ITIVITI

LET IT TRADE

# Алгоритмы и структуры данных

Косяков Михаил Сергеевич
к.т.н., доцент кафедры ВТ

УНИВЕРСИТЕТ ИТМО

# Содержание курса

- Введение в теорию алгоритмов
- **Алгоритмы сортировок**
- Структуры данных
  - Линейные структуры
  - Бинарные деревья поиска
  - Хеши и хеш-функции
- Алгоритмы на графах
  - Обходы графов в ширину и глубину
  - Минимальные остовные деревья
  - Поиск кратчайших путей в графе

# Коэффициент ранговой корреляции Кендалла

ITIVITI

- С помощью сортировки и нехитрой математики Алиса легко сможет выбрать, с кем пойти в кино

| ID | Фильм | Алиса | Вася | Саша |
|---|---|---|---|---|
| **1** | Звездные войны | 6 | 5 | 1 |
| **2** | Терминатор | 5 | 6 | 2 |
| **3** | Аватар | 2 | 2 | 5 |
| **4** | Властелин колец | 3 | 1 | 4 |
| **5** | Гарри Поттер | 4 | 4 | 3 |
| **6** | Титаник | 1 | 3 | 6 |

$$\tau = \frac{\text{число пар в одном порядке} - \text{число инверсий}}{\text{число всех пар}}$$

LET IT TRADE



УНИВЕРСИТЕТ ИТМО

# Коэффициент ранговой корреляции Кендалла

- Отсортируем ID фильмов по предпочтениям Алисы

| ID | Фильм | Алиса | Вася (0.47) | Саша (-1) |
|---|---|---|---|---|
| **6** | Титаник | 1 | 3 | 6 |
| **3** | Аватар | 2 | 2 | 5 |
| **4** | Властелин колец | 3 | 1 | 4 |
| **5** | Гарри Поттер | 4 | 4 | 3 |
| **2** | Терминатор | 5 | 6 | 2 |
| **1** | Звездные войны | 6 | 5 | 1 |

- Сколько всего пар?
- Сколько инверсий у Васи? у Саши?

# Коэффициент ранговой корреляции Кендалла

ITIVITI

- Почему сумма модулей разности (квадратов разности не подходит)?

| **Фильм** | **Алиса** | **Вася** | **Саша** |
|---|---|---|---|
| Титаник | 1 | 2 | 3 |
| Аватар | 2 | 3 | 2 |
| Властелин колец | 3 | 1 | 1 |
| **Сумма модулей разности** | | **4** | **4** |

- … хотя Титаник Васе нравится больше Аватара, как и Алисе

LET IT TRADE

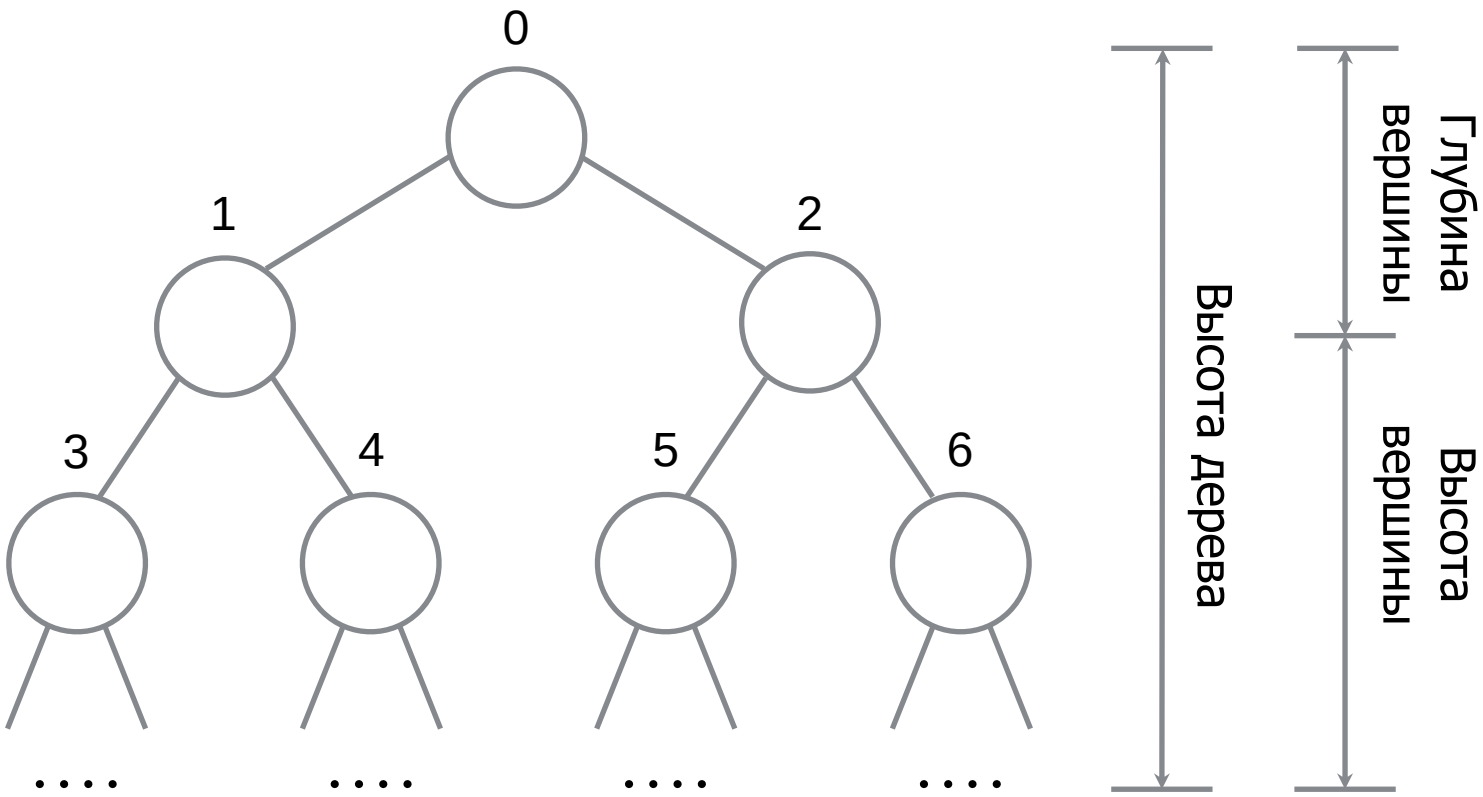


УНИВЕРСИТЕТ ИТМО

# Приоритетная очередь

- Работаем с множеством *S* из элементов *x*; с каждым элементом связан ключ *key*
- Поддерживаемые операции:
  - `Insert(S, x)`
  - `Maximum(S) / Minimum(S)`
  - `Extract_Max(S) / Extract_Min(S)`
  - `Increase_Key(S, x, key) / Decrease_Key(S, x, key)`
  - если несколько элементов с одинаковыми приоритетами, то при `Maximum()` и `Extract_Max()` достаем любой из них
- Обычная очередь – частный случай очереди с приоритетами, где приоритет – это время входа
- Реализация на (не)отсортированном массиве

# Почти полное двоичное дерево




- Корень, внутренние вершины, листья, родитель, предки, дети, потомки, братья

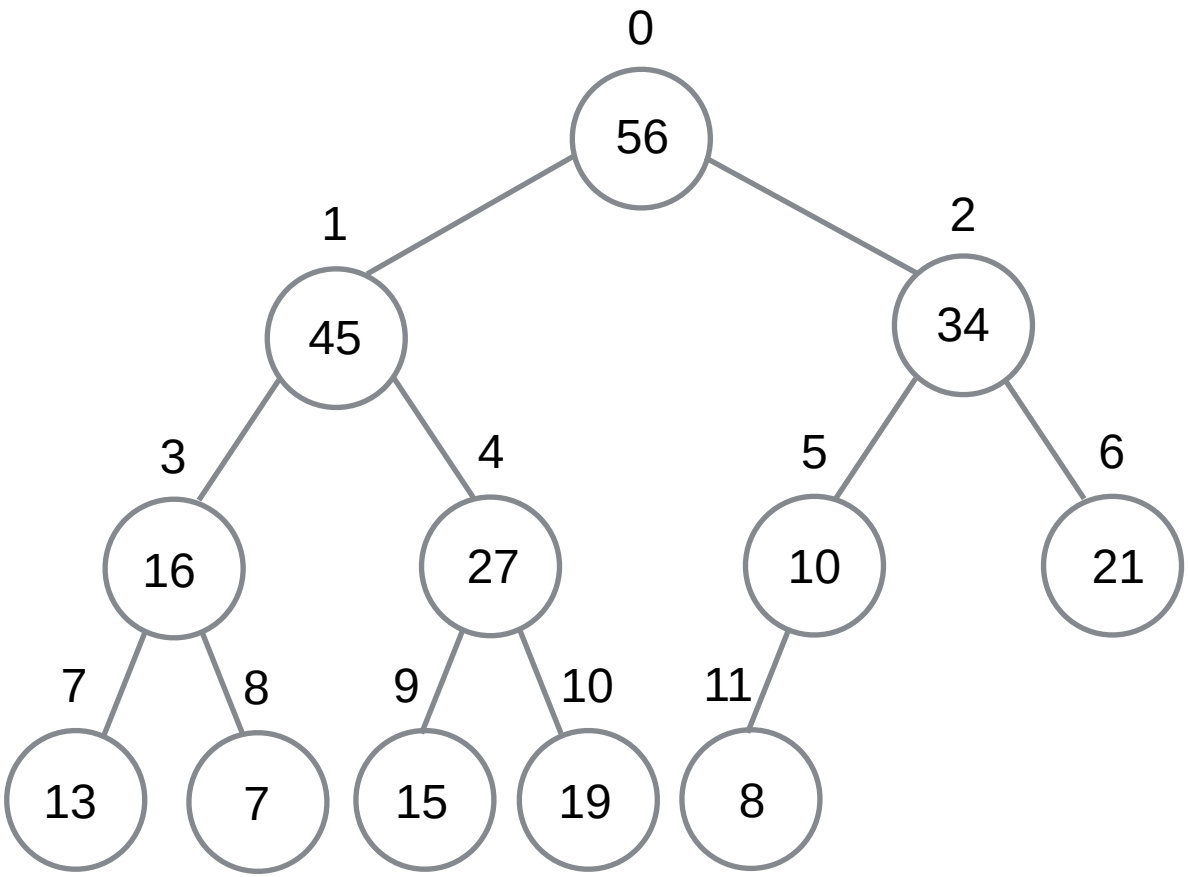
# Пирамида

- Высота $h$ пирамиды из $N$ элементов составляет $\lfloor \log_2 N \rfloor$: число вершин в слое $k$ (с глубиной $k$) не больше $2^k$;
  $2^h \le N \le 2^{h+1} - 1 \quad => \quad 2^h \le N < 2^{h+1}$
- Для вершины с номером $i$ номера детей и родителя
  - Левый ребенок: $2i + 1$
  - Правый ребенок: $2i + 2$

  Почему?
  - Родитель: $\lfloor (i - 1)/ 2 \rfloor$ - следует из правил для детей
- Следствие – удобно хранить в массиве $A[0 \ldots N-1]$
- Условие (невозрастающей) пирамиды:
  - $A[\text{ Родитель}(i)\ ] \ge A[i]$
- Следствие – в корне находится максимальный элемент

# Пирамида

- Пример пирамиды из 12 элементов:




| |
|---|
| 56 |
| 45 |
| 34 |
| 16 |
| 27 |
| 10 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 8 |

# Пирамида (пример операций)

ITIVITI

- Индексы родителя и детей:

```
1.  int parent(int i)
2.  {
3.      return (i - 1) / 2;
4.  }
5.
6.  int left(int i)
7.  {
8.      return 2 * i + 1;
9.  }
10.
11. int right(int i)
12. {
13.     return 2 * i + 2;
14. }
```

