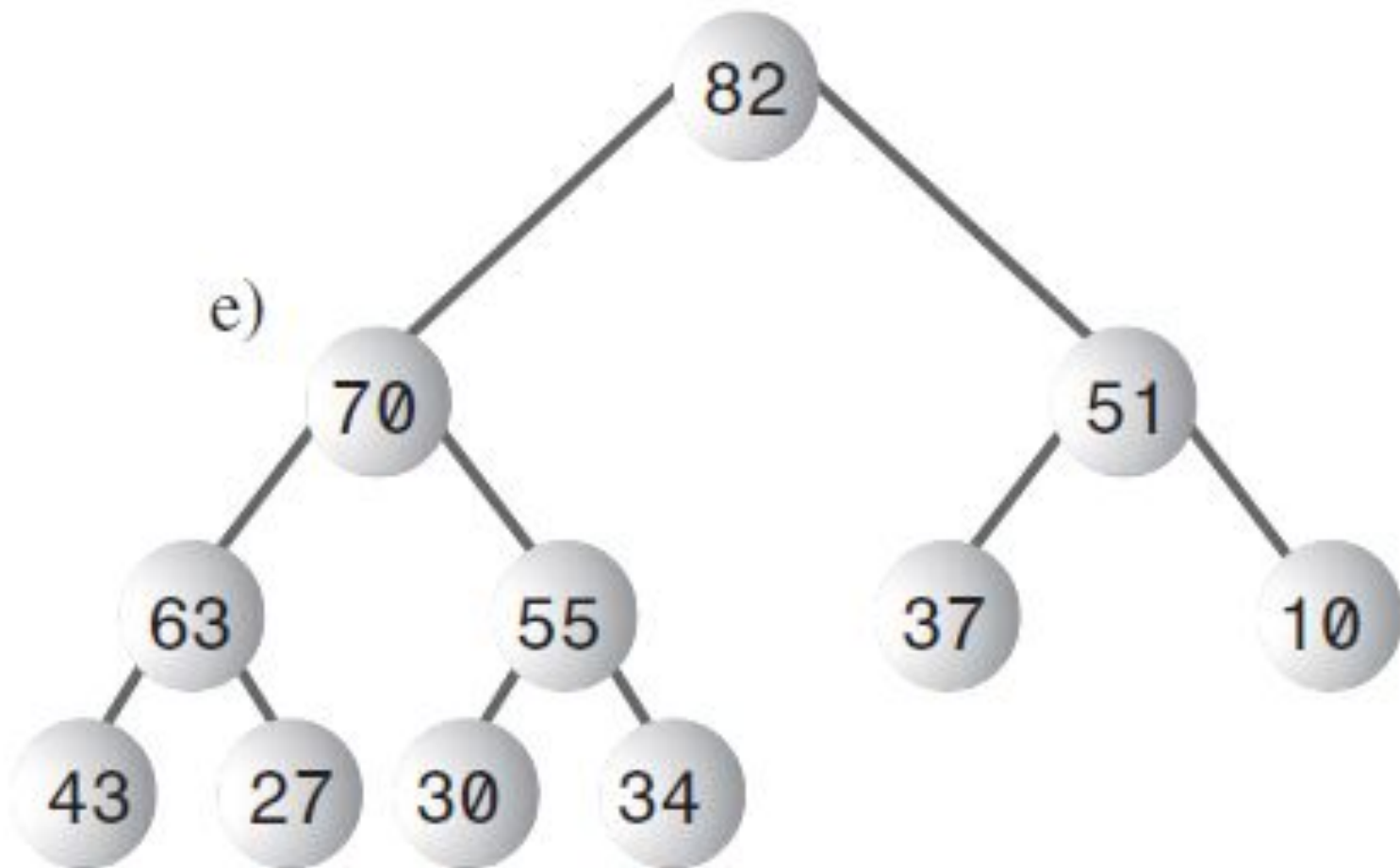


# Пирамиды и пирамидальная сортировка

# Пирамида

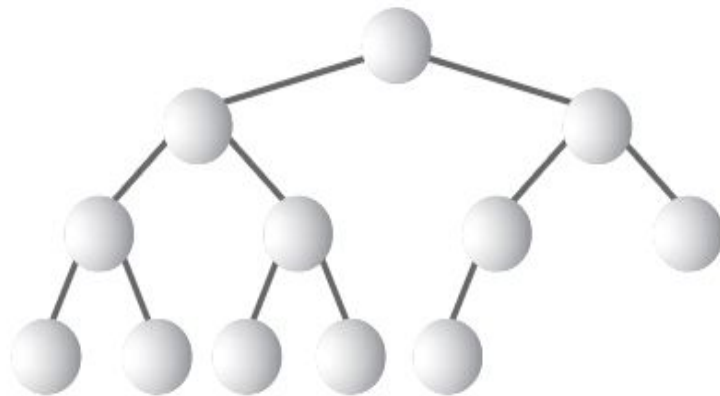
- Пирамида является структурой данных, позволяющей обратиться к наибольшему элементу за время  $O(1)$ , а также позволяющая удалять наибольший элемент и вставлять новый элемент за время  $O(\log_2 N)$ .
- *Пирамида* – это разновидность двоичного дерева, обладающего двумя свойствами. Первое свойство заключается в том, что любой узел такого дерева больше либо равен любому из своих потомков.
