Основы программного конструирования

ЛЕКЦИЯ №3

27 ФЕВРАЛЯ 2023

Сортировка Шелла

1. Выбираем длину промежутка .
2. Разбиваем массив на d подмассивов:
3. Каждый подмассив сортируем сортировкой вставками
4. Если то уменьшаем и переходим на шаг 2. Иначе массив отсортирован.

Сортировка Шелла

37	5	10	12	9	15	23	8	22	32	1	12	15

Сортировка Шелла

37	5	10	12	9	15	23	8	22	32	1	12	15

9	5	10	12	15	15	23	8	22	32	1	12	37

Сортировка Шелла

37	5	10	12	9	15	23	8	22	32	1	12	15

9	5	10	12	15	15	23	8	22	32	1	12	37

9	5	1	12	15	15	10	8	22	32	23	12	37

Сортировка Шелла

37	5	10	12	9	15	23	8	22	32	1	12	15
9	$\boxed{5}$	10	12	15	$\boxed{15}$	23	8	22	$\boxed{32}$	1	12	37
9	5	$\boxed{1}$	12	15	15	$\boxed{10}$	8	22	32	$\boxed{23}$	12	37
9	5	1	$\boxed{8}$	15	15	10	$\boxed{12}$	22	32	23	$\boxed{12}$	37

Сортировка Шелла

37	5	10	12	9	15	23	8	22	32	1	12	15
9	5	10	12	15	15	23	8	22	32	1	12	37
9	5	1	12	15	15	10	8	22	32	23	12	37
9	5	1	8	15	15	10	12	22	32	23	12	37
9	5	1	8	15	15	10	12	22	32	23	12	37

Сортировка Шелла

9	5	1	8	15	15	10	12	22	32	23	12	37

9	5	1	8	15	15	10	12	22	32	23	12	37

Сортировка Шелла

9	5	1	8	15	15	10	12	22	32	23	12	37
9	5	1	8	15	15	10	12	22	32	23	12	37
1	5	9	8	10	15	15	12	22	32	23	12	37

Сортировка Шелла

9	5	1	8	15	15	10	12	22	32	23	12	37
9	5	1	8	15	15	10	12	22	32	23	12	37
1	5	9	8	10	15	15	12	22	32	23	12	37
1	5	9	8	10	12	15	12	22	15	23	32	37

Сортировка Шелла

9	5	1	8	15	15	10	12	22	32	23	12	37
9	5	1	8	15	15	10	12	22	32	23	12	37
1	5	9	8	10	15	15	12	22	32	23	12	37
1	5	9	8	10	12	15	12	22	15	23	32	37
1	5	9	8	10	12	15	12	22	15	23	32	37

Сортировка Шелла

1	5	9	8	10	12	15	12	22	15	23	32	37

1	5	8	9	10	12	12	15	15	22	23	32	37

Сортировка Шелла

1	5	9	8	10	12	15	12	22	15	23	32	37

1	5	8	9	10	12	12	15	15	22	23	32	37

1	5	8	9	10	12	12	15	15	22	23	32	37

Сортировка Шелла

ТУТ БЫЛА АНИМАЦИЯ, НО Я ЕЕ ПОТЕРЯЛ?

Анализ сортировки Шелла

Быстродействие зависит от выбора длин промежутков.

- Шелл:
- Хиббард:
- Пратт:
- Седжвик: нетривиальные последовательности

«Разделяй и властвуй»

- **Разделение:** задачи на несколько подзадач.
- **Покорение:** решение подзадач.
- **Комбинирование:** решения исходной задачи из решений подзадач.

Быстрая сортировка «Quicksort»

- **Разделение:** Массив путем переупорядочения элементов разбивается на две части и При этом запись «стоит на своем месте» т.е. все ключи левой части не больше , а все ключи правой части – не меньше.

- **Покорение:** Процедура Quicksort вызывается рекурсивно для левой и правой частей.

- **Комбинирование:** Не требуется.

Процедура разделения

4	3	8	2	7	6	5	0	1	9

Процедура разделения

4	3	8	2	7	6	5	0	1	9

Процедура разделения

4	**3**	8	2	7	6	5	0	1	**9**

Процедура разделения

4	3	**8**	2	7	6	5	0	**1**	9

Процедура разделения

4	3	**1**	2	7	6	5	0	**8**	9

Процедура разделения

4	3	1	**2**	7	6	5	**0**	8	9

Процедура разделения

4	3	1	2	**7**	6	5	**0**	8	9

Процедура разделения

4	3	1	2	**0**	6	5	**7**	8	9

Процедура разделения

4	3	1	2	0	**6**	**5**	7	8	9

Процедура разделения

4	3	1	2	0	**6**	5	7	8	9

Процедура разделения

4	3	1	2	**0**	6	5	7	8	9

Процедура разделения

4	3	1	2	**0**	6	5	7	8	9

Процедура разделения

0	3	1	2	**4**	6	5	7	8	9

Процедура разделения

- Выбираем разделяющий элемент . Он в конце станет .
- Два указателя: и

1) Увеличиваем I, пока не найдем
2) Уменьшаем j, пока не найдем .
3) Если , меняем местами и , увеличиваем i, уменьшаем j и продолжаем просмотр (пока не станет).

- Меняем местами и

1	2			i		j			N

$$K_i \le K_1 \qquad K_j \ge K_1$$

Процедура разделения и равные элементы

4	4	4	4	4

Процедура разделения и равные элементы

4	**4**	4	4	**4**

Процедура разделения и равные элементы

4	4	4	4	4

Процедура разделения и равные элементы

4	4	**4**	4	4

Процедура разделения и равные элементы

4	4	4	4	4

Процедура разделения и равные элементы

4	4	4	4	4

Если не переставлять элементы, равные разделяющему, то массив одинаковых элементов разделится «плохо»:

4	4	4	4	4

Основная теорема о рекуррентных соотношениях (The master method)

Количество подзадач

Сложность разделения и объединения

Пусть **1, b 2, d 0**

Размер каждой подзадачи

, если

, если

, если

Быстродействие Quicksort

Удачный разделяющий элемент:

В худшем случае:

Выбор разделяющего элемента

Как выбирать?

- Выбирать случайный элемент.
  - Или перед сортировкой можно перемешать весь массив (например тасованием Фишера-Йетса).
- Выбрать медиану малого подмассива (например из первого, последнего и серединного элементов).
- Но в худшем случае все-равно

Алгоритм выбора

- Задача состоит в поиске k-го по величине элемента в массиве.

Алгоритм выбора

- Задача состоит в поиске k-го по величине элемента в массиве.
  - Для **k = 1**, **k = N**, задача решает очевидно за **O(N)**.

Алгоритм выбора

- Задача состоит в поиске k-го по величине элемента в массиве.
  - Для **k = 1**, **k = N**, задача решает очевидно за **O(N)**.
- Можно отсортировать и взять k-ый: в среднем
- Можно ли за O**(N)**?

Линейный алгоритм выбора

- **Разделение:** Разделяем массив аналогично быстрой сортировке.

- **Покорение:** Если **k < q**, то продолжаем выбор **k**-го в левой половине. Если **k > q**, то продолжаем выбор **(k-q)**-го в правой половине. Если **k = q**, то заканчиваем алгоритм

- **Комбинирование:** Не требуется.

Основная теорема о рекуррентных соотношениях (The master method)

Количество подзадач

Сложность разделения и объединения

Пусть **1, b 2, d 0**

Размер каждой подзадачи

, если

, если

, если

Сложность линейного алгоритма выбора

- В среднем , в худшем случае .
- Все проблемы аналогичны проблемам быстрой сортировки.
- Существует алгоритм BFPRT работающий за в худшем случае.

Проблема стабильности

- Неотсортированный массив по возрасту:

[Вася – 17 лет, Коля – 18 лет, Петя – 16 лет, Юля – 17 лет]

Проблема стабильности

- Неотсортированный массив по возрасту:

[Вася – 17 лет, Коля – 18 лет, Петя – 16 лет, Юля – 17 лет]

- Стабильная сортировка по возрасту:

[Петя – 16 лет, Вася – 17 лет, Юля – 17 лет, Коля – 18 лет]

- Нестабильная сортировка по возрасту:

[Петя – 16 лет, Юля – 17 лет, Вася – 17 лет, Коля – 18 лет]

Стабильность сортировок

- Стабильная сортировка не меняет относительный порядок элементов с равными ключами.

Стабильность сортировок

- Стабильная сортировка не меняет относительный порядок элементов с равными ключами.
- Стабильные: Выбором, пузырьковая, вставками.
- Нестабильные: bozosort, bogosort, Шелла, быстрая.

Сортировка слиянием (Mergesort)

- **Разделение:** Каким-либо образом делим массив на 2 равные части.

- **Покорение:** Для каждой из частей размером больше 1 алгоритм вызывается рекурсивно.

- **Комбинирование:** Линейное слияние отсортированных подмассивов.

Сортировка слиянием (Mergesort)

5	7	9	8	2	7	3	1	4

Сортировка слиянием (Mergesort)

5	7	9	8	2	7	3	1	4

5	7	9	8	2

7	3	1	4

Сортировка слиянием (Mergesort)

5	7	9	8	2	7	3	1	4

5	7	9	8	2

7	3	1	4

2	5	7	8	9

1	3	4	7

Сортировка слиянием (Mergesort)

2	5	7	8	9	-

1	3	4	7	-

Сортировка слиянием (Mergesort)

2	5	7	8	9	-

1	3	4	7	-

Сортировка слиянием (Mergesort)

2	5	7	8	9	-

1	3	4	7	-

1								

Сортировка слиянием (Mergesort)

2	5	7	8	9	-

1	3	4	7	-

1	2							

Сортировка слиянием (Mergesort)

2	5	7	8	9	-

1	3	4	7	-

1	2	3						

Сортировка слиянием (Mergesort)

2	5	7	8	9	-

1	3	4	7	-

1	2	3	4					

Сортировка слиянием (Mergesort)

2	5	7	8	9	-

1	3	4	7	-

1	2	3	4	5				

Сортировка слиянием (Mergesort)

2	5	7	8	9	-

1	3	4	7	-

1	2	3	4	5	7			

Сортировка слиянием (Mergesort)

2	5	7	8	9	-

1	3	4	7	-

1	2	3	4	5	7	7		

Сортировка слиянием (Mergesort)

2	5	7	8	9	-

1	3	4	7	-

1	2	3	4	5	7	7	8	9

Сортировка слиянием (Mergesort)

Сортировка слиянием (Mergesort)

Линейный алгоритм слияния 2 отсортированных массивов:

- Заводим 2 индекса, изначально указывающих на начала массивов.
- На каждом шаге выбираем наименьший элемент из двух, на которые указывают индексы, записываем его в результирующий массив и сдвигаем этот индекс. При равенстве элементов, берем из первого.
- Когда кончился один из массивов, переписываем все оставшиеся элементы в результирующий массив. Результирующий массив отсортирован.