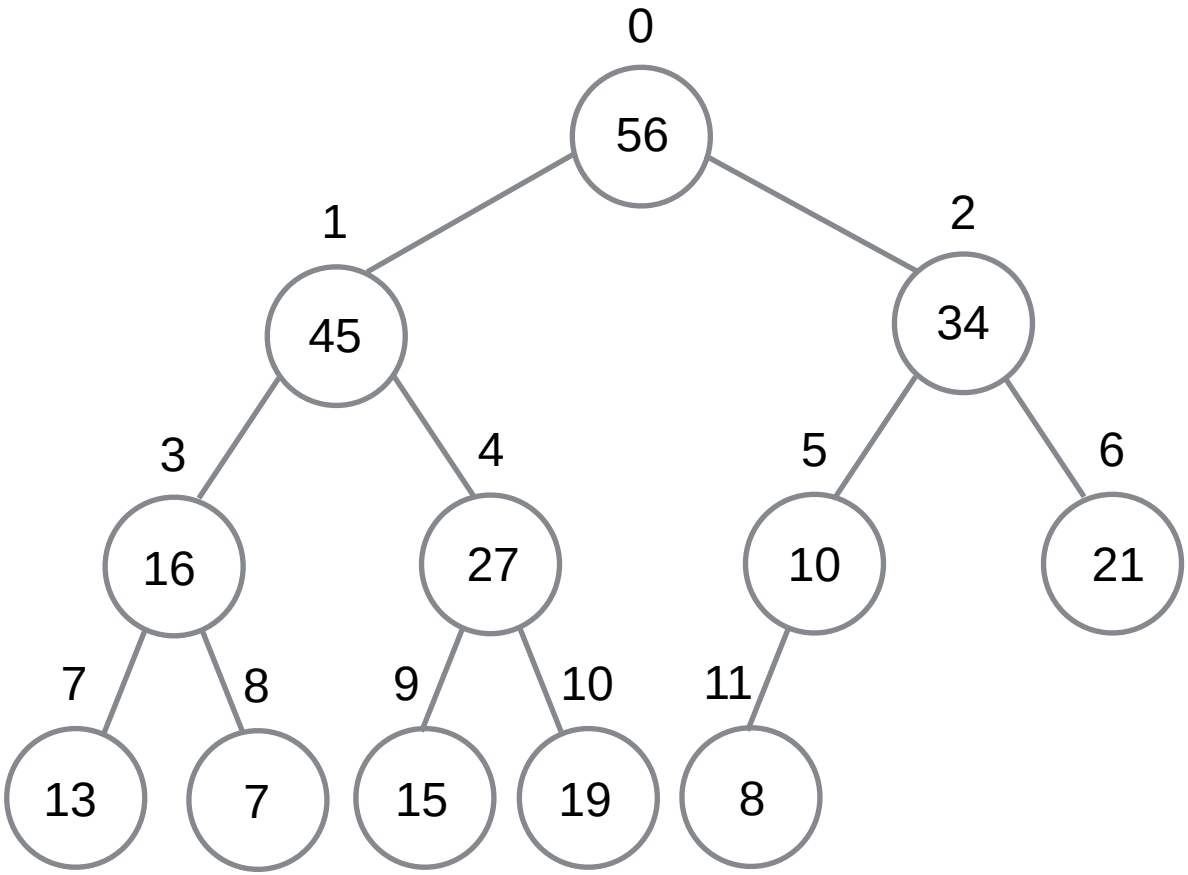
LET IT TRADE



УНИВЕРСИТЕТ ИТМО

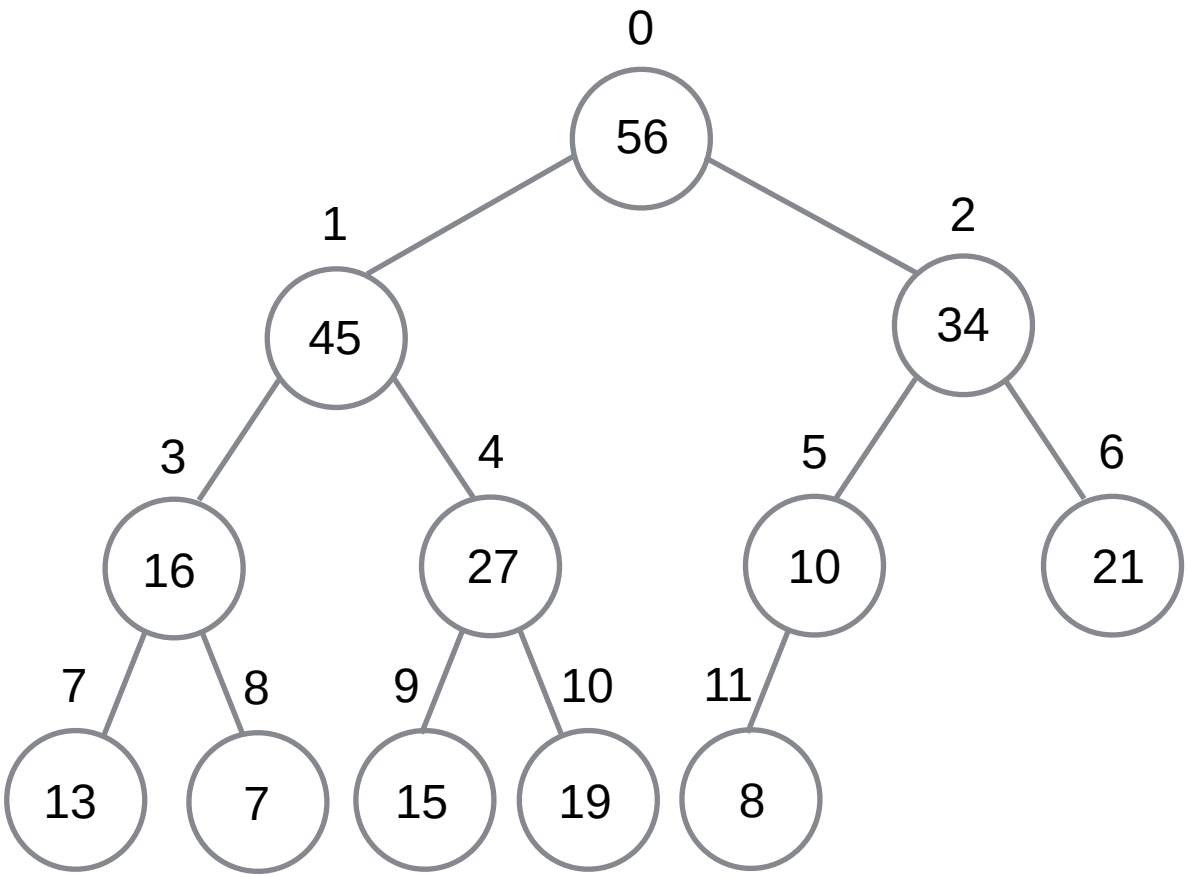
# Пирамида (пример операций)

- Пример операций: `Insert(50)`



# Пирамида (пример операций)

- Пример операций: `Insert(50)`

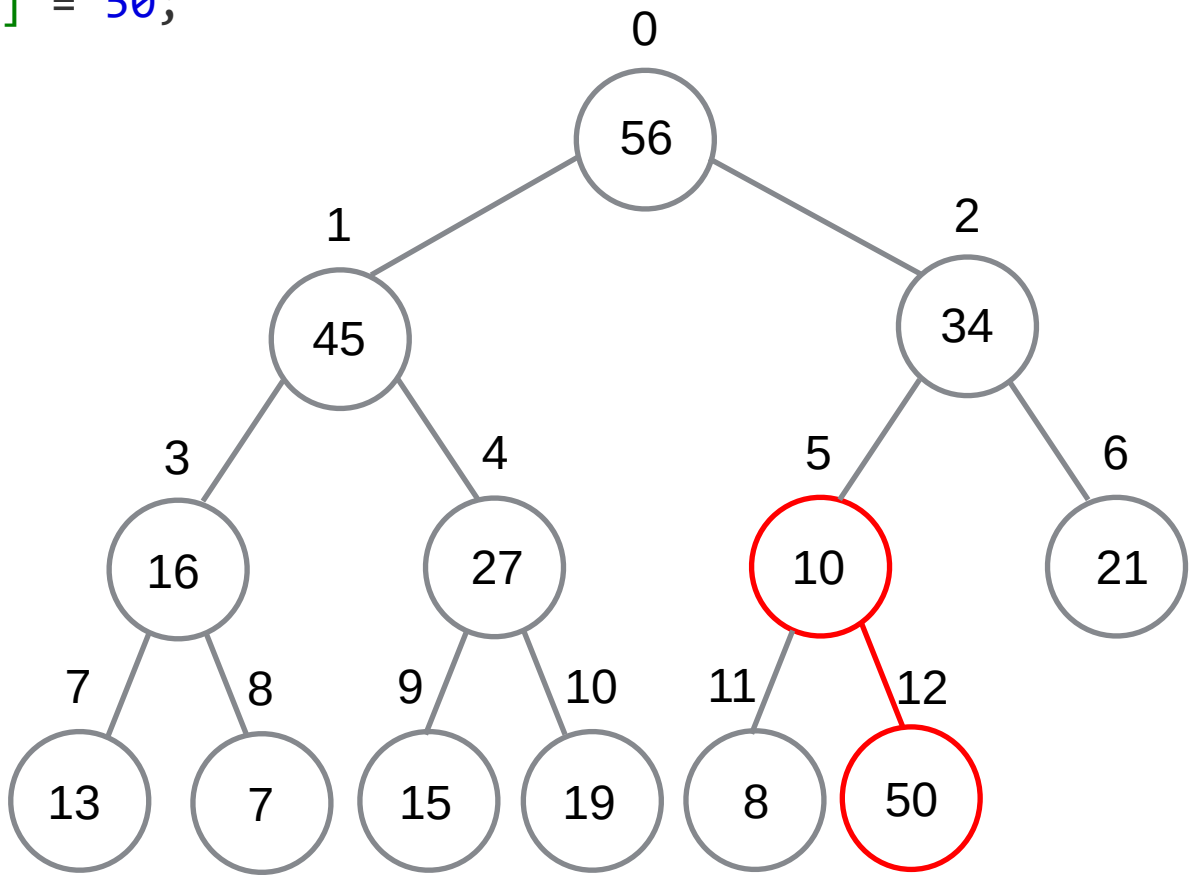



| 56 |
|---|
| 45 |
| 34 |
| 16 |
| 27 |
| 10 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 8 |

# Пирамида (просеивание вверх)

- Пример операций: `Insert(50)`

```
a[12] = 50;
```

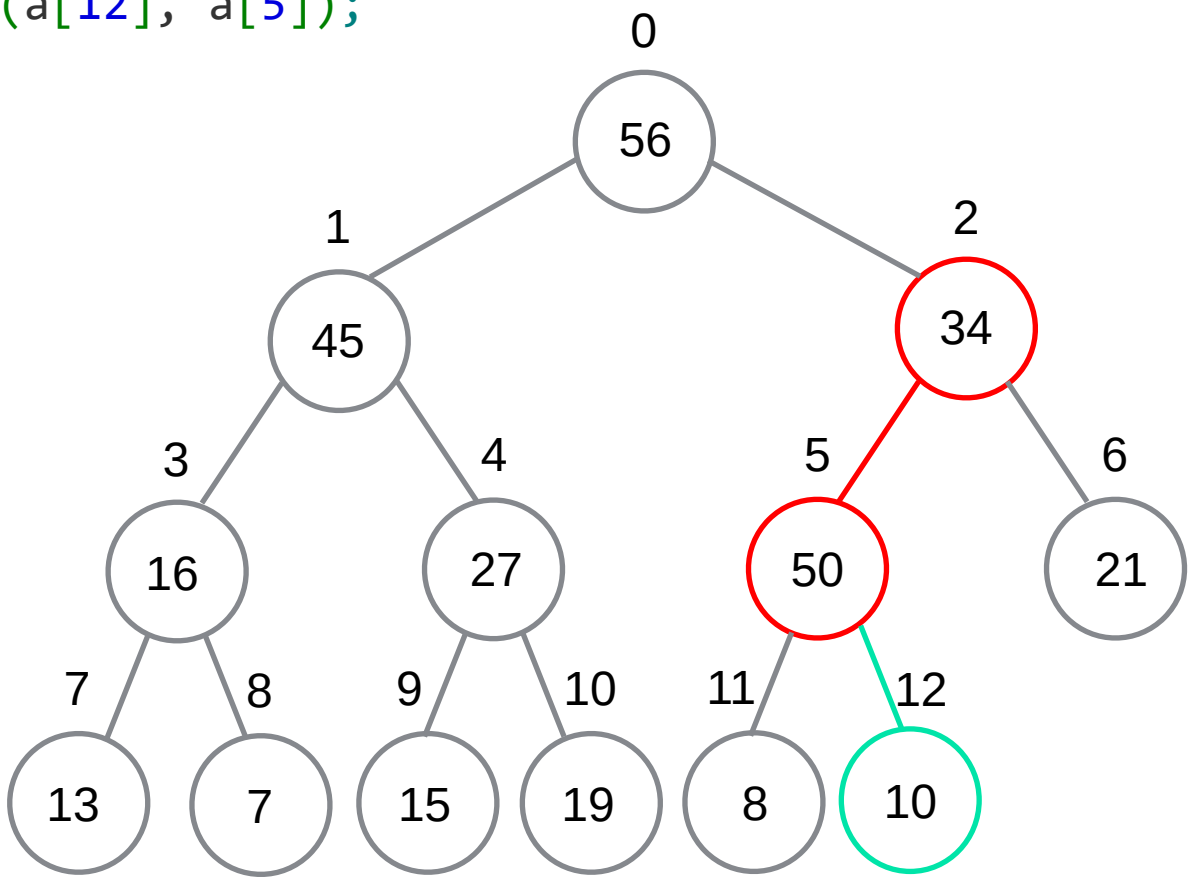



| |
|---|
| 56 |
| 45 |
| 34 |
| 16 |
| 27 |
| 10 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 8 |
| 50 |