- Это свойство называется свойством *слабой упорядоченности*.

# Пример пирамиды

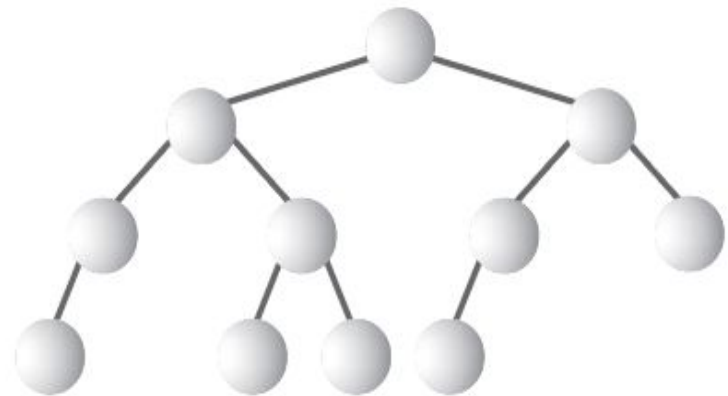


# СВОЙСТВО ПОЛНОТЫ

- Второе свойство называется свойством полноты и заключается в том, что это дерево содержит все возможные узлы при заполнении слева направо и сверху вниз.
- На рисунке слева изображено полное (Complete) дерево, а справа – неполное (Incomplete).