Анализ сортировки слиянием

- Сложность в худшем случае
- Требуется дополнительная память (еще один массив размером **N**).
- Осуществляет доступ к сортируемым элементам последовательно. Поэтому годится для сортировки связанных списков.
- Является стабильной.

Пирамидальная сортировка (Heapsort)

Пирамида – массив, в котором для каждого элемента **n** выполнено:

- (если).
- (если).

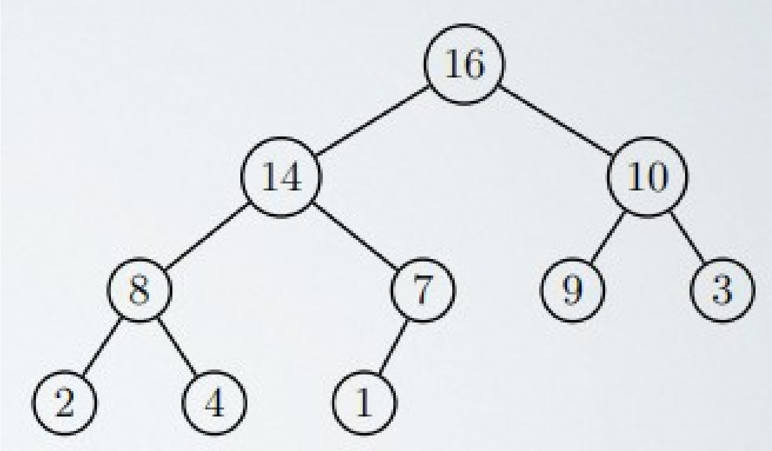
Пример: 16, 14, 10, 8, 7, 9, 3, 2, 4, 1.

Пирамидальная сортировка (Heapsort)

Пирамида – массив, в котором для каждого элемента **n** выполнено:

- (если).
- (если).

Пример: 16, 14, 10, 8, 7, 9, 3, 2, 4, 1.



Алгоритм сортировки:

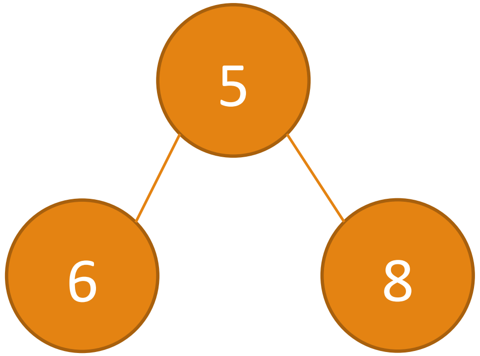
- Из массива делаем пирамиду (перестановками).
- Из пирамиды делаем отсортированный массив (перестановками).

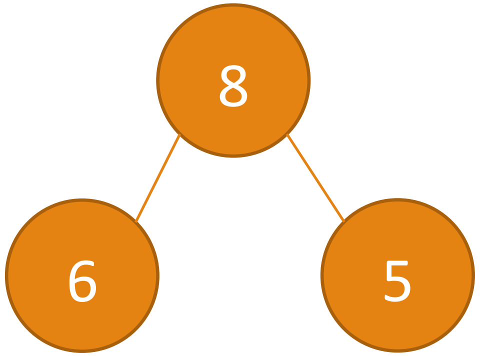
Утопление элемента (SINK)

Если элемент не удовлетворяет условиям пирамиды:

- Находим
- Меняем местами и
- При необходимости повторяем процедуру для нового положения .

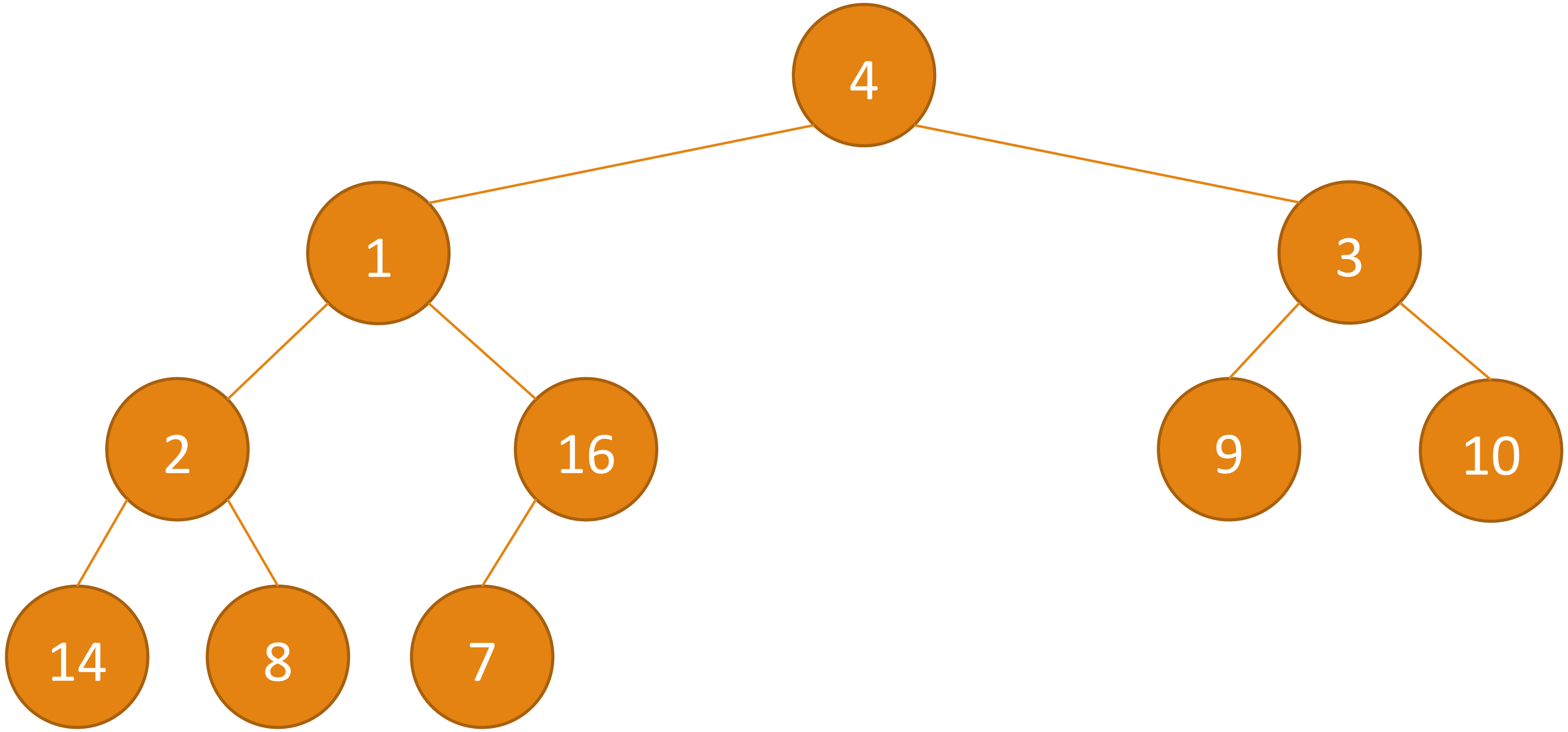
Если для всех узлов , пирамида корректна, то для тоже.