# Пирамида (просеивание вверх)

- Пример операций: `Insert(50)`

```
swap(a[12], a[5]);
```

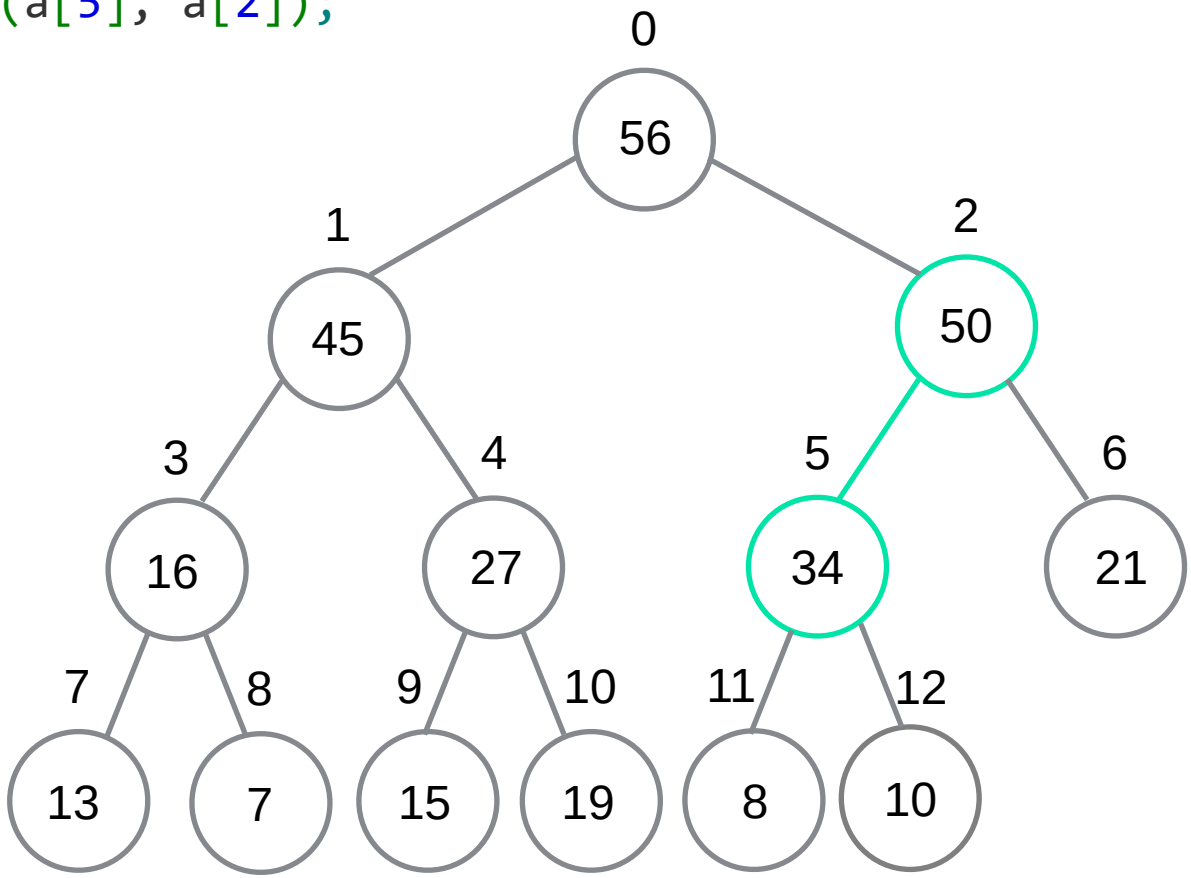



| |
|---|
| 56 |
| 45 |
| 34 |
| 16 |
| 27 |
| 50 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 8 |
| 10 |

# Пирамида (просеивание вверх)

- Пример операций: `Insert(50)`

```
swap(a[5], a[2]);
```




| 56 |
|---|
| 45 |
| 50 |
| 16 |
| 27 |
| 34 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 8 |
| 10 |

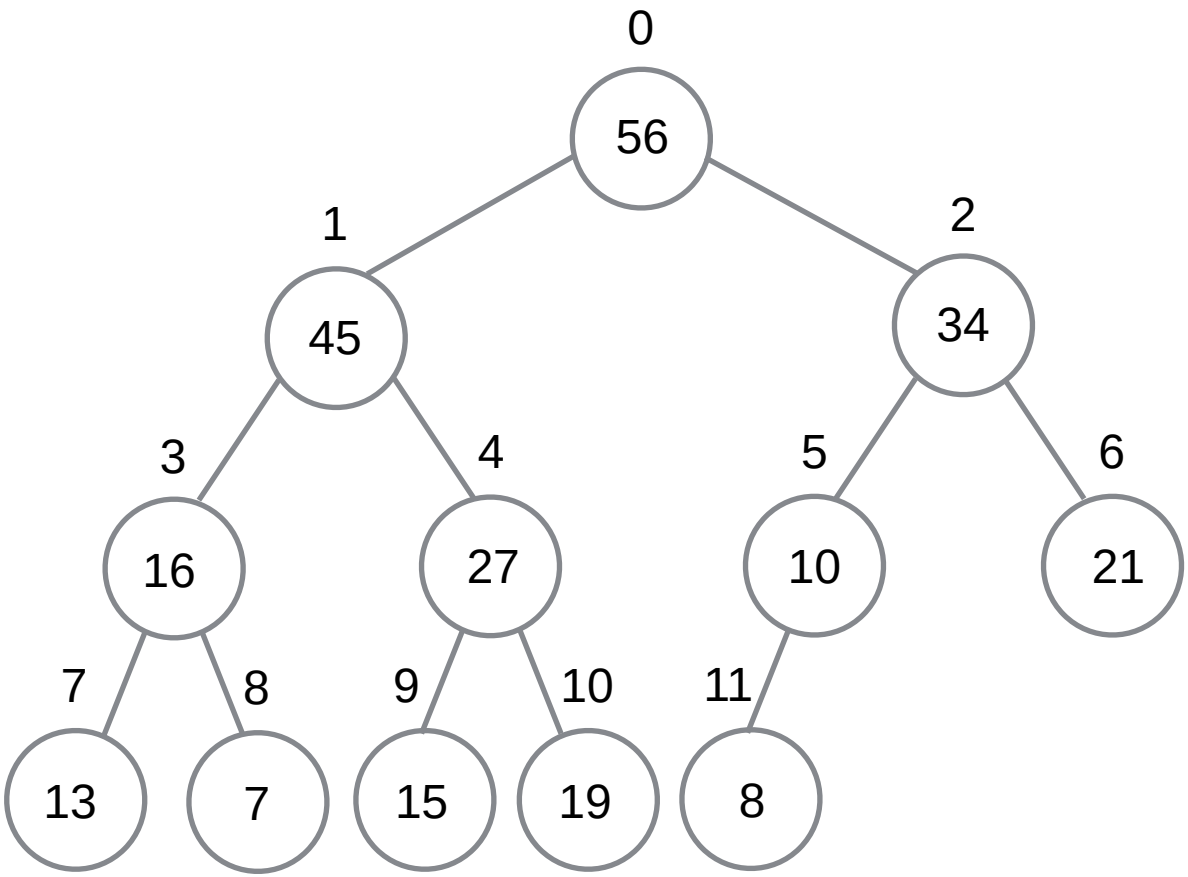
# Пирамида (просеивание вверх)

```
1. void siftup(int * a, int i)
2. {
3.     while (i > 0 && a[parent(i)] < a[i])
4.     {
5.         swap(a[i], a[parent(i)]);
6.         i = parent(i);
7.     }
8. }
9.
10. void insert(int * a, int & n, int x)
11. {
12.     a[n++] = x;
13.     siftup(a, n-1);
14. }
```

- Оценка сложности: $O(\log N)$
- Дополнительная память: $O(1)$

# Пирамида (пример операций)

- Пример операций: `Extract_Max()`

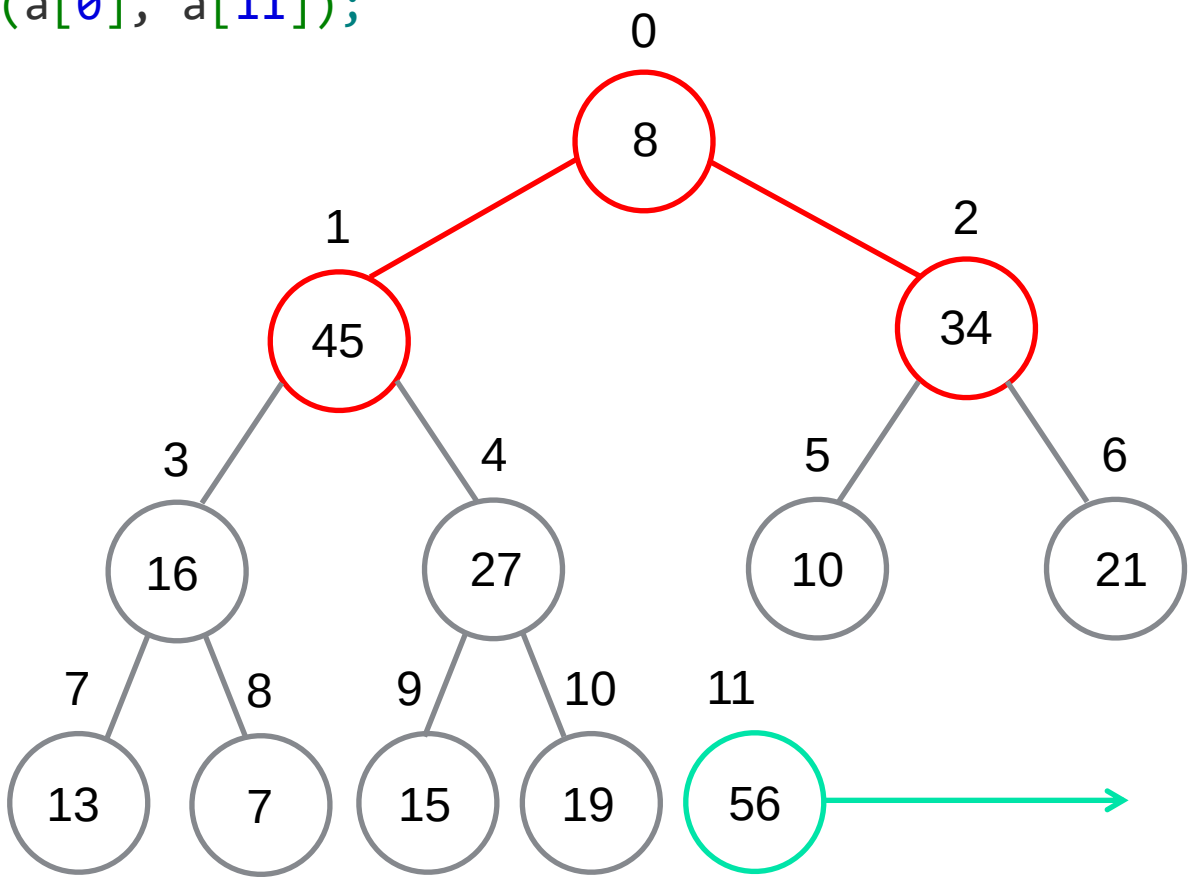



| 56 |
|---|
| 45 |
| 34 |
| 16 |
| 27 |
| 10 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 8 |