a) Complete

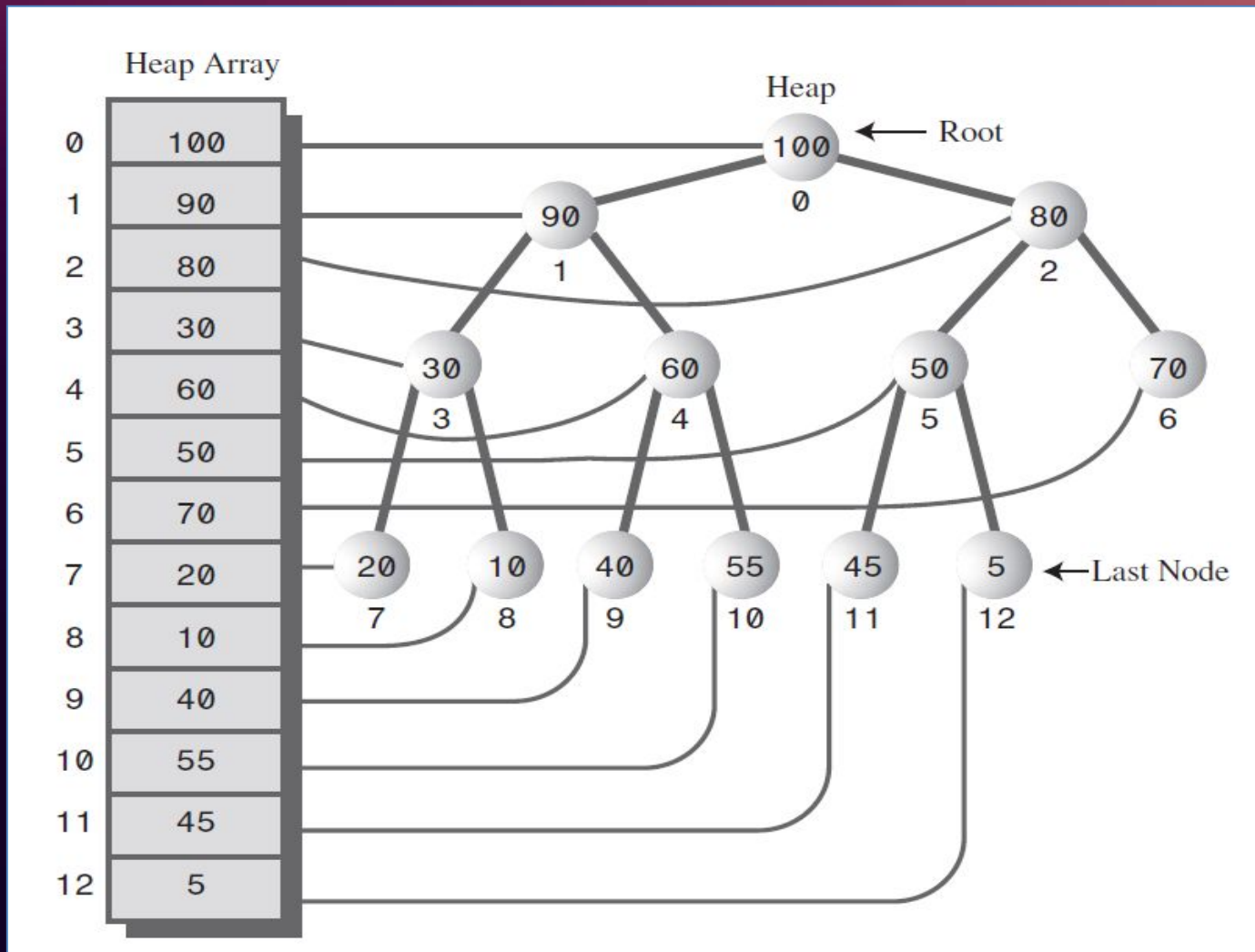


b) Incomplete

# Пирамида на основе массива

- Наиболее простой реализацией пирамиды является реализация на основе массива.
- При этом каждому узлу дерева соответствует конкретная ячейка массива.
- Именно свойство полноты позволяет осуществить такое отображение (обеспечить отсутствие «дыр» в массиве).

# Пример пирамиды на основе массива



# Связь между деревом и массивом

- Благодаря такому отображению для любого узла можно определить индексы его родителя и потомков. Пусть  $i$  – индекс некоторого узла в массиве, тогда
  - индекс родителя равен  $(i-1) / 2$ ;
  - индекс левого потомка равен  $2*i+1$ ;