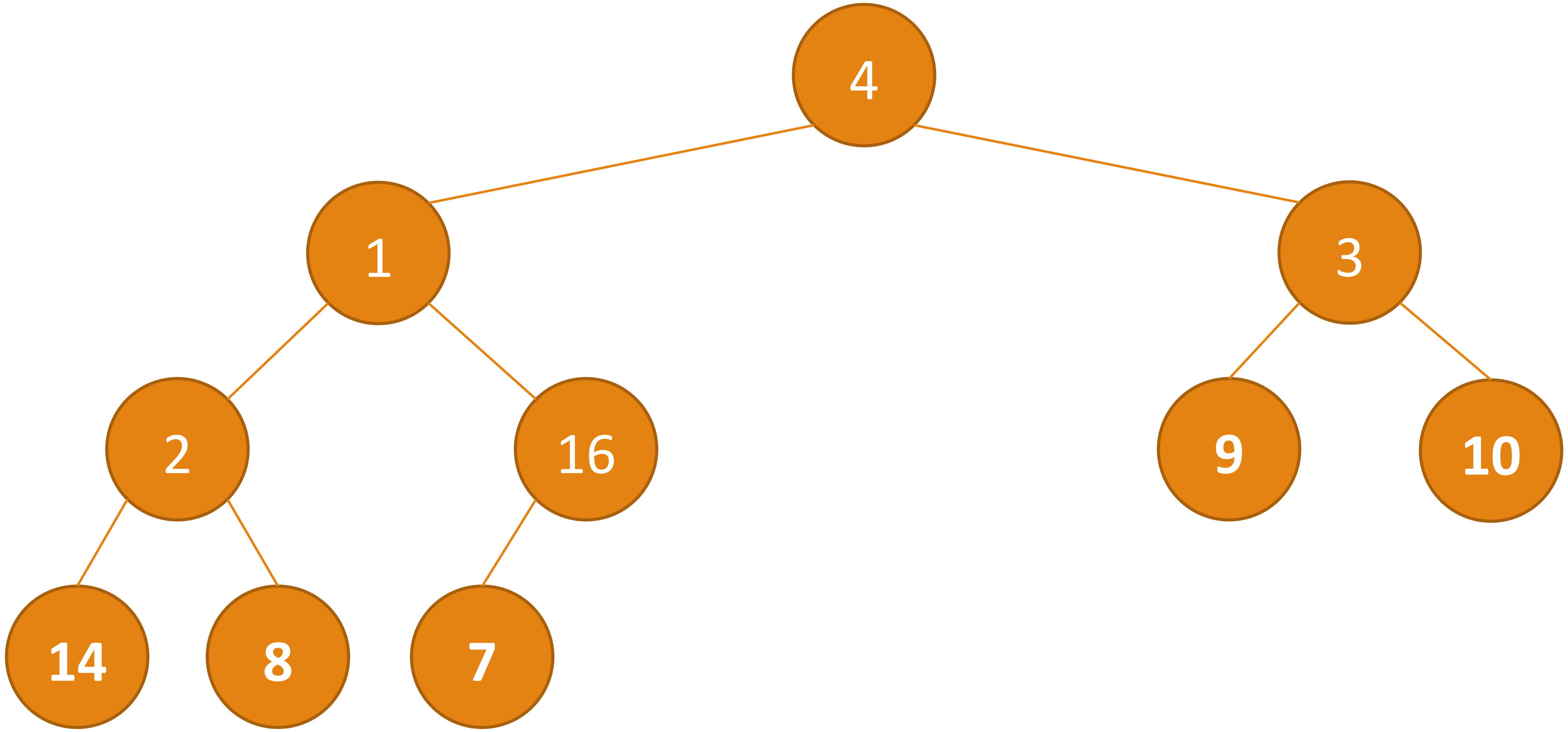
Построение пирамиды

4	1	3	2	16	9	10	14	8	7



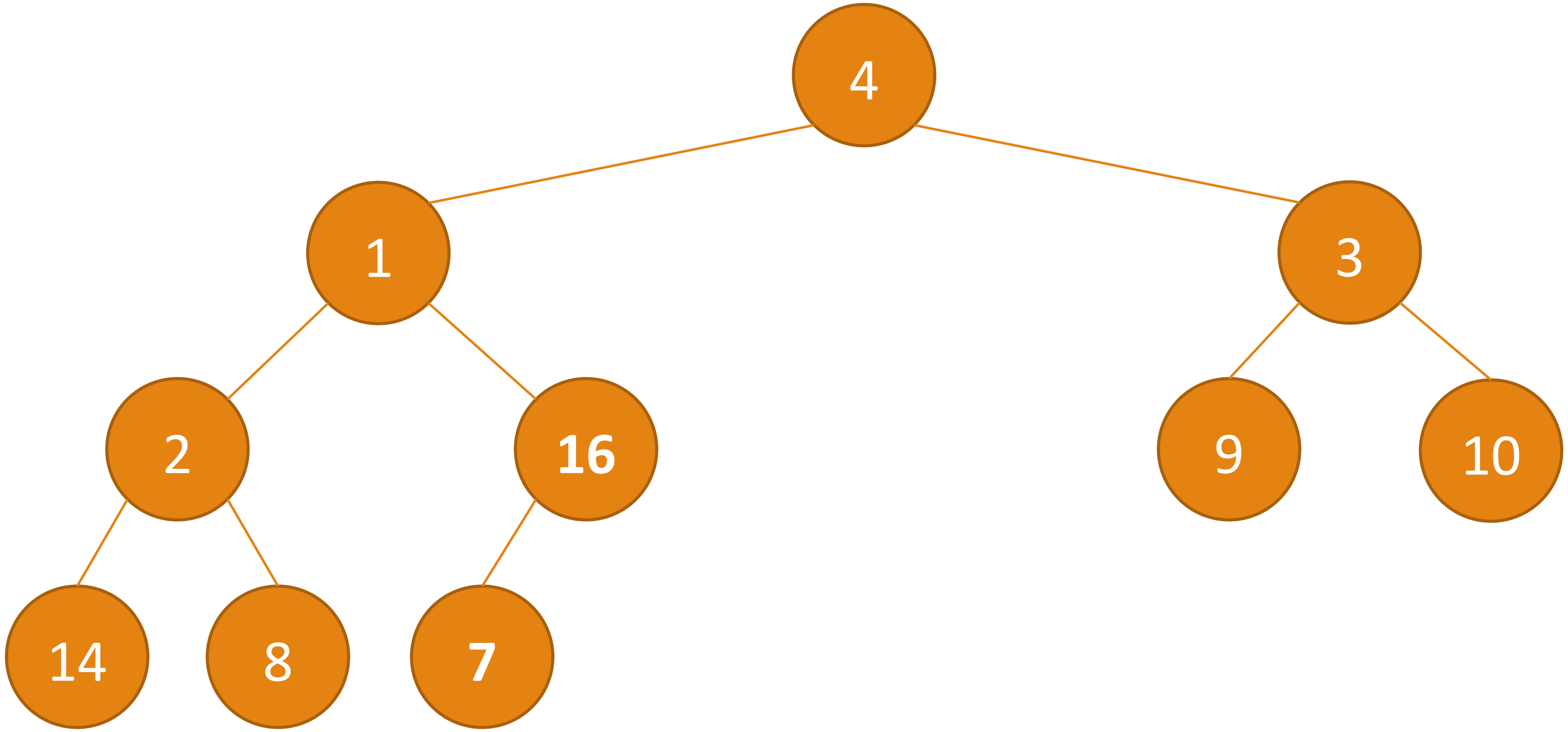
Построение пирамиды

4	1	3	2	16	9	10	14	8	7



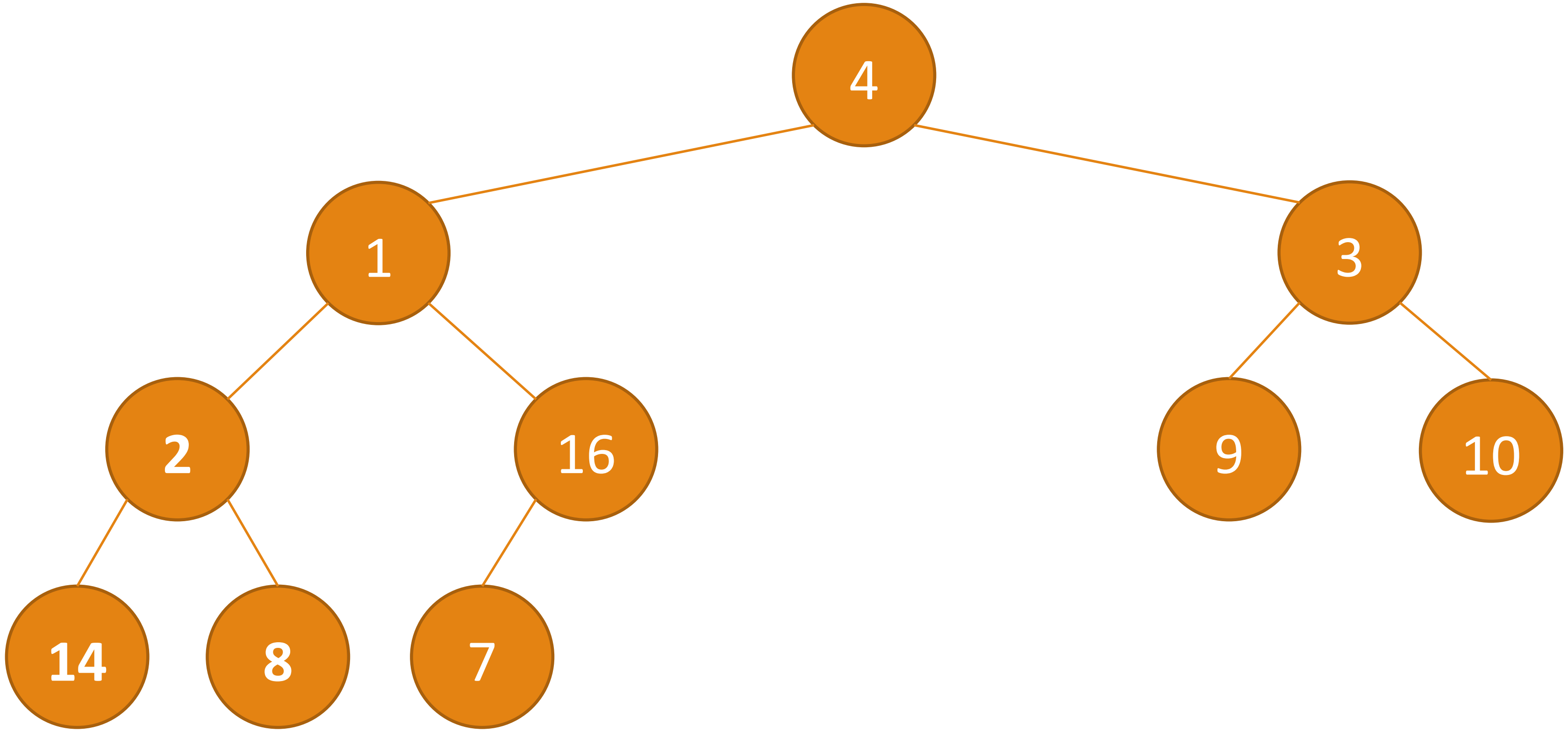
Построение пирамиды

4	1	3	2	**16**	9	10	14	8	**7**