# Пирамида (просеивание вниз)

- Пример операций: `Extract_Max()`

```
swap(a[0], a[11]);
```

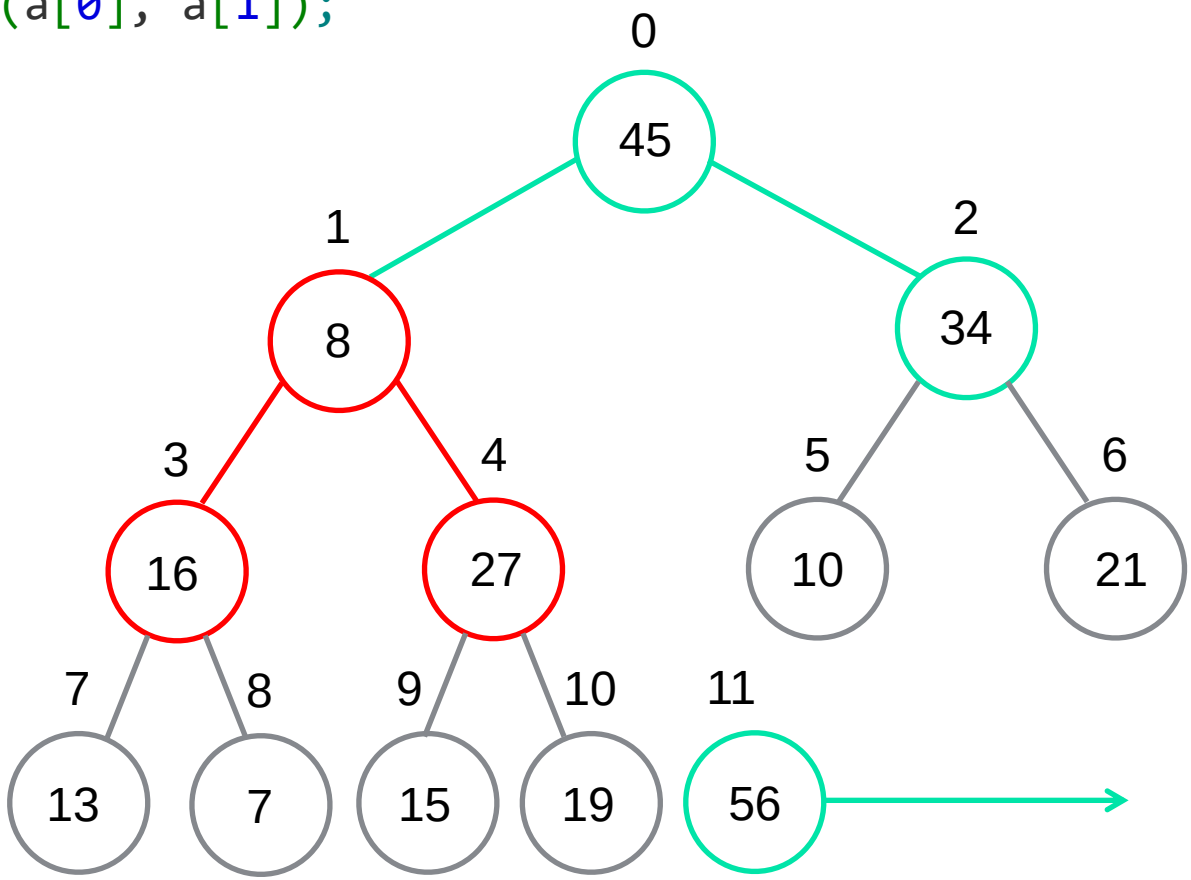


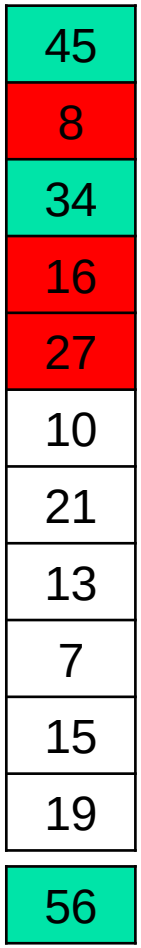

| |
|---|
| 8 |
| 45 |
| 34 |
| 16 |
| 27 |
| 10 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 56 |

# Пирамида (просеивание вниз)

- Пример операций: `Extract_Max()`

```
swap(a[0], a[1]);
```

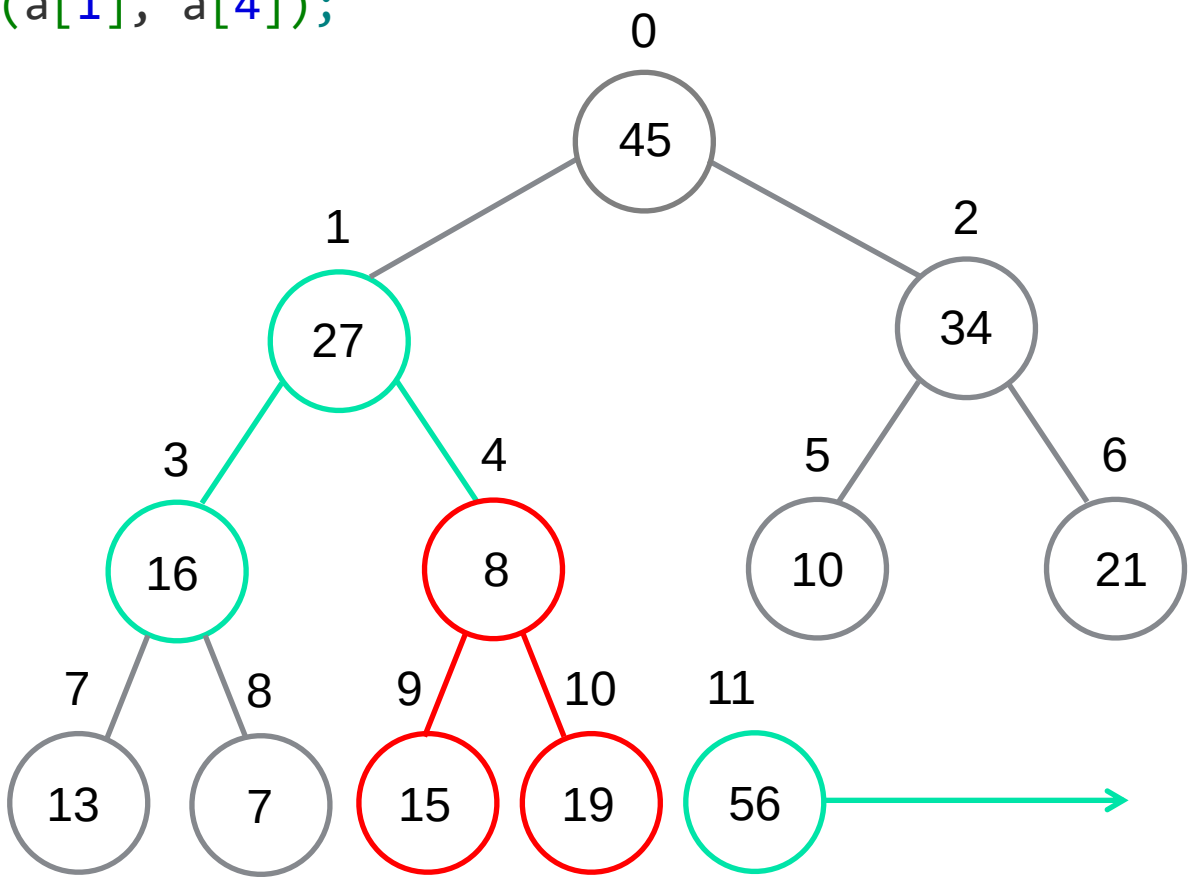


# Пирамида (просеивание вниз)

- Пример операций: `Extract_Max()`

```
swap(a[1], a[4]);
```

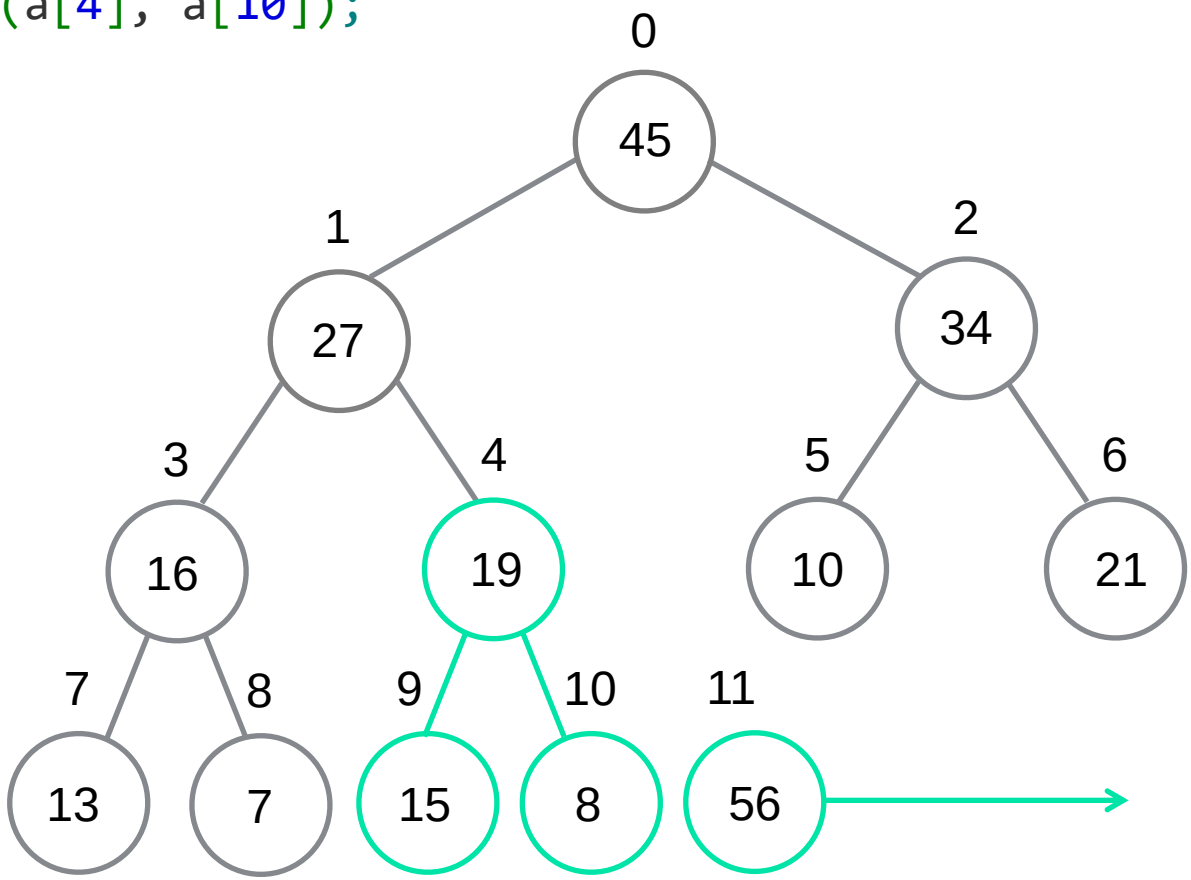



| |
|---|
| 45 |
| 27 |
| 34 |
| 16 |
| 8 |
| 10 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 56 |

# Пирамида (просеивание вниз)

- Пример операций: `Extract_Max()`

```
swap(a[4], a[10]);
```




| |
|---|
| 45 |
| 27 |
| 34 |
| 16 |
| 19 |
| 10 |
| 21 |
| 13 |
| 7 |
| 15 |
| 8 |
| 56 |

# Пирамида (просеивание вниз)

```
1.  void siftdown(int * a, int n, int i)
2.  {
3.      int j = i;
4.      if (left(i) < n && a[j] < a[left(i)]) {
5.          j = left(i);
6.      }
7.      if (right(i) < n && a[j] < a[right(i)]) {
8.          j = right(i);
9.      }
10.     if (i != j) {
11.         swap(a[i], a[j]);
12.         siftdown(a, n, j);
13.     }
14. }
```

- Оценка сложности: $O(\log N)$
- Дополнительная память: $O(\log N)$

# Концевая рекурсия (tail recursion)

ITIVITI

LET IT TRADE

```
1. // An example of tail recursive function
2. void print(int n) {
3.     if (n < 0) return;
4.     cout << " " << n;
5.
6.      // The last executed statement is recursive call
7.     print(n-1);
8. }
```

- Переиспользуем стековый фрейм вызывающей функции

https://www.geeksforgeeks.org/tail-call-elimination/



УНИВЕРСИТЕТ ИТМО

# Концевая рекурсия (tail recursion)

```
1. void print(int n) {
2. start:
3.     if (n < 0) return;
4.     cout << " " << n;
5.     // Update parameters of recursive call
6.     // and replace recursive call with goto
7.     n = n-1;
8.     goto start;
9. }
```

- Или чуть по-другому...

```
1. void print(int n) {
2.     while (n >= 0) {
3.         cout << " " << n;
4.         n = n-1;
5.     }
6. }
```

https://www.geeksforgeeks.org/tail-call-elimination/

# Пирамида (просеивание вниз) Концевая рекурсия


```
1.  void siftdown(int * a, int n, int i)
2.  {
3.      int j = i;
4.      while (true) {
5.        if (left(i) < n && a[j] < a[left(i)]) {
6.            j = left(i);
7.        }
8.        if (right(i) < n && a[j] < a[right(i)]) {
9.            j = right(i);
10.       }
11.       if (i == j) break;
12.       swap(a[i], a[j]);
13.       i = j;
14.     }
15. }
```