  - индекс правого потомка равен  $2*i+2$ .
- В качестве упражнения проверьте эти формулы для приведённого рисунка.

# Реализация операций

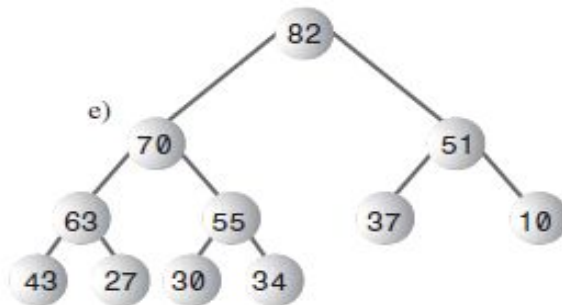
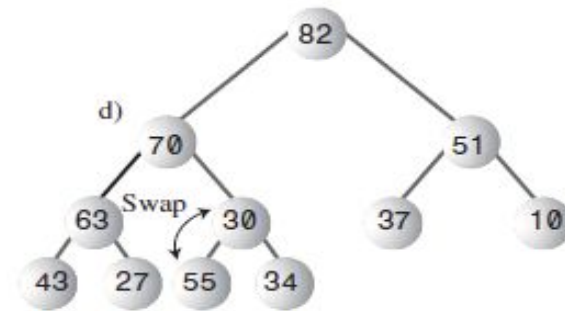
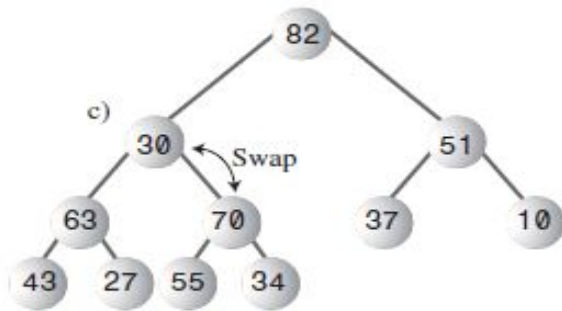
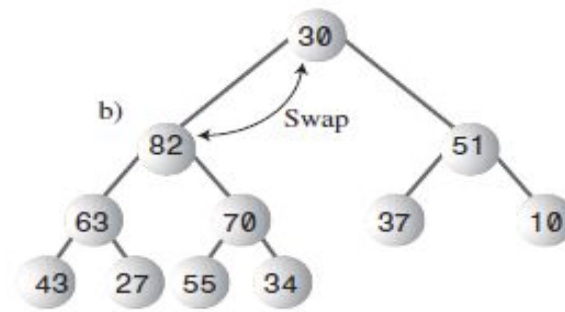
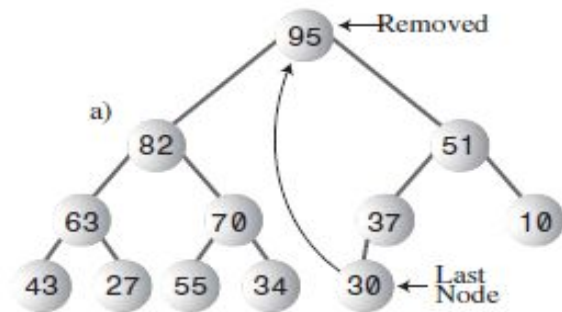
- Рассмотрим теперь, как реализуются операции взятия наибольшего элемента, вставки нового элемента и удаления наибольшего.
- Для обращения к наибольшему элементу пирамиды достаточно взять первый элемент массива.
- Операции вставки нового элемента и удаления наибольшего элемента несколько сложнее, то осуществляются быстро – за время  $O(\log_2 N)$ .