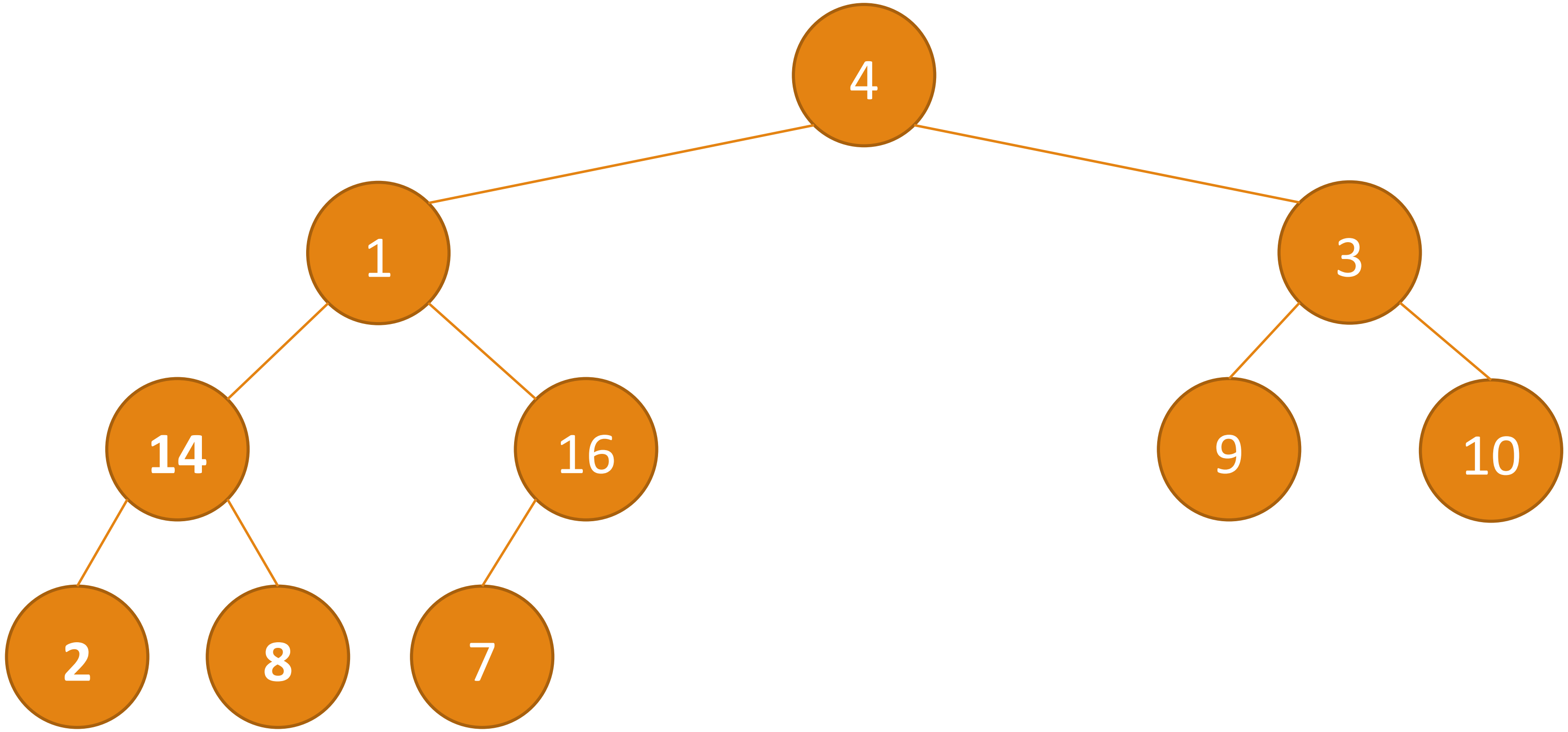
Построение пирамиды

4	1	3	**2**	16	9	10	**14**	**8**	7



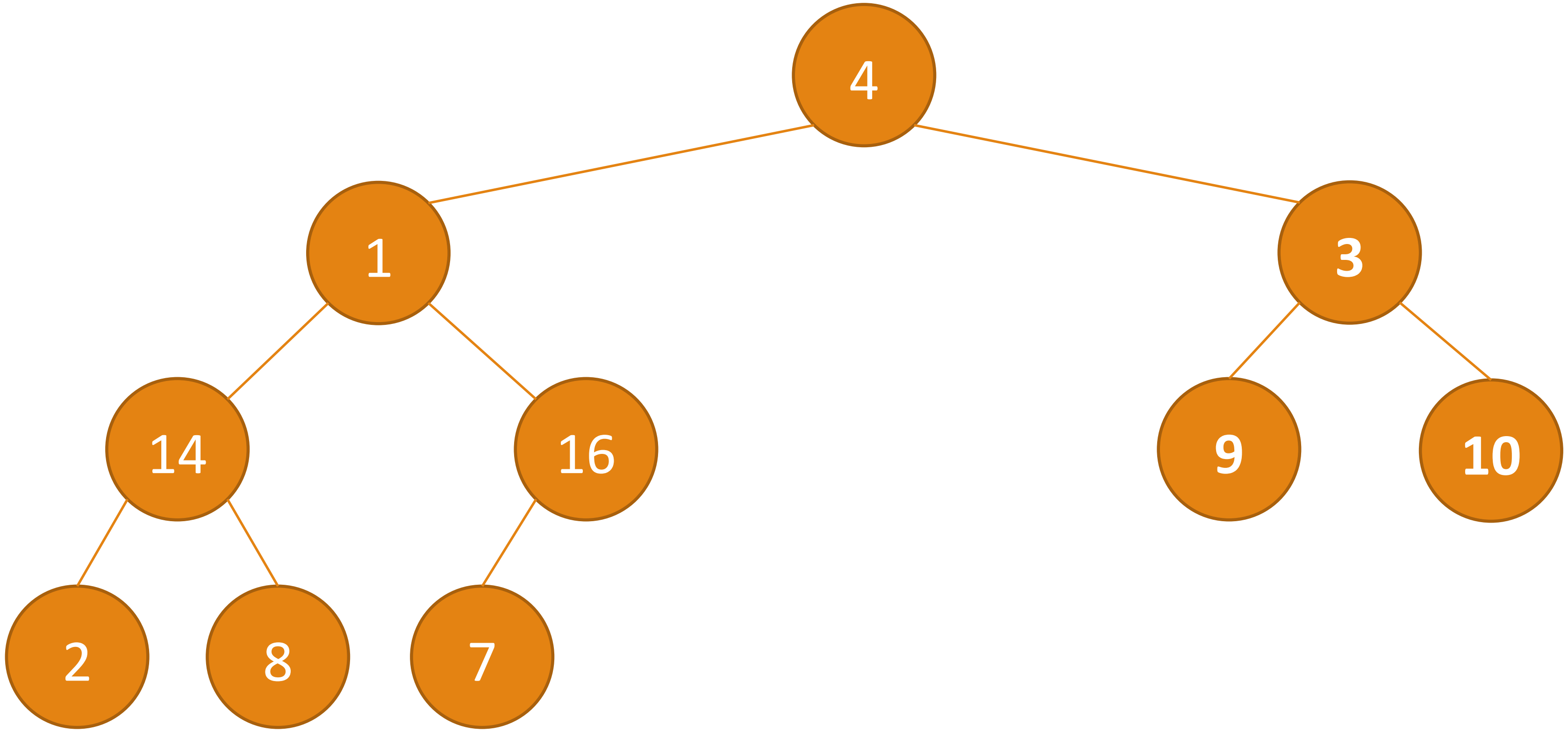
Построение пирамиды

4	1	3	**14**	16	9	10	**2**	**8**	7



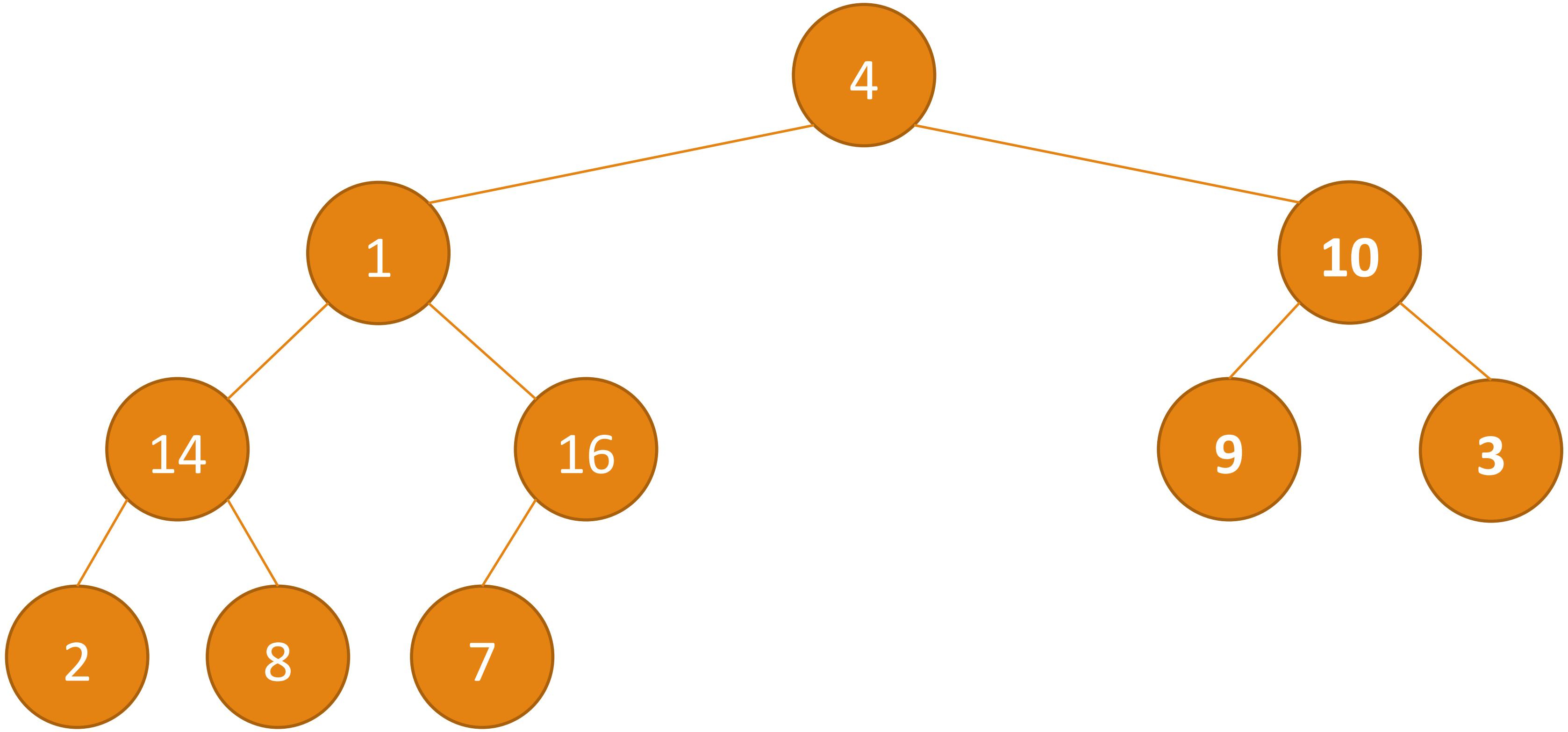
Построение пирамиды

4	1	**3**	14	16	**9**	**10**	2	8	7



