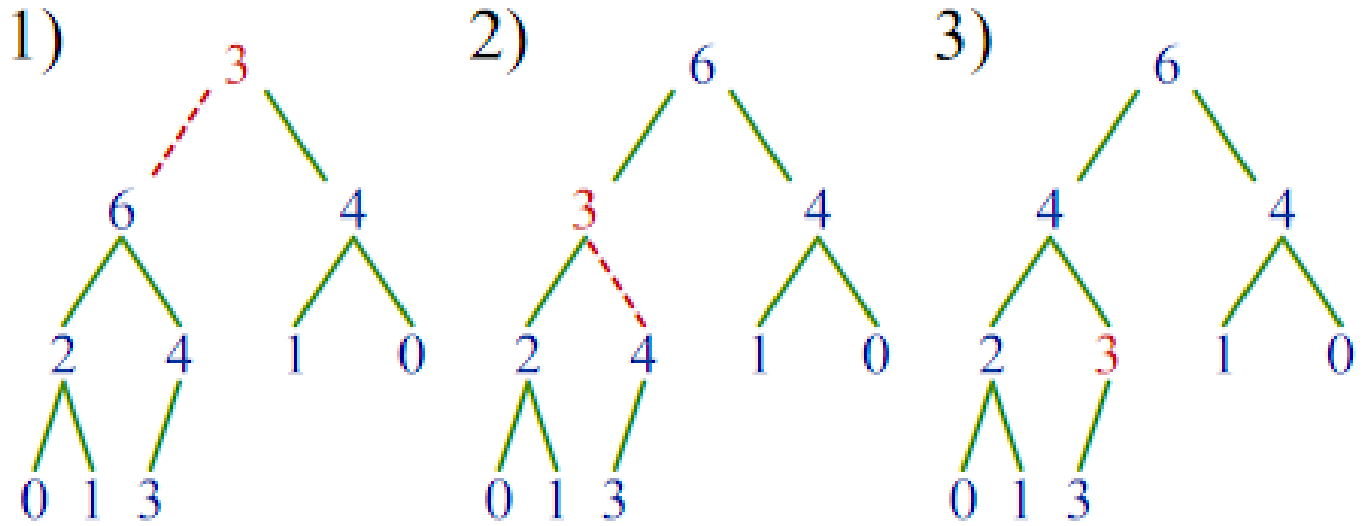


1	3	5	2	4	6	7	2	3	8

Сортировка кучей
(heapsort)

Просеивание вниз:

1. если элемент листовой, или его значение меньше/равно значению потомков, то конец;
2. иначе меняем местами значения элемента и его потомка, имеющего максимальное значение;
3. переходим к изменившемуся потомку, и выполняем для него этот же алгоритм, начиная с пункта 1.



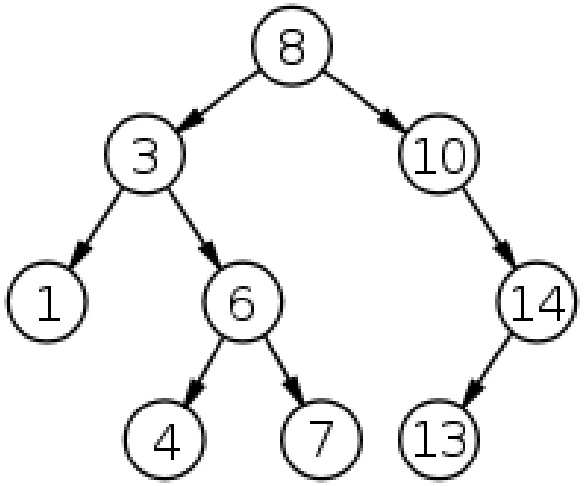
Сортировка кучей
(heapsort)

Сортировка:

1. поменять местами значения первого и последнего элементов пирамиды;
2. отделить последний элемент от дерева, уменьшив размер дерева на единицу (элемент остаётся в массиве);
3. восстановить пирамиду, просеяв вниз её новый корневой элемент;
4. перейти к пункту 1;

Сортировка двоичным деревом
(tree sort)

1. Построение двоичного дерева.
2. Сборка результирующего массива путём обхода узлов в необходимом порядке следования.



- Лучшая сложность O(N)
- Средняя сложность O(N log N)
- Худшая сложность $O(N^2)$

- Память O(N)