- Оценка сложности: $O(\log N)$, но чуть быстрее
- Дополнительная память: $O(1)$

# Пирамида (пример операций)

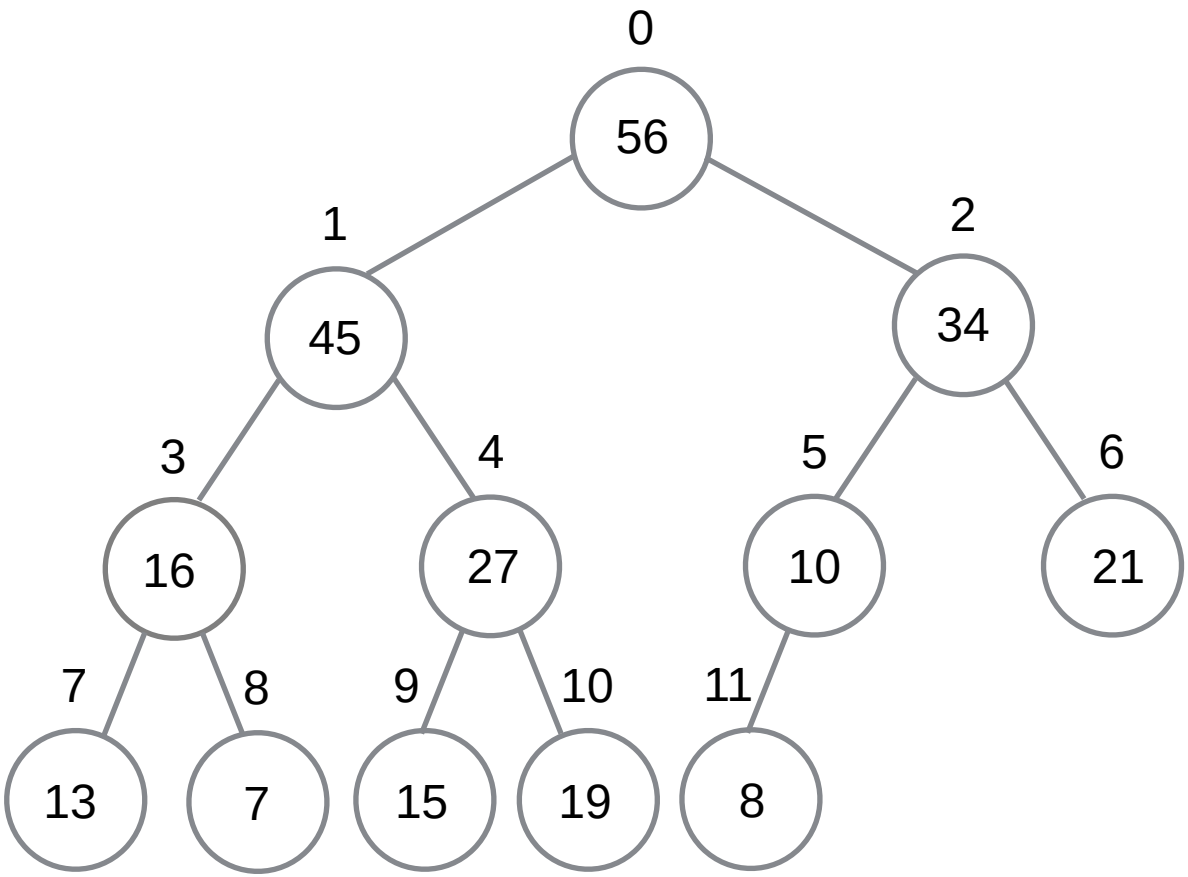
- Пример операций: `Extract_Max()`

```
1. int extract_max(int * a, int & n)
2. {
3.     swap(a[0], a[--n]);
4.     siftdown(a, n, 0);
5.     return a[n];
6. }
```

- Оценка сложности: $O(\log N)$
- Дополнительная память: $O(1)$, при нерекурсивном просеивании вниз

# Пирамида (пример операций)

- Пример операций: `Increase_Key(16, 77)`

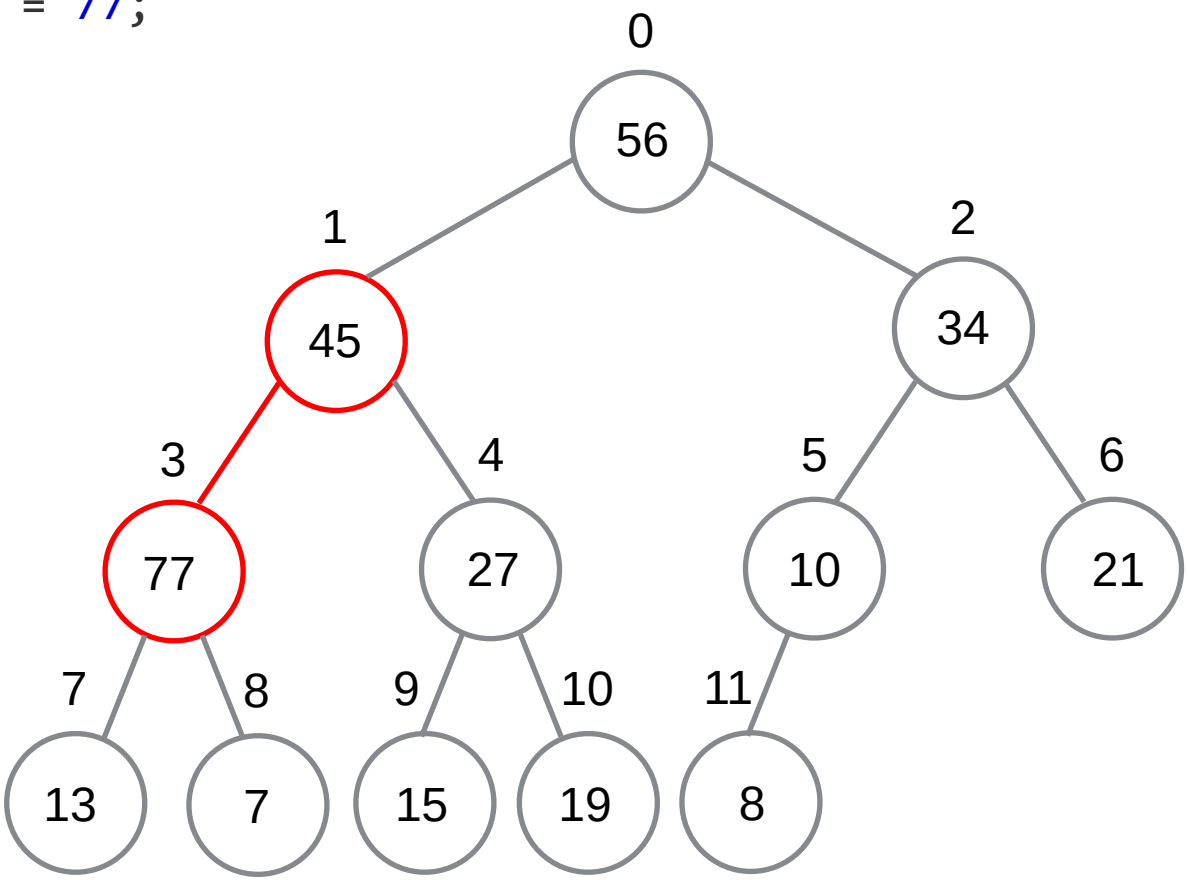



| 56 |
|---|
| 45 |
| 34 |
| 16 |
| 27 |
| 10 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 8 |

# Пирамида (пример операций)

- Пример операций: `Increase_Key(16, 77)`

```
a[3] = 77;
```

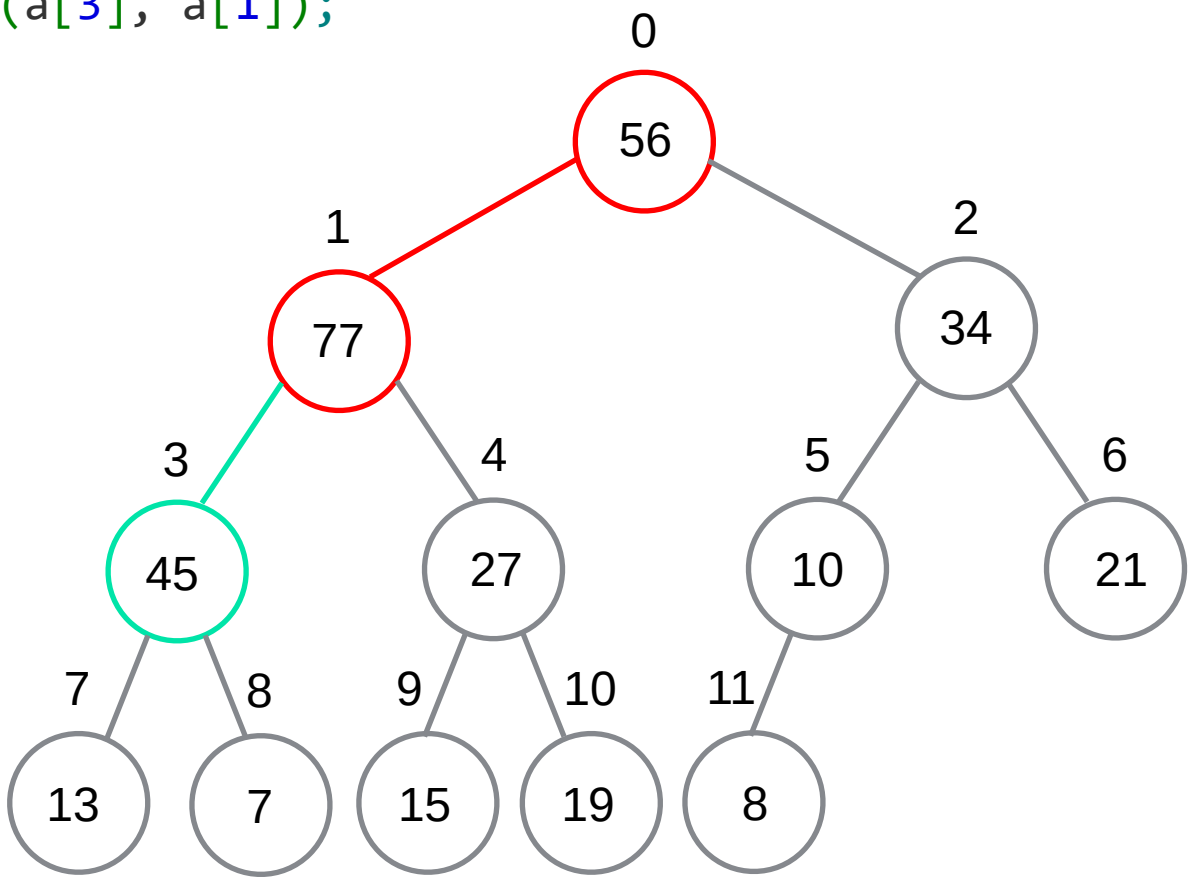



| |
|---|
| 56 |
| 45 |
| 34 |
| 77 |
| 27 |
| 10 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 8 |

# Пирамида (пример операций)

- Пример операций: `Increase_Key(16, 77)`

```
swap(a[3], a[1]);
```

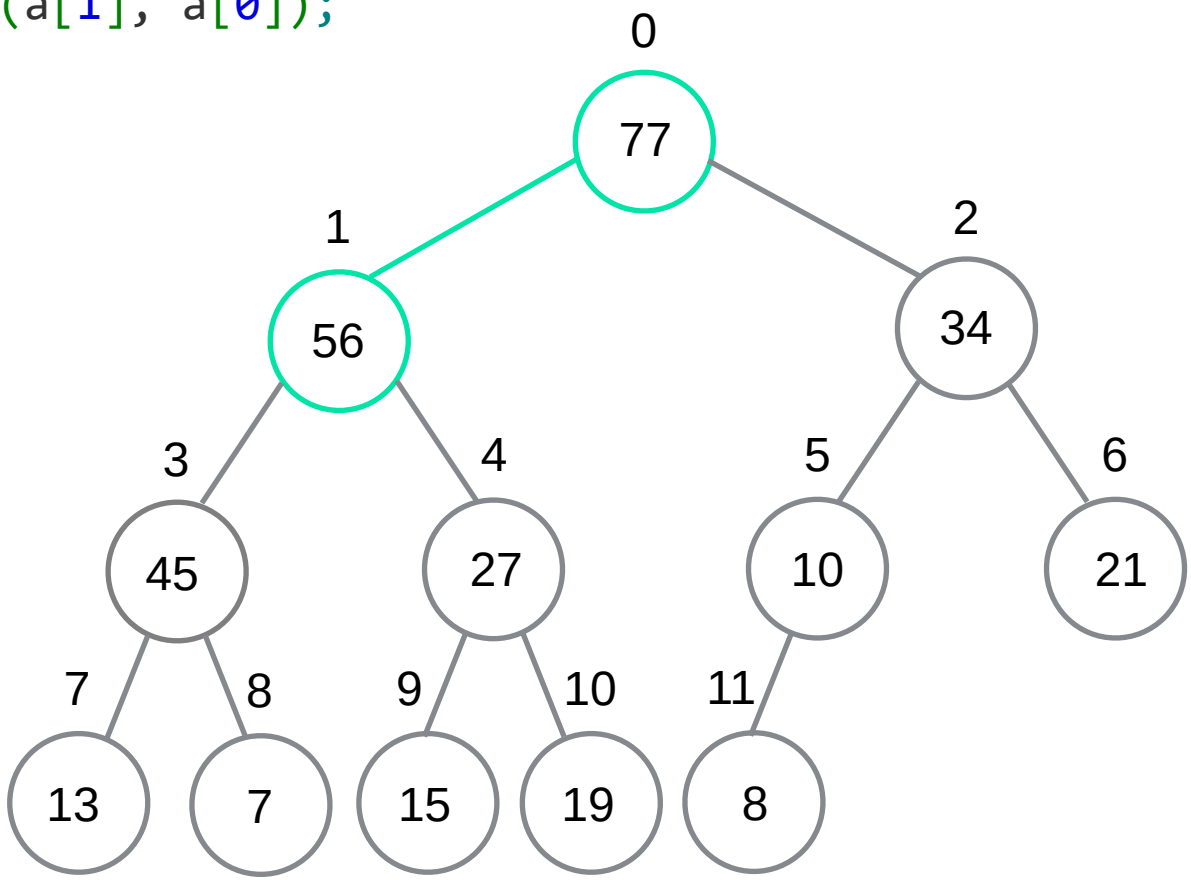



| |
|---|
| 56 |
| 77 |
| 34 |
| 45 |
| 27 |
| 10 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 8 |