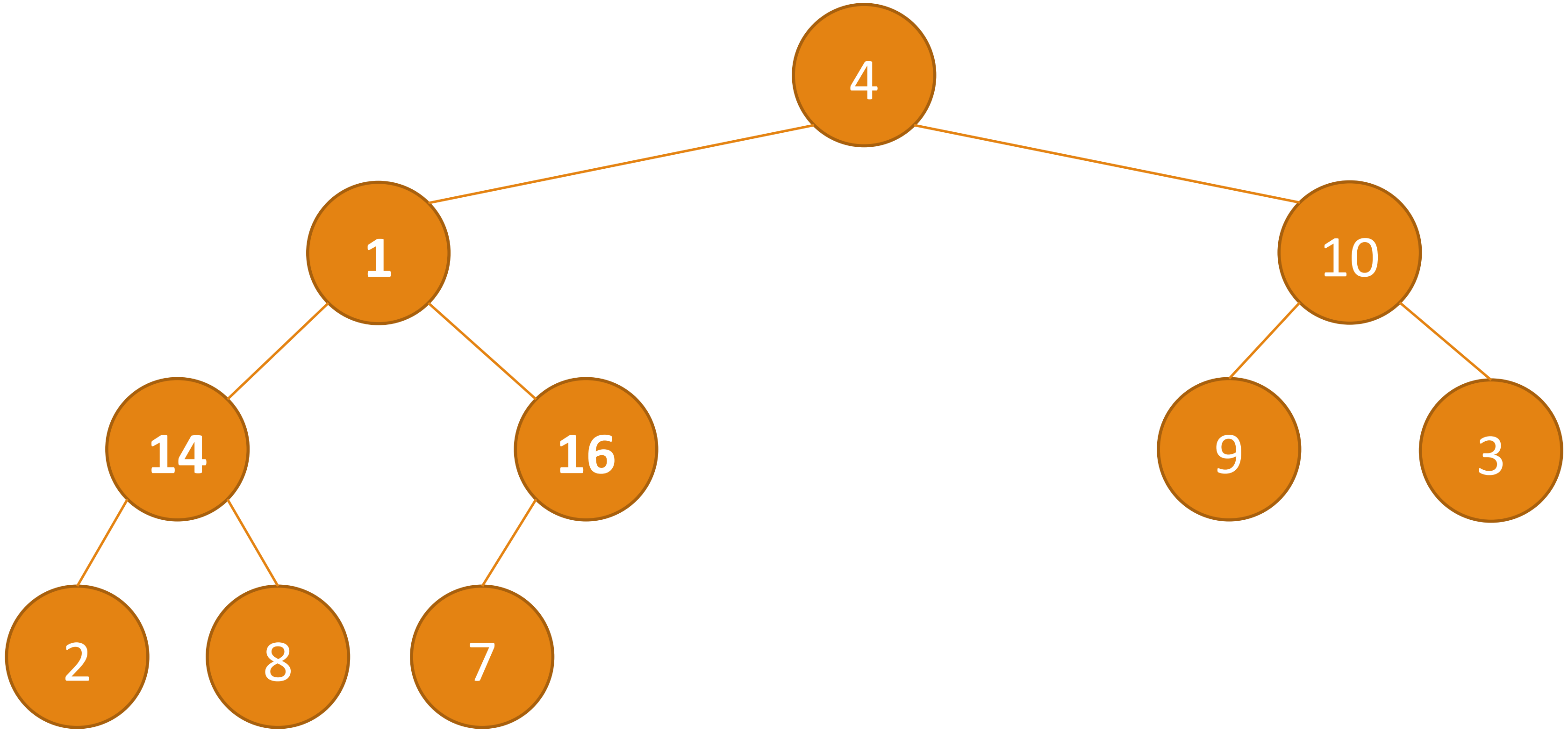
Построение пирамиды

4	1	**10**	14	16	**9**	**3**	2	8	7



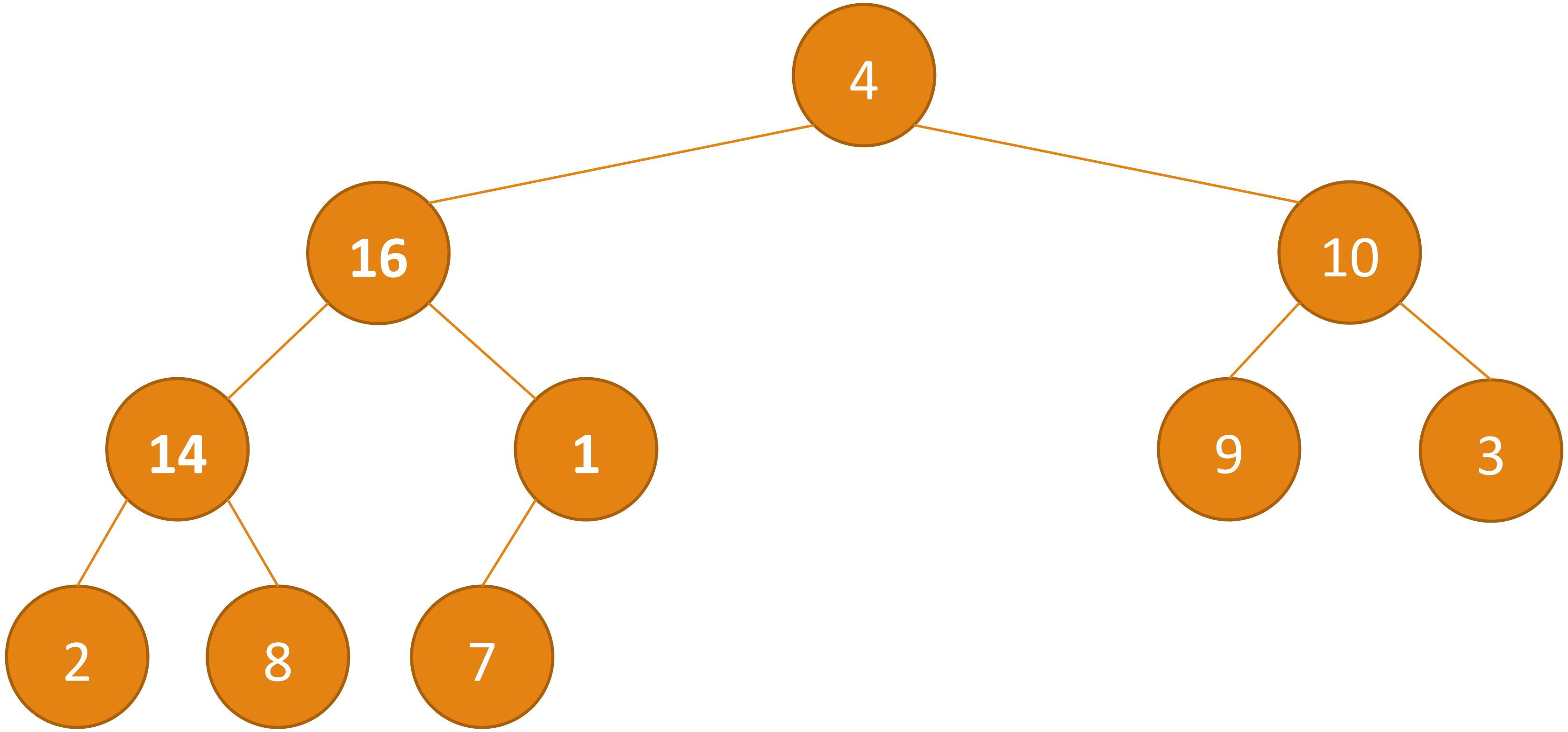
Построение пирамиды

4	**1**	10	**14**	**16**	9	3	2	8	7



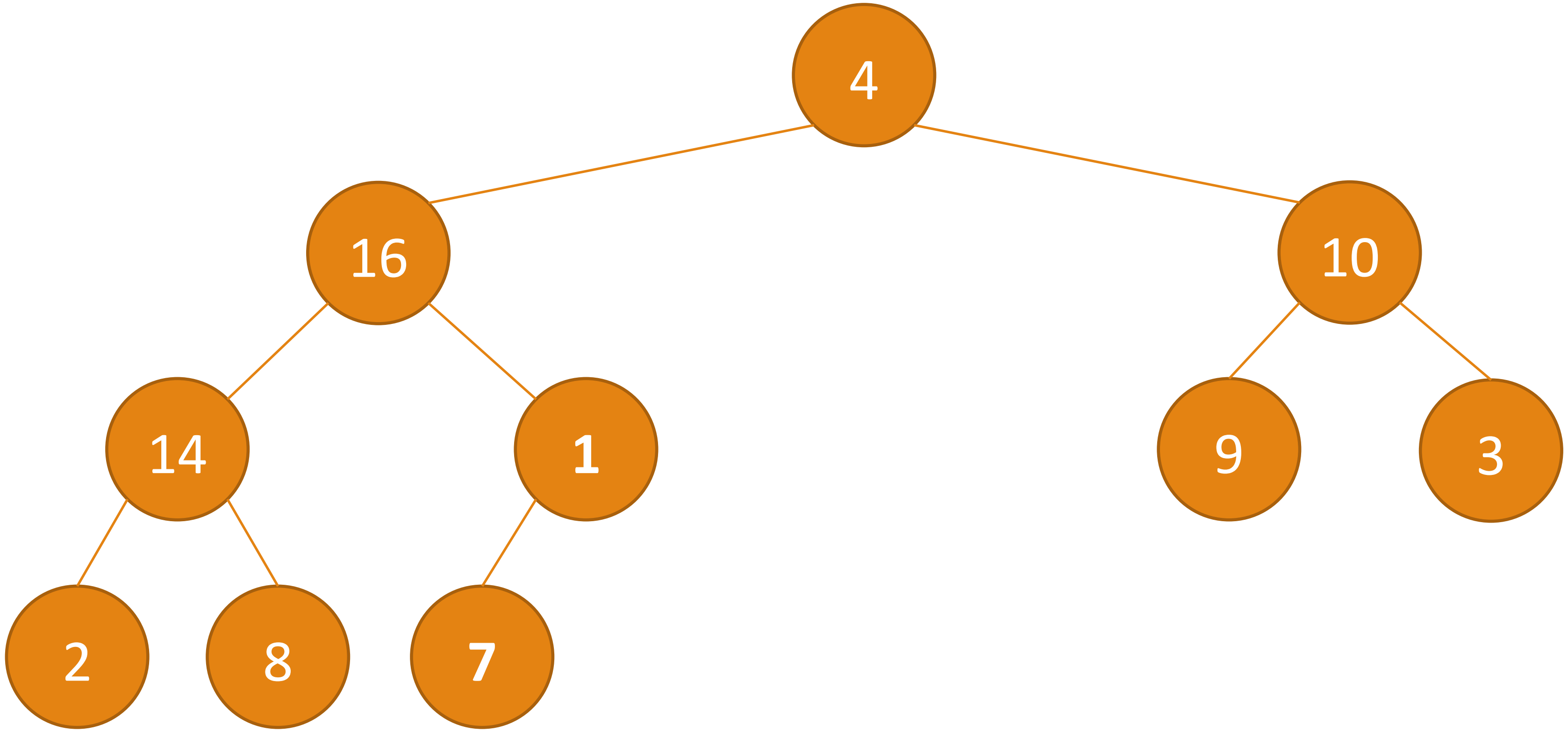
Построение пирамиды