# Удаление вершины

- Поскольку удаляется первый элемент, то в массиве образуется «дыра», и возникает необходимость исключить её, восстановив тем самым полноту дерева. Для этого существует следующий алгоритм:
  - переместить последний узел в корень (на место удаленного);
  - сместить его вниз до тех пор, пока он не окажется на подходящем месте (меньше либо равен родителям и больше либо равен потомков).
- При смещении вниз, очевидно, существуют две альтернативы: влево или вправо. Так вот смещать следует в сторону большего из этих двух узлов, чтобы не нарушить слабую упорядоченность.

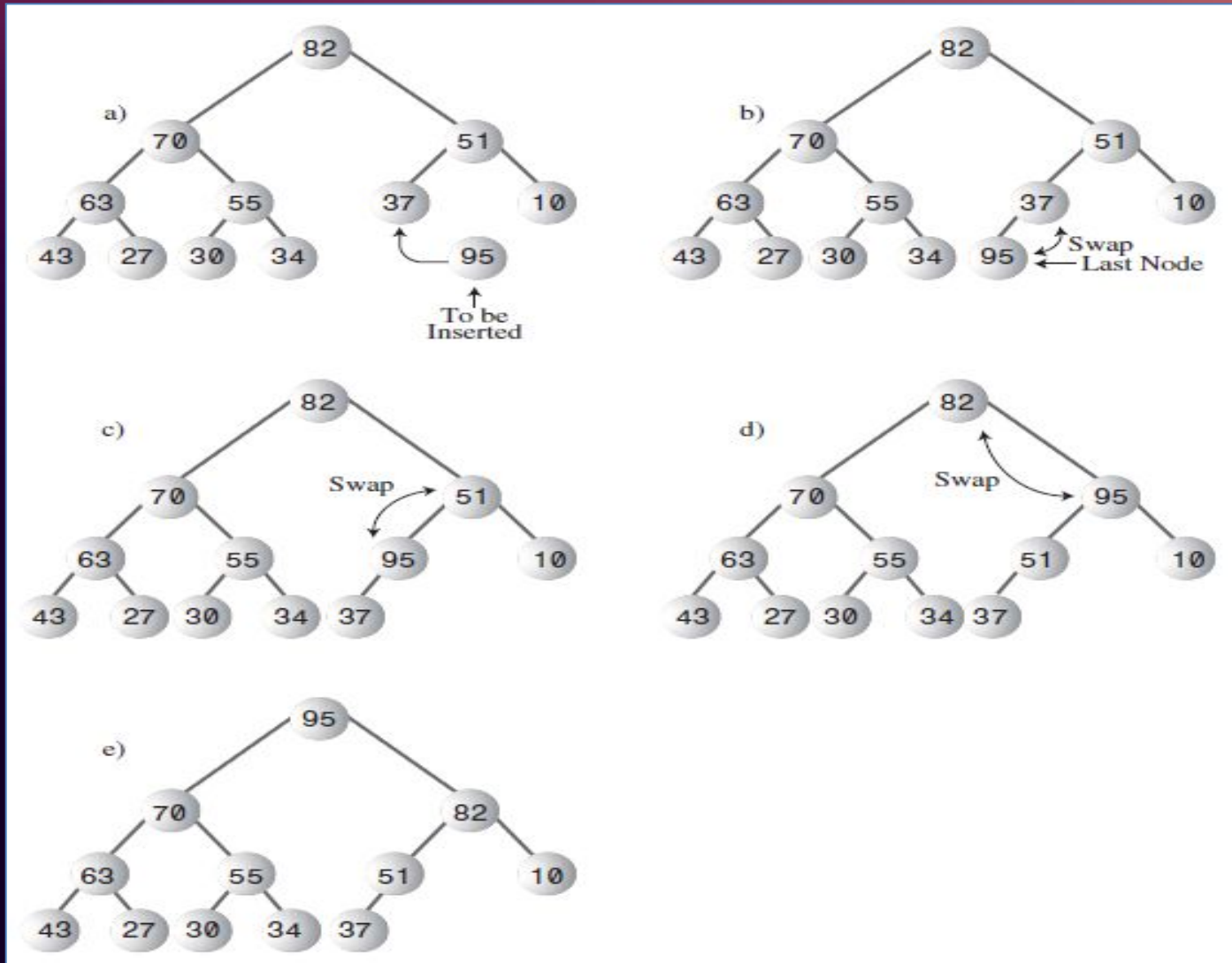
# Пример удаления вершины



# Добавление нового элемента

- Рассмотрим теперь операцию добавления нового элемента.
- Сложность этой процедуры состоит в том, что после добавления должна быть сохранена слабая упорядоченность дерева.
- Алгоритм добавления похож на алгоритм удаления и состоит из следующих шагов:
  - поместить новый узел на последнюю позицию;
  - сместить его вверх до тех пор, пока он не станет меньше либо равен своего родителя.

# Пример добавления



# Пирамидальная сортировка

- С использованием пирамиды можно осуществить сортировку массива по следующему алгоритму.

```
Цикл  $i=0..N-1$   
    Добавить (Массив [ $i$ ] ) ;
```

```
Цикл  $i=0..N-1$   
    Массив [ $i$ ] := Вершина Пирамиды ;  
    Удалить Вершину Пирамиды ;
```

- Массив окажется отсортированным в силу того, что вершина пирамиды – это её наибольший элемент. Этот алгоритм напоминает сортировку методом прямого выбора, но с более эффективным поиском максимального элемента.

# Реализация пирамиды на C++

```
//Вместимость пирамиды
const int size = 20;
int pyramid[size];
//Текущее число элементов в пирамиде
int n;
void swap(int i, int j) {
    int t=pyramid[i];
    pyramid[i]=pyramid[j];
    pyramid[j]=t;
}
```

# Реализация добавления

```
void add(int x) {  
    n++; int i=n-1; pyramid[i] = x;  
    while (i>0 && pyramid[i]>pyramid[(i-1)/2]) {  
        swap(i, (i-1)/2);  
        i = (i-1)/2;  
    }  
}
```

```
void init() {  
    n=0;  
    for (int i=0; i<20; i++) {  
        add(rand() %100);  
    }  
}
```

# Тестирование программы

```
void displayPyramid() {
    for (int i=0; i<n; i++) {
        cout.width(3);
        cout << pyramid[i];
    }
    cout << endl;
}

int main() {
    init();
    displayPyramidTree();
    system("pause");
    return 0;
}
```