# Пирамида (пример операций)

- Пример операций: `Increase_Key(16, 77)`

```
swap(a[1], a[0]);
```




| |
|---|
| 77 |
| 56 |
| 34 |
| 45 |
| 27 |
| 10 |
| 21 |
| 13 |
| 7 |
| 15 |
| 19 |
| 8 |

# Пирамида (пример операций)

ITIVITI

LET IT TRADE

```
1.  MaxVector<int> h;
2.  h.push_back(10);
3.  h.push_back(50);
4.  h.push_back(30);
5.  h.push_back(5);
6.  h[1] = h[3];
7.  h[3] = 100;
8.  h[3] = 20;
9.
10. for (int i = 0; i < h.size(); i++) cout << h[i] << " ";
11. cout << endl;
12. cout << "max = " << h.max() << endl;
```

- push_back() и индексация []: $O(\log N)$
- Операция max(): $O(1)$

# Пирамида (пример операций)

ITIVITI

LET IT TRADE

```
1.  template<typename T>
2.  class MaxVector {
3.  private:
4.    vector<T> m_vec; // элементы
5.    vector<size_t> m_heap; // элементы m_heap - индексы m_vec
6.    vector<size_t> m_inverted_index; // обратный индекс:
7.    // позволяет по номеру элемента найти его позицию в m_heap
8.    size_t m_n = 0;  // всего элементов
9.    ...
10. public:
11.   void push_back(const T el) {
12.     m_vec.push_back(el);
13.     m_heap.push_back(m_n);
14.     m_inverted_index.push_back(m_n);
15.     siftup(m_n++);
16.   }
17.   ...
```

# Пирамида (пример операций)

ITIVITI

LET IT TRADE

```
1. void siftup(size_t i) // i - номер в пирамиде m_heap
2. {
3.   assert(i == m_inverted_index[m_heap[i]]);
4.
5.   while (i > 0 && \
6.          m_vec[m_heap[parent(i)]] < m_vec[m_heap[i]])
7.   {
8.       swap(m_inverted_index[m_heap[i]], \
9.            m_inverted_index[m_heap[parent(i)]]);
10.
11.      swap(m_heap[i], m_heap[parent(i)]);
12.
13.      i = parent(i);
14.  }
15.}
```

# Пирамида (пример операций)

ITIVITI

LET IT TRADE

```
1.  template<typename T>
2.  class MaxVector {
3.  private:
4.    vector<T> m_vec; // элементы
5.    vector<size_t> m_heap; // элементы m_heap - индексы m_vec
6.    vector<size_t> m_inverted_index; // обратный индекс:
7.    // позволяет по номеру элемента найти его позицию в m_heap
8.    size_t m_n = 0;  // всего элементов
9.    ...
10. public:
11.   Proxy operator[] (const int index) {
12.     assert(index == m_heap[m_inverted_index[index]]);
13.     return Proxy(*this, index);
14.   }
15.   T max() {
16.     if (m_n > 0) return m_vec[m_heap[0]]; else return T();
17.   }
```

# Пирамида (пример операций)

ITIVITI

LET IT TRADE

```
1.  class Proxy {
2.  private:
3.    MaxVector& m_parent;
4.    const size_t m_index;
5.  public:
6.    T& operator = (const T new_el) {
7.       T& el = m_parent.m_vec[m_index];
8.       if (new_el == el) return el;
9.       T tmp = el;
10.      el = new_el;
11.      if (new_el > tmp) \
12.        m_parent.siftup(m_parent.m_inverted_index[m_index]);
13.      if (new_el < tmp) \
14.        m_parent.siftdown(m_parent.m_inverted_index[m_index]);
15.      return el;
16.   }
17.   ...
```

# Пирамида (пример операций)

ITIVITI

LET IT TRADE

```
1. class Proxy {
2. private:
3.   MaxVector& m_parent;
4.   const size_t m_index;
5. public:
6.   Proxy& operator = (const Proxy& p) {
7.     if (this == &p) return *this;
8.     this->operator= \
9.       (static_cast<T>(p.m_parent.m_vec[p.m_index]));
10.    return *this;
11.  }
12.  operator T() const { return m_parent.m_vec[m_index]; }
13.};
```

# Построение пирамиды

- Пусть есть массив $A[0 \ldots N-1]$; как построить пирамиду?
- Просеиванием вверх (последовательность insert-ов)
  - $T(N) = O(\log N) + T(N-1)$
  - Сложность: $O(N \log N)$

```
1. void build_heap_insert(int * a, int n)
2. {
3.     int sz = 0;
4.     while(sz < n) {
5.         insert(a, sz, a[sz]);
6.     }
7. }
```

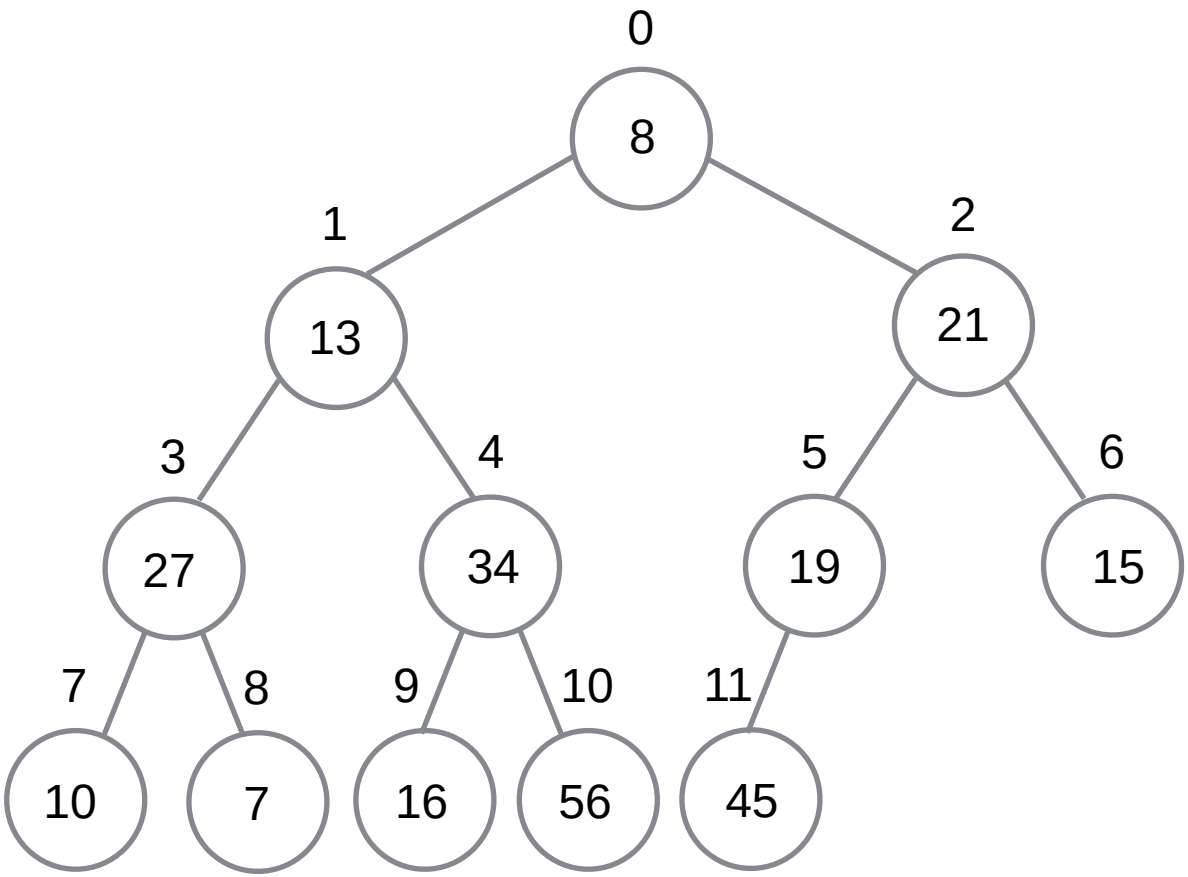
# Построение пирамиды

- Просеиванием вниз корня каждого поддерева
  - На последнем уровне не более $2^h$ элементов (их не надо просеивать вниз); на предпоследнем уровне $2^{h-1}$ элементов (они просеятся на один уровень вниз); на препредпоследнем уровне $2^{h-2}$ элементов (они просеятся на два уровня вниз) и т.д.
  - $T(N) = \sum_{j=0}^{h} j 2^{h-j} = \sum_{j=0}^{h} j \frac{2^h}{2^j} = 2^h \sum_{j=0}^{h} \frac{j}{2^j} \le 2^h \sum_{j=0}^{\infty} \frac{j}{2^j} = 2^h \cdot 2 = 2^{h+1}$
  - Сложность: $O(N)$, потому что $\sum_{i=0}^{\infty} i x^i = \frac{x}{(1-x)^2}$ для $x < 1$

```
1. void build_heap(int * a, int n)
2. {
3.    for(int i = n / 2 - 1; i >= 0; --i) {
4.    siftdown(a, n, i);
5.    }
6. }
```

# Построение пирамиды

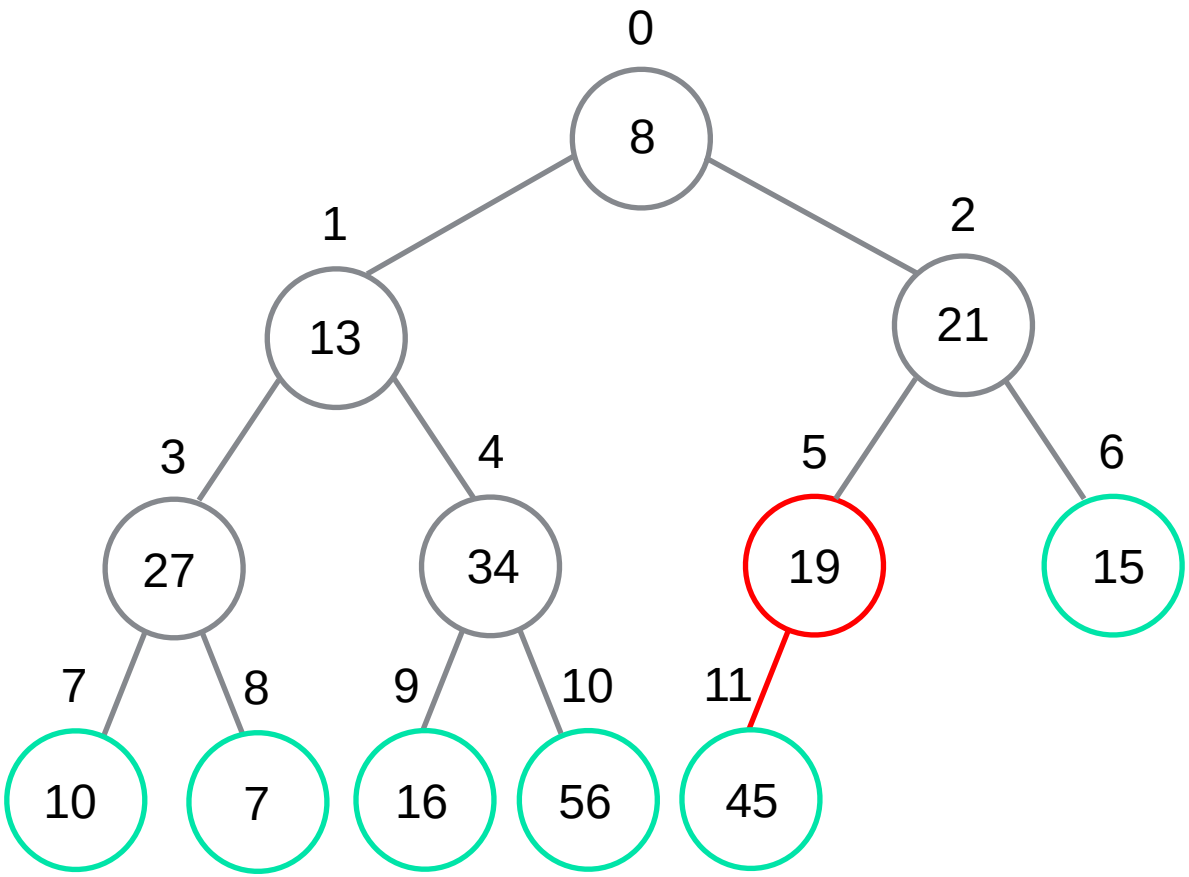
- Просеивание вниз корня каждого поддерева




| 8 |
|---|
| 13 |
| 21 |
| 27 |
| 34 |
| 19 |
| 15 |
| 10 |
| 7 |
| 16 |
| 56 |
| 45 |