4	**16**	10	**14**	**1**	9	3	2	8	7



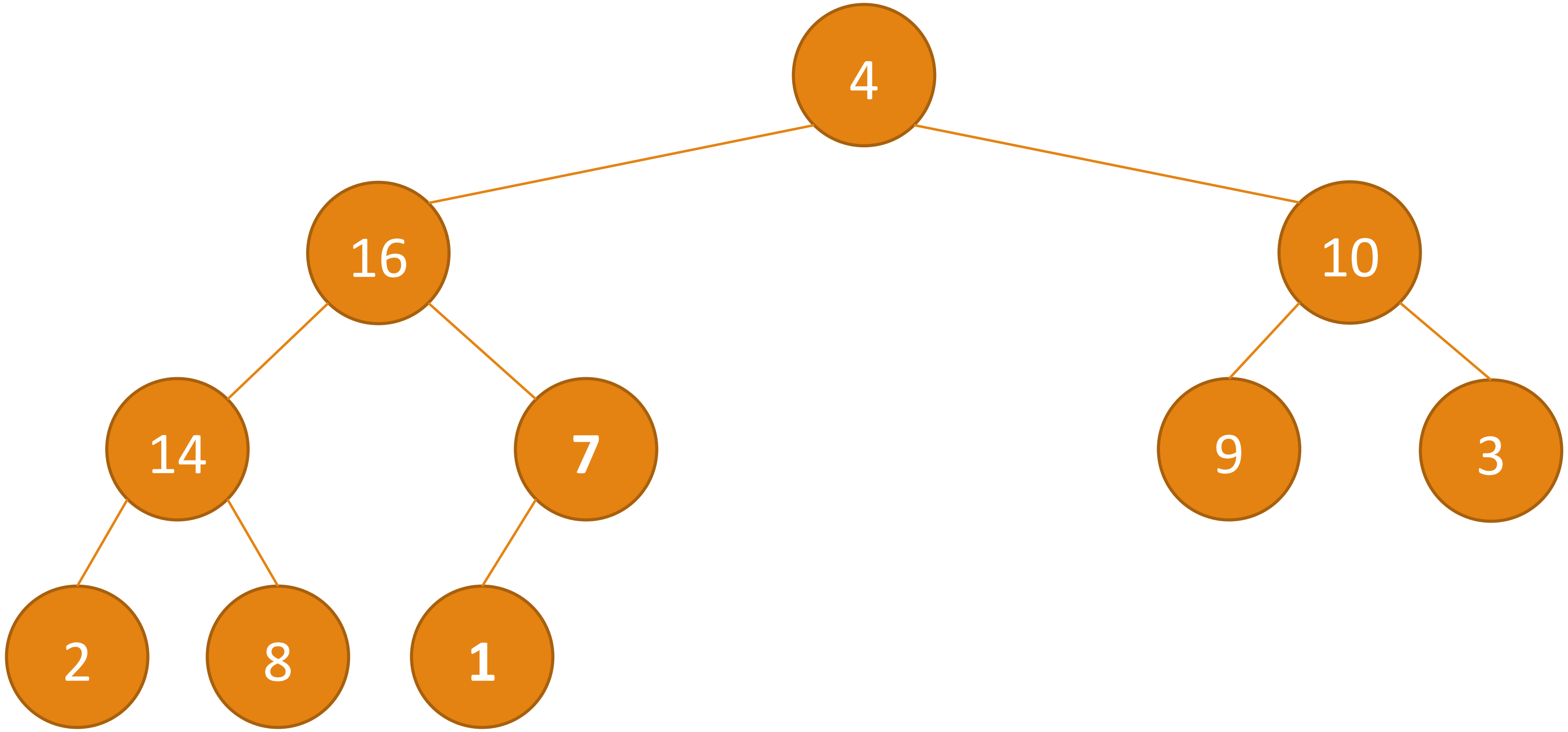
Построение пирамиды

4	16	10	14	**1**	9	3	2	8	**7**



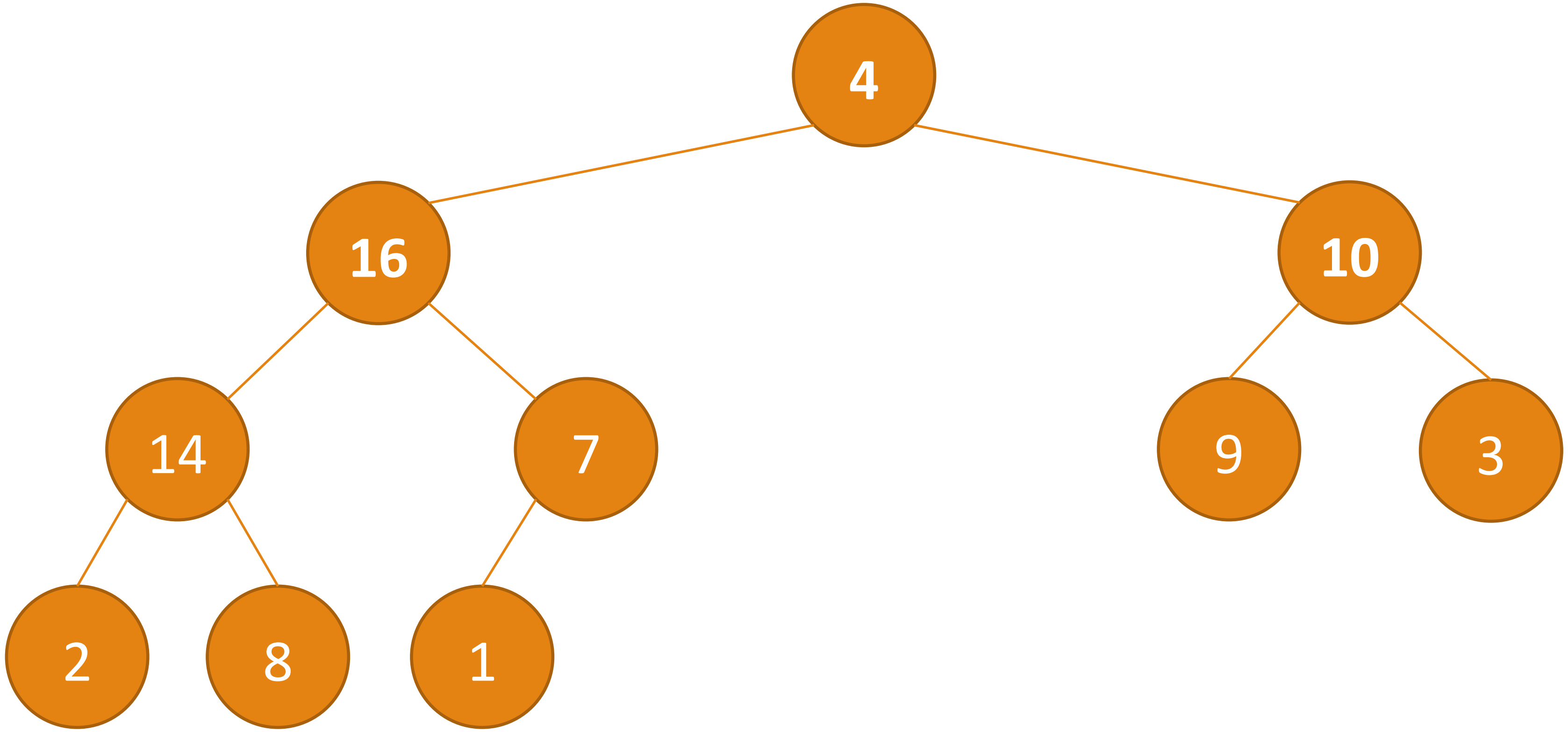
Построение пирамиды

4	16	10	14	**7**	9	3	2	8	**1**



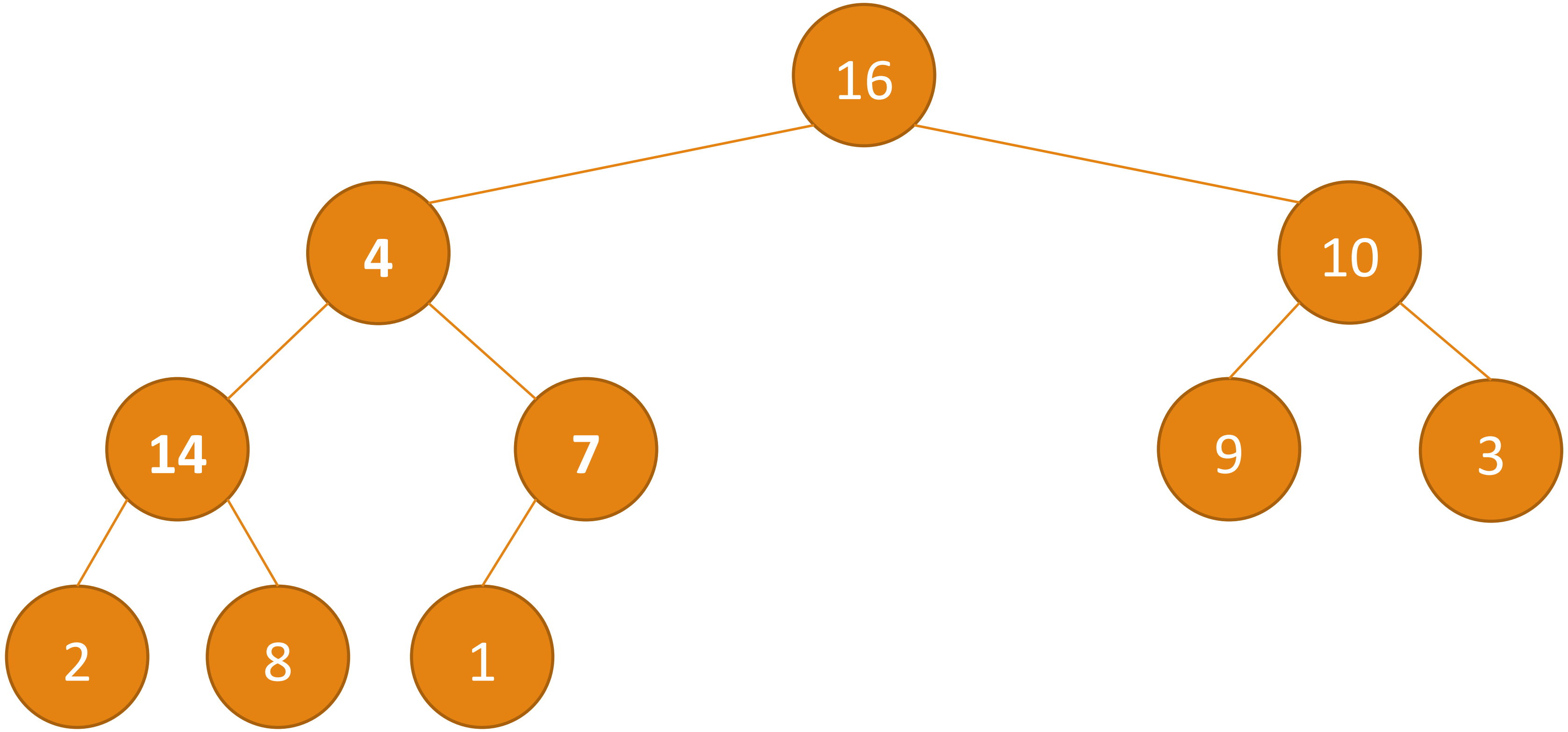