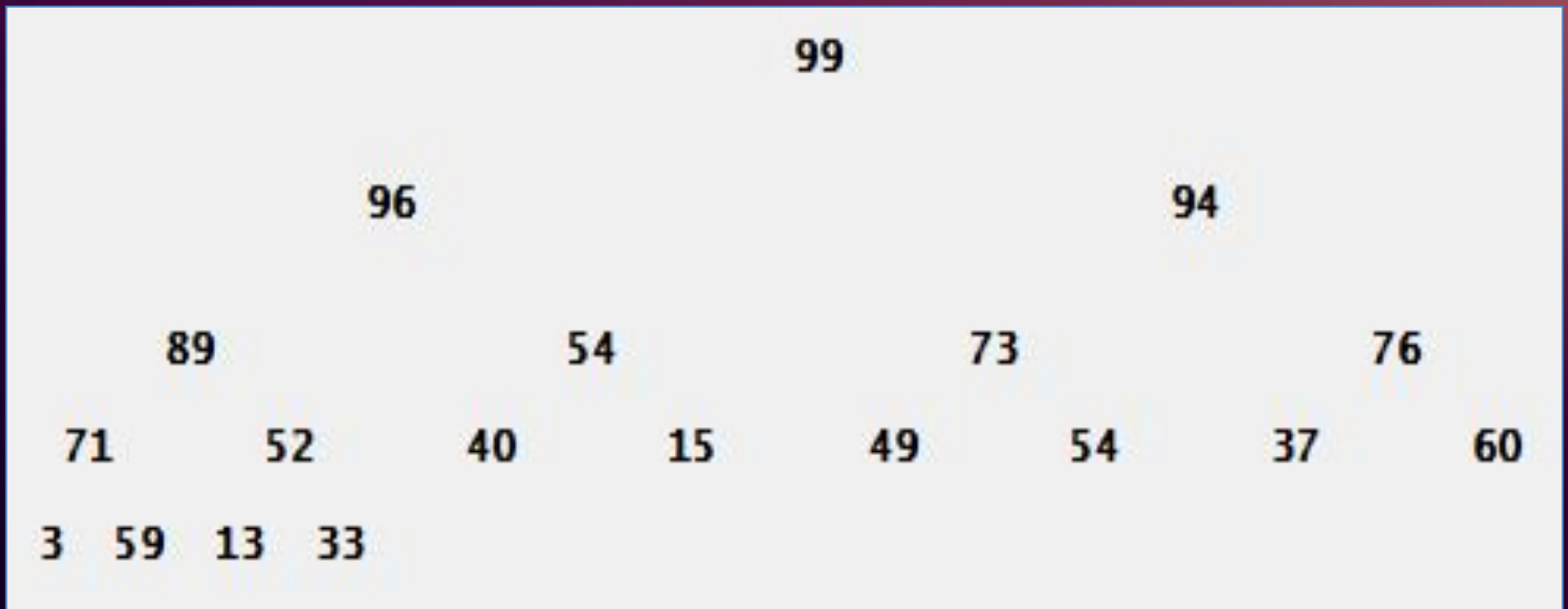
95 91 78 81 64 69 61 67 58 41 5 24 45 27 34 0 62 42 27 36



# Реализация вывода пирамиды в виде дерева

```
void displayPyramidTree() {
    int k=1;
    int j=0;
    int sp=pow(2.0,log2(n));
    for (int l=0; l<log2(n)+1; l++) {
        makeSpaces(sp);
        for (int i=0; i<k; i++) {
            cout.width(2);
            cout << pyramid[j];
            makeSpaces(sp*2-1);
            j++;
            if (j==n-1) {
                break;
            }
        }
        k*=2;
        sp/=2;
        cout << endl << endl;
    }
}
```

# Вывод пирамиды в виде дерева



# Реализация удаления

## ВЕРШИНЫ

```
int top() { return pyramid[0]; }

void pop() {
    pyramid[0] = pyramid[n-1];
    n--;
    int i=0;
    while (i<n && (2*i+1<n && pyramid[i]<pyramid[2*i+1]
        || 2*i+1<n && pyramid[i]<pyramid[2*i+2])) {
        if (2*i+2<n && pyramid[2*i+1]<pyramid[2*i+2]) {
            swap(i,2*i+2);
            i = 2*i+2;
        } else {
            swap(i,2*i+1);
            i = 2*i+1;
        }
    }
}
```

# Реализация пирамидальной сортировки - 1

```
int mas[size];
int n_mas = 20;

void initArray() {
    for (int i=0; i<20; i++) {
        mas[i] = rand()%100;
    }
    n_mas = 20;
}

void displayArray() {
    for (int i=0; i<n_mas; i++) {
        cout.width(3);
        cout << mas[i];
    }
    cout << endl;
}
```

# Реализация пирамидальной сортировки - 2

```
void sort() {
    for (int i=0; i<n_mas; i++) {
        add(mas[i]);
    }
    for (int i=0; i<n_mas; i++) {
        mas[i] = top();
        pop();
    }
}
int main() {
    initPyramid(); initArray(); sort();
    displayArray(); return 0;
}
```

Всем спасибо за внимание!

:p