# Построение пирамиды

- Просеивание вниз корня каждого поддерева

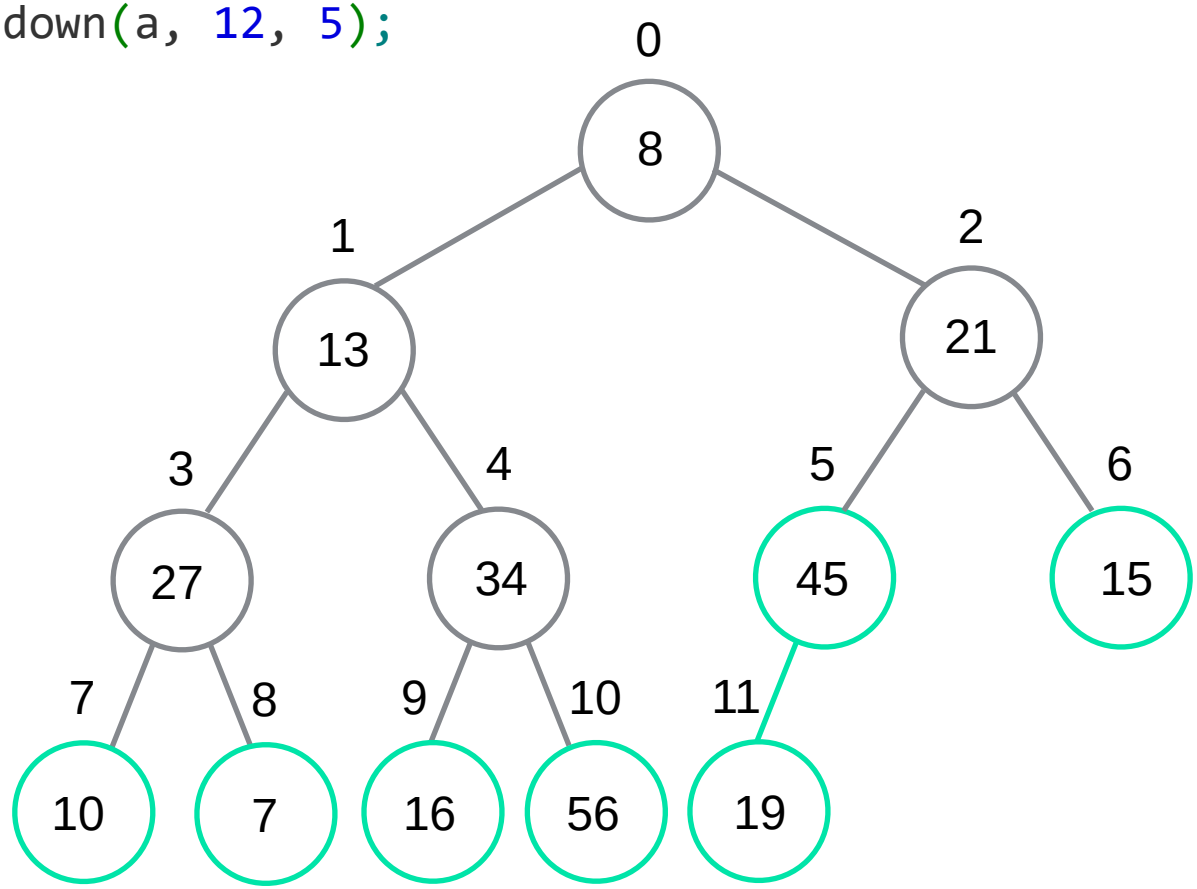



| 8 |
|---|
| 13 |
| 21 |
| 27 |
| 34 |
| 19 |
| 15 |
| 10 |
| 7 |
| 16 |
| 56 |
| 45 |

# Построение пирамиды

- Просеивание вниз корня каждого поддерева

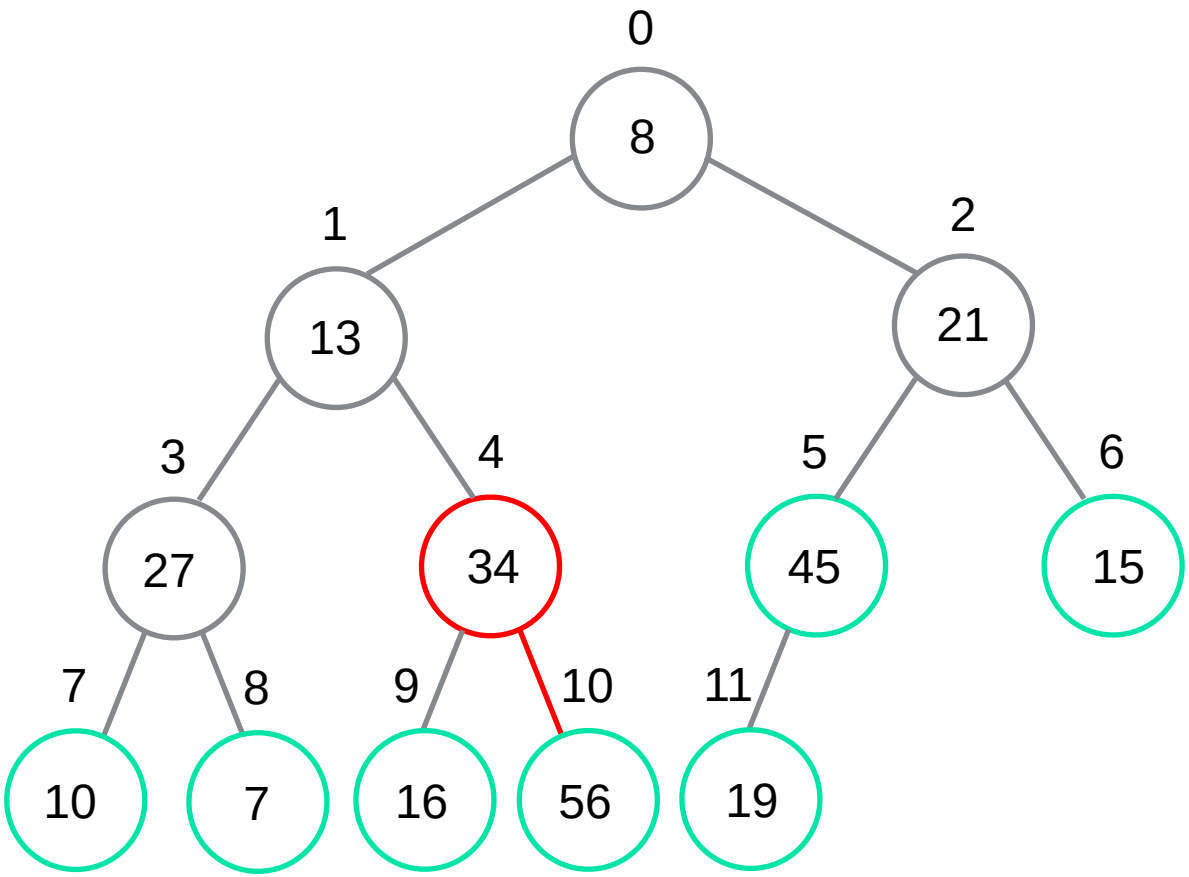
```
siftdown(a, 12, 5);
```




| |
|---|
| 8 |
| 13 |
| 21 |
| 27 |
| 34 |
| 45 |
| 15 |
| 10 |
| 7 |
| 16 |
| 56 |
| 19 |

# Построение пирамиды

- Просеивание вниз корня каждого поддерева

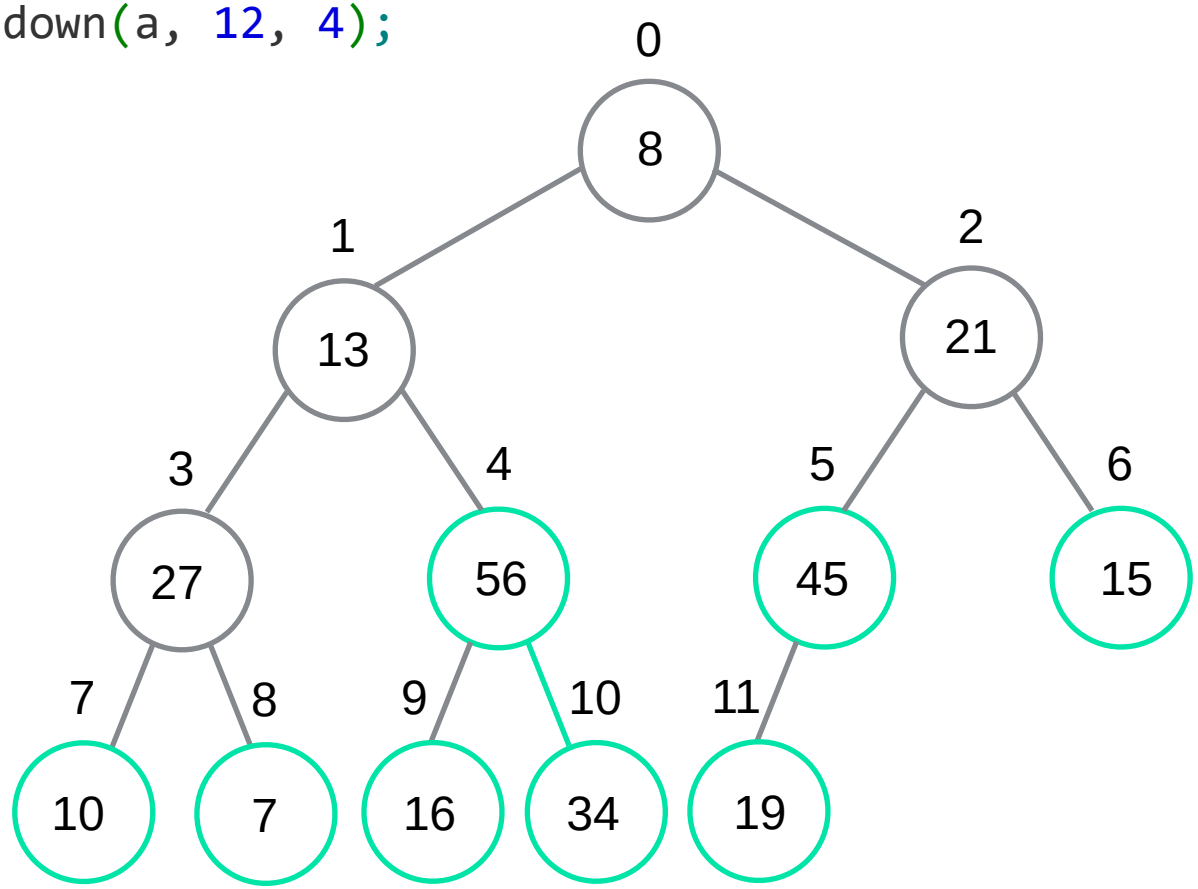



| 8 |
|---|
| 13 |
| 21 |
| 27 |
| 34 |
| 45 |
| 15 |
| 10 |
| 7 |
| 16 |
| 56 |
| 19 |

# Построение пирамиды

- Просеивание вниз корня каждого поддерева

```
siftdown(a, 12, 4);
```

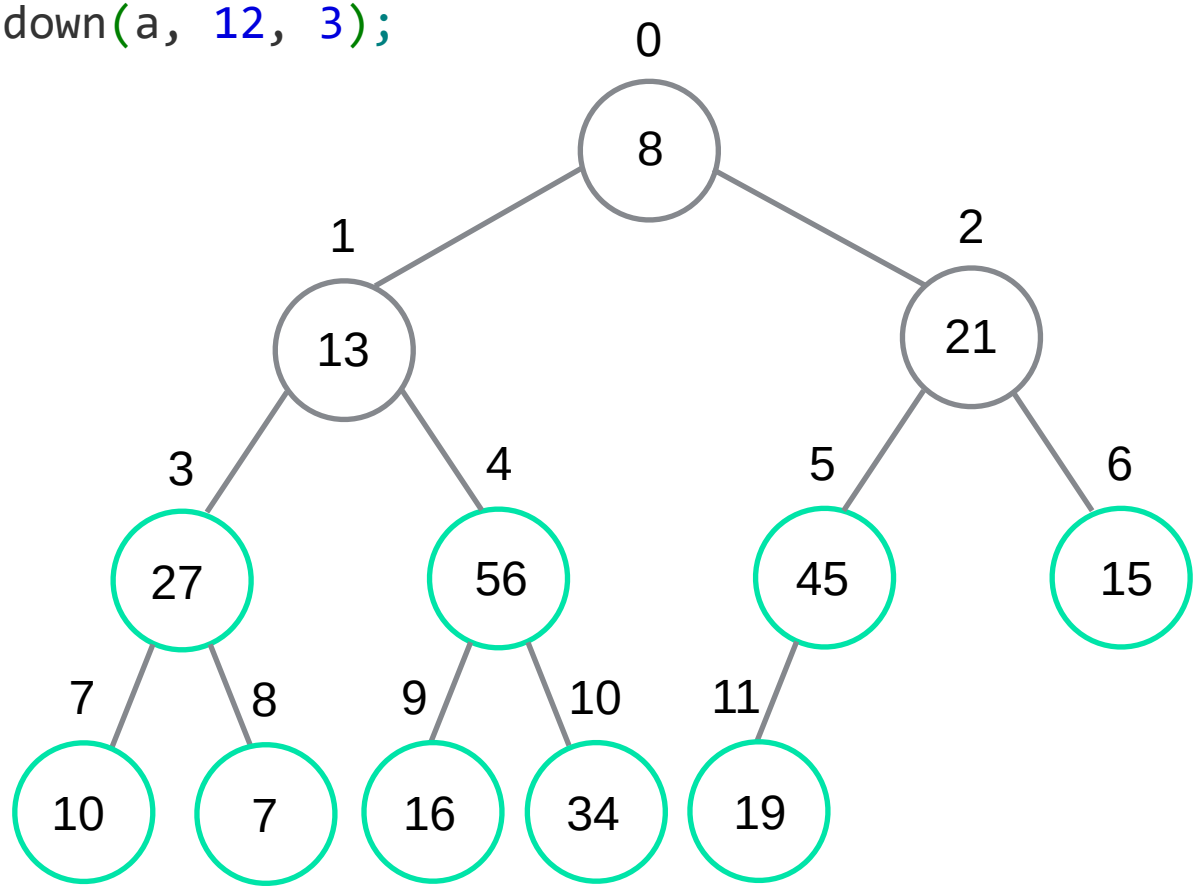



| 8 |
|---|
| 13 |
| 21 |
| 27 |
| 56 |
| 45 |
| 15 |
| 10 |
| 7 |
| 16 |
| 34 |
| 19 |