Построение пирамиды

4	**16**	**10**	14	7	9	3	2	8	1



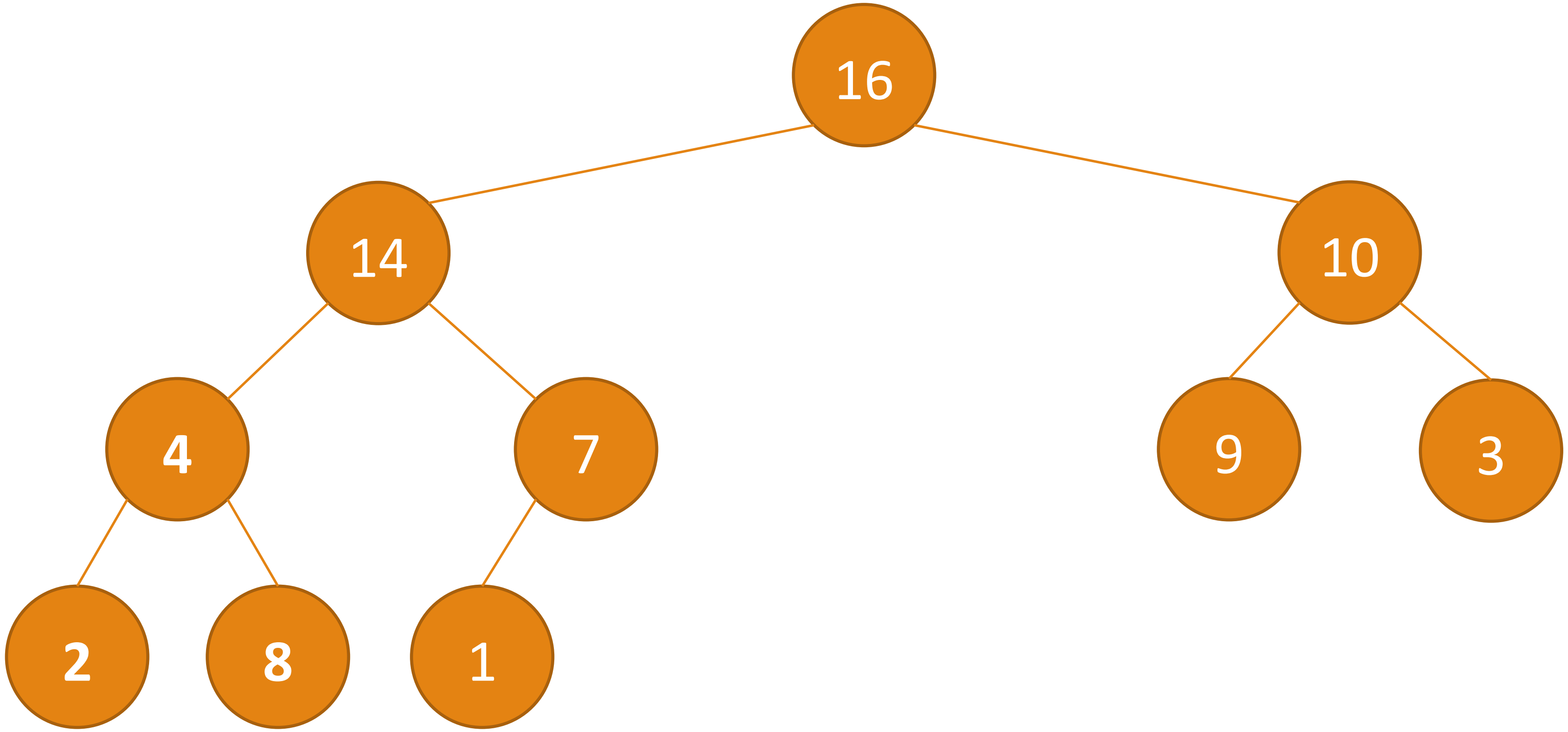
Построение пирамиды

16	**4**	10	**14**	**7**	9	3	2	8	1



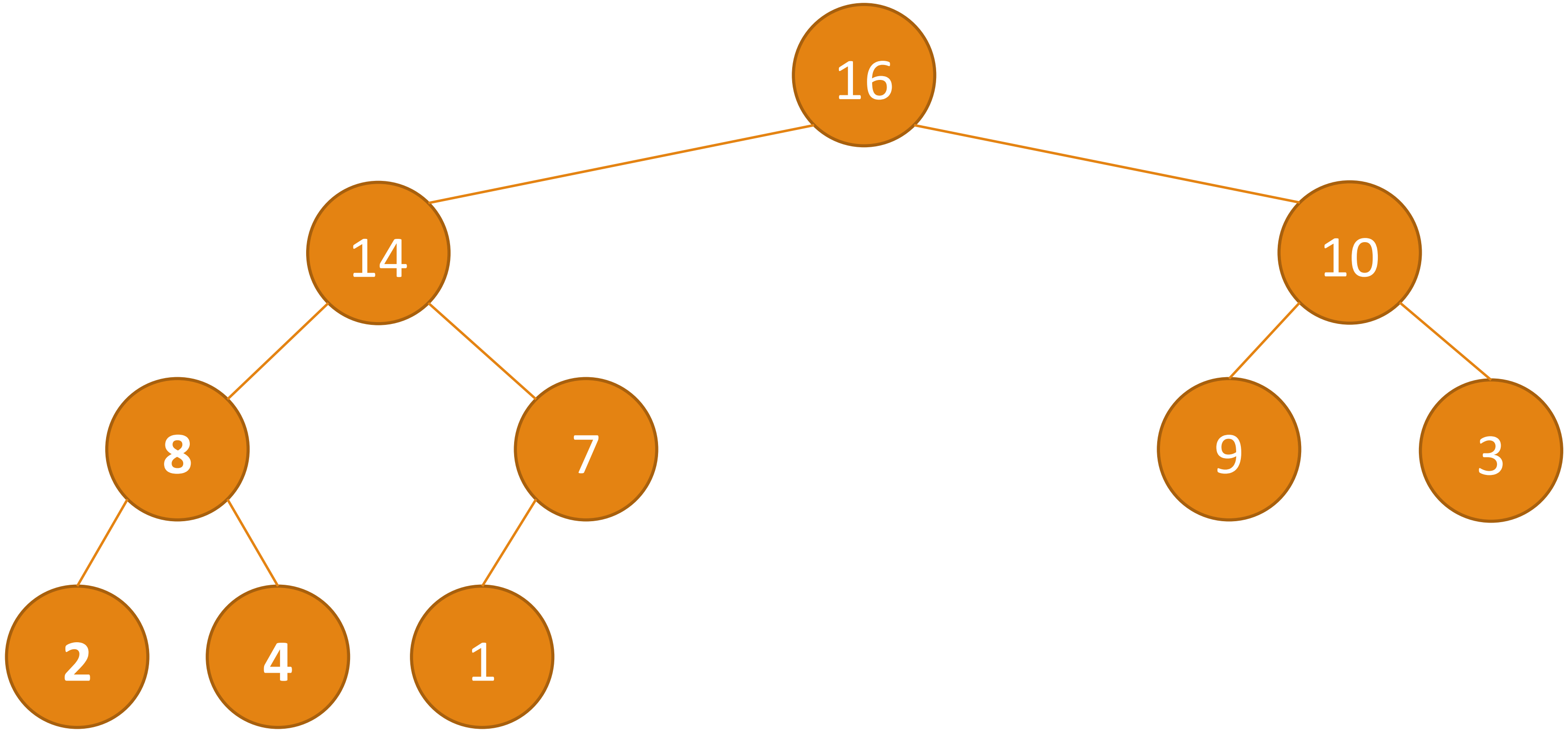
Построение пирамиды

16	14	10	**4**	7	9	3	**2**	**8**	1



Построение пирамиды