# Построение пирамиды

- Просеивание вниз корня каждого поддерева

```
siftdown(a, 12, 3);
```

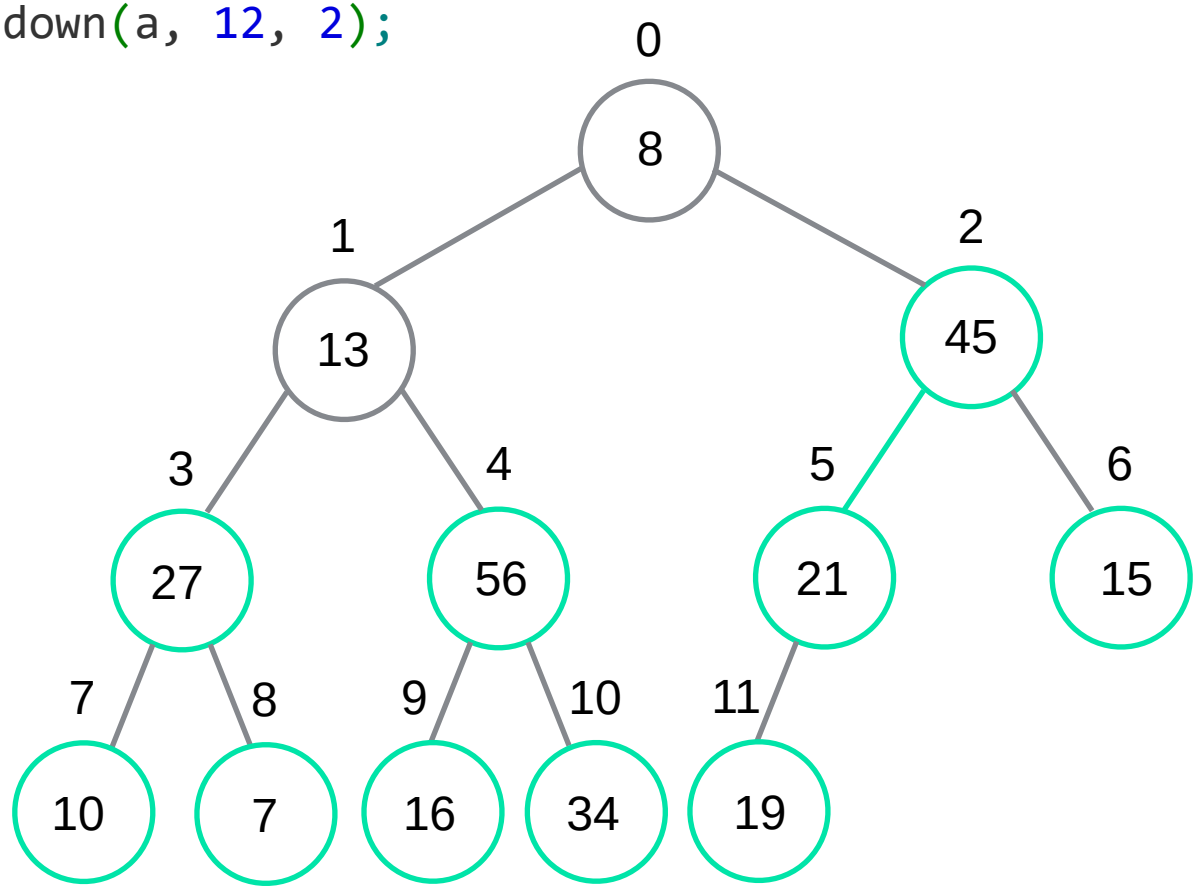



| |
|---|
| 8 |
| 13 |
| 21 |
| 27 |
| 56 |
| 45 |
| 15 |
| 10 |
| 7 |
| 16 |
| 34 |
| 19 |

# Построение пирамиды

- Просеивание вниз корня каждого поддерева

```
siftdown(a, 12, 2);
```

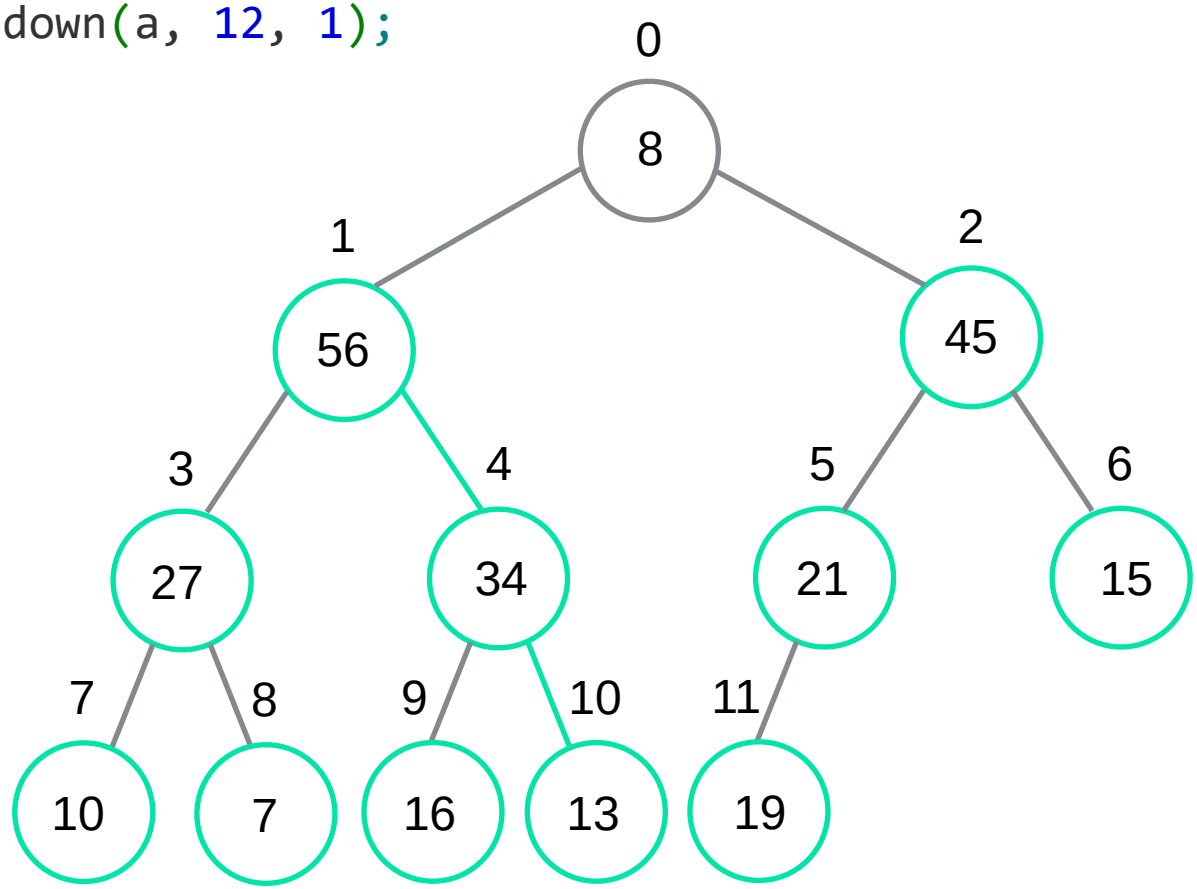



| 8 |
|---|
| 13 |
| 45 |
| 27 |
| 56 |
| 21 |
| 15 |
| 10 |
| 7 |
| 16 |
| 34 |
| 19 |

# Построение пирамиды

- Просеивание вниз корня каждого поддерева

```
siftdown(a, 12, 1);
```

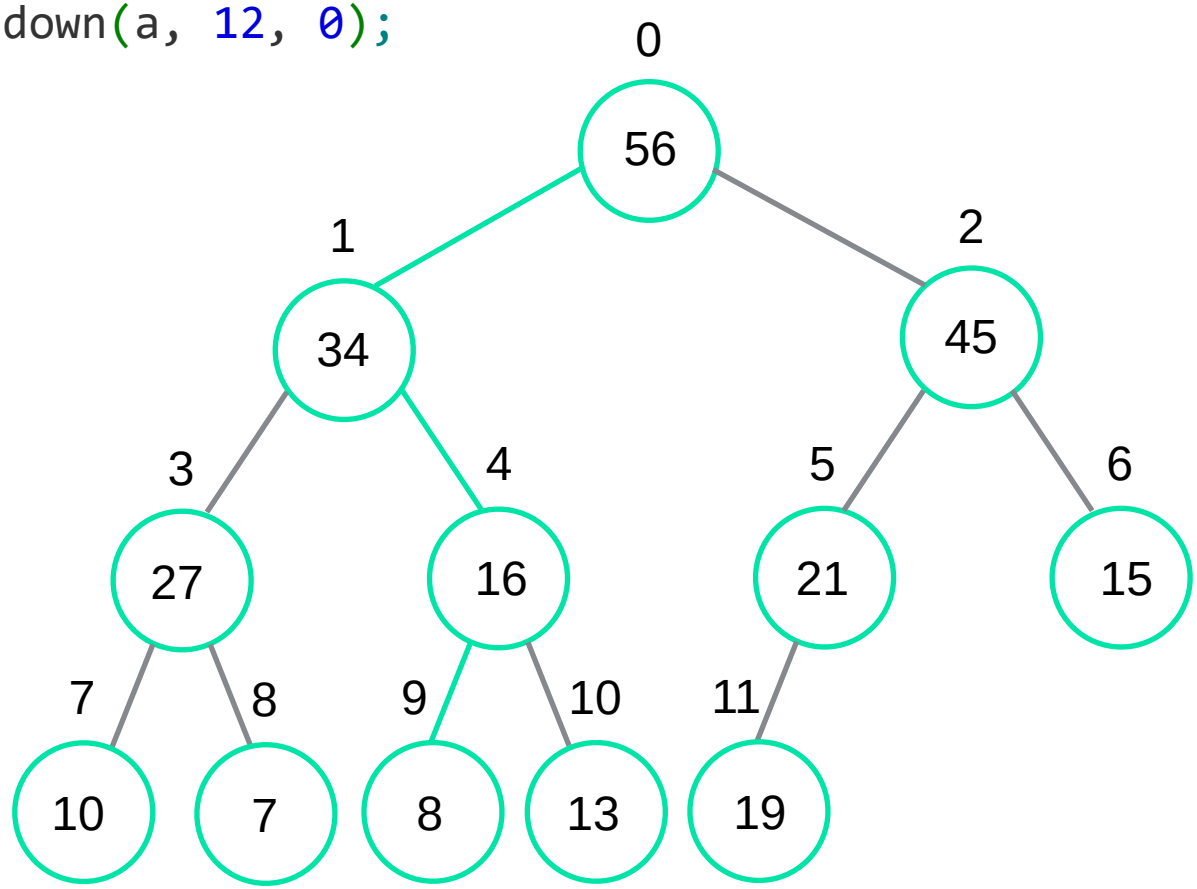



| 8 |
|---|
| 56 |
| 45 |
| 27 |
| 34 |
| 21 |
| 15 |
| 10 |
| 7 |
| 16 |
| 13 |
| 19 |

# Построение пирамиды

- Просеивание вниз корня каждого поддерева

```
siftdown(a, 12, 0);
```




| 56 |
|---|
| 34 |
| 45 |
| 27 |
| 16 |
| 21 |
| 15 |
| 10 |
| 7 |
| 8 |
| 13 |
| 19 |

# Сортировка пирамидой

- Неубывающей пирамидой – два массива (heap и target)
- Невозрастающей пирамидой – in place сортировка

```
1. void heap_sort(int * a, int n)
2. {
3.     build_heap(a, n);
4.     for(int l = n - 1; l > 0; --l) {
5.         swap(a[0], a[l]);
6.         siftdown(a, l, 0);
7.     }
8. }
```

- Оценка сложности: $O(N \log N)$
- Дополнительная память: $O(1)$

ITIVITI

LET IT TRADE

# Спасибо за внимание!



УНИВЕРСИТЕТ ИТМО