16	14	10	8	7	9	3	2	4	1

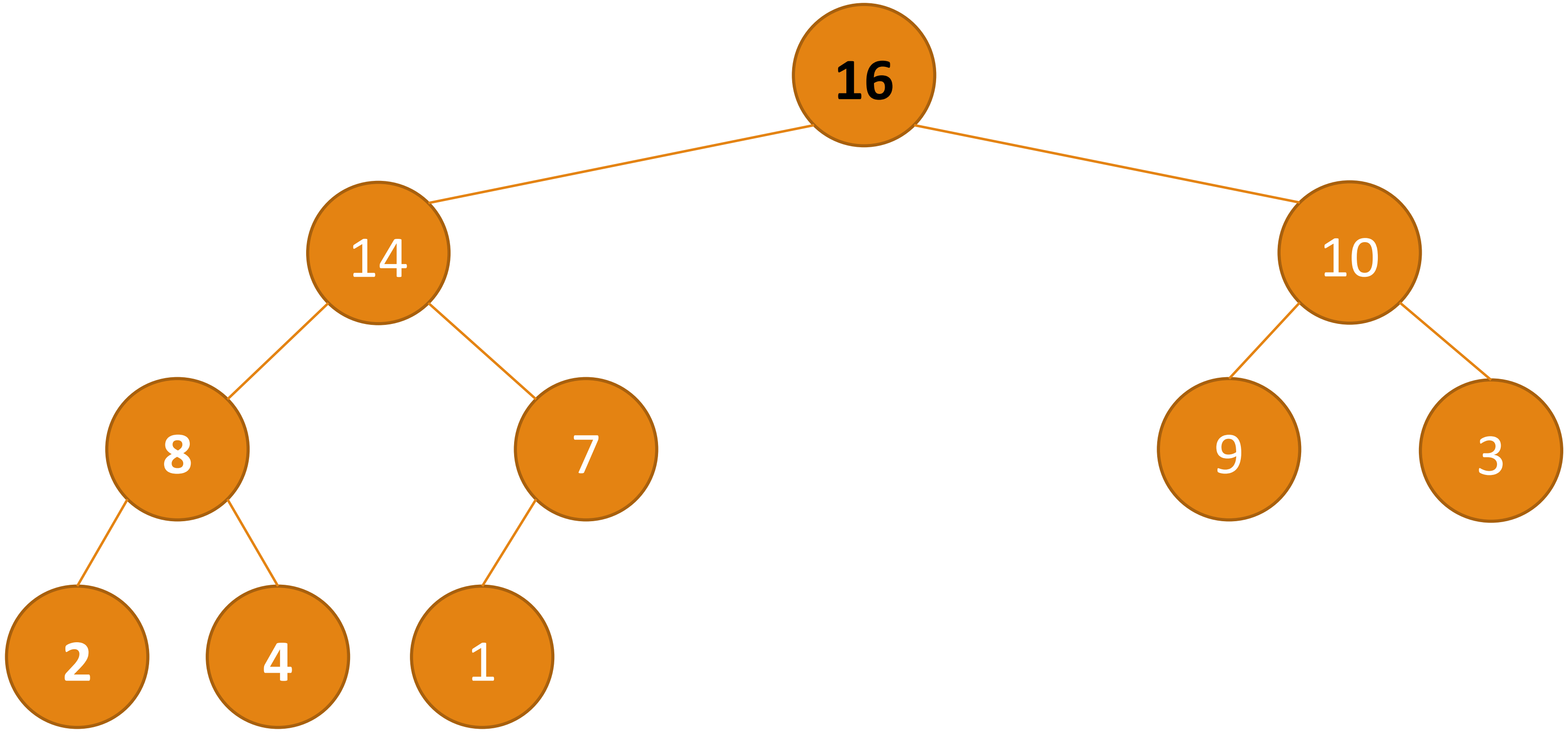


Построение пирамиды

- Идем с конца массива и топим элементы.
- Вторая половина массива изначально удовлетворяет условию пирамиды.

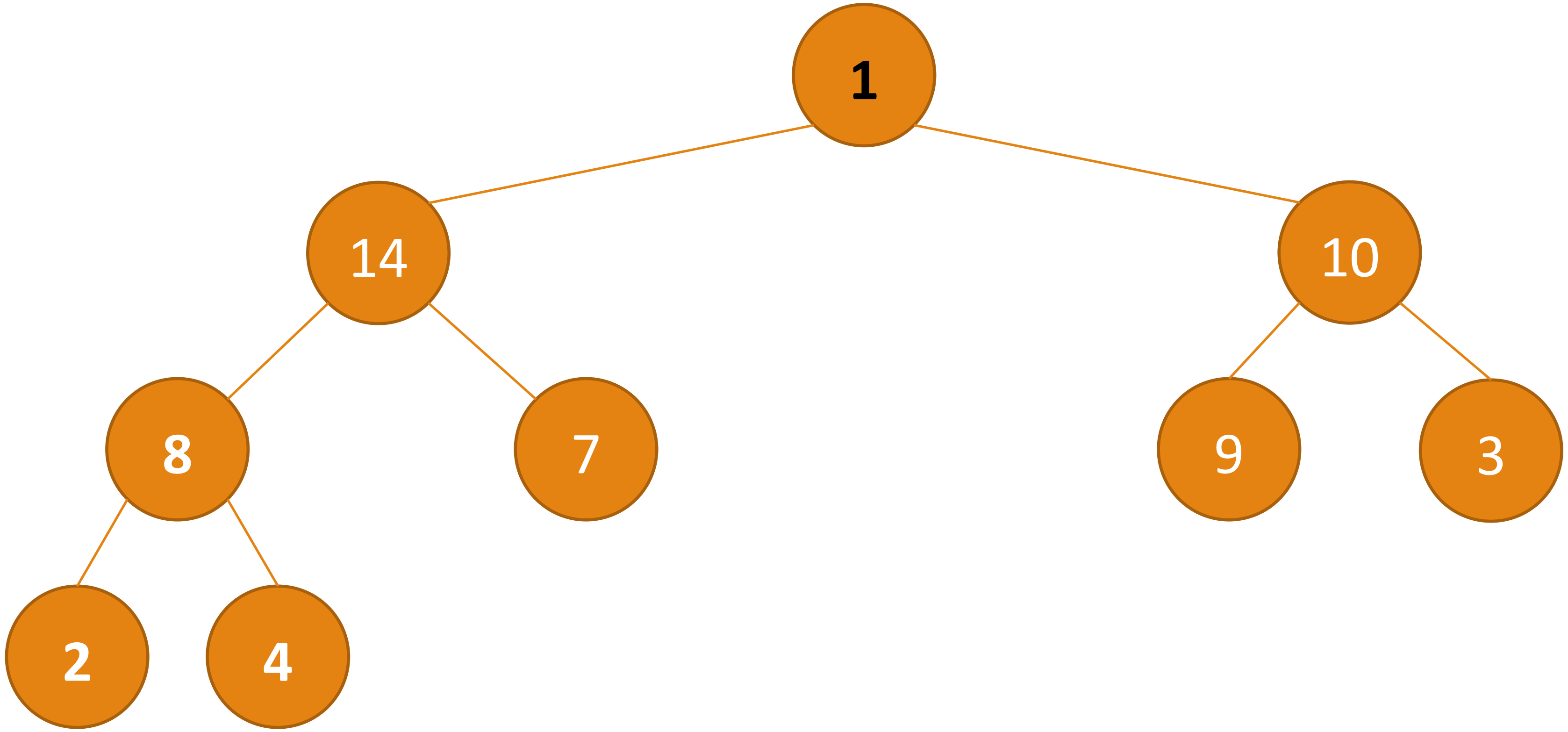
Получение массива

16	14	10	8	7	9	3	2	4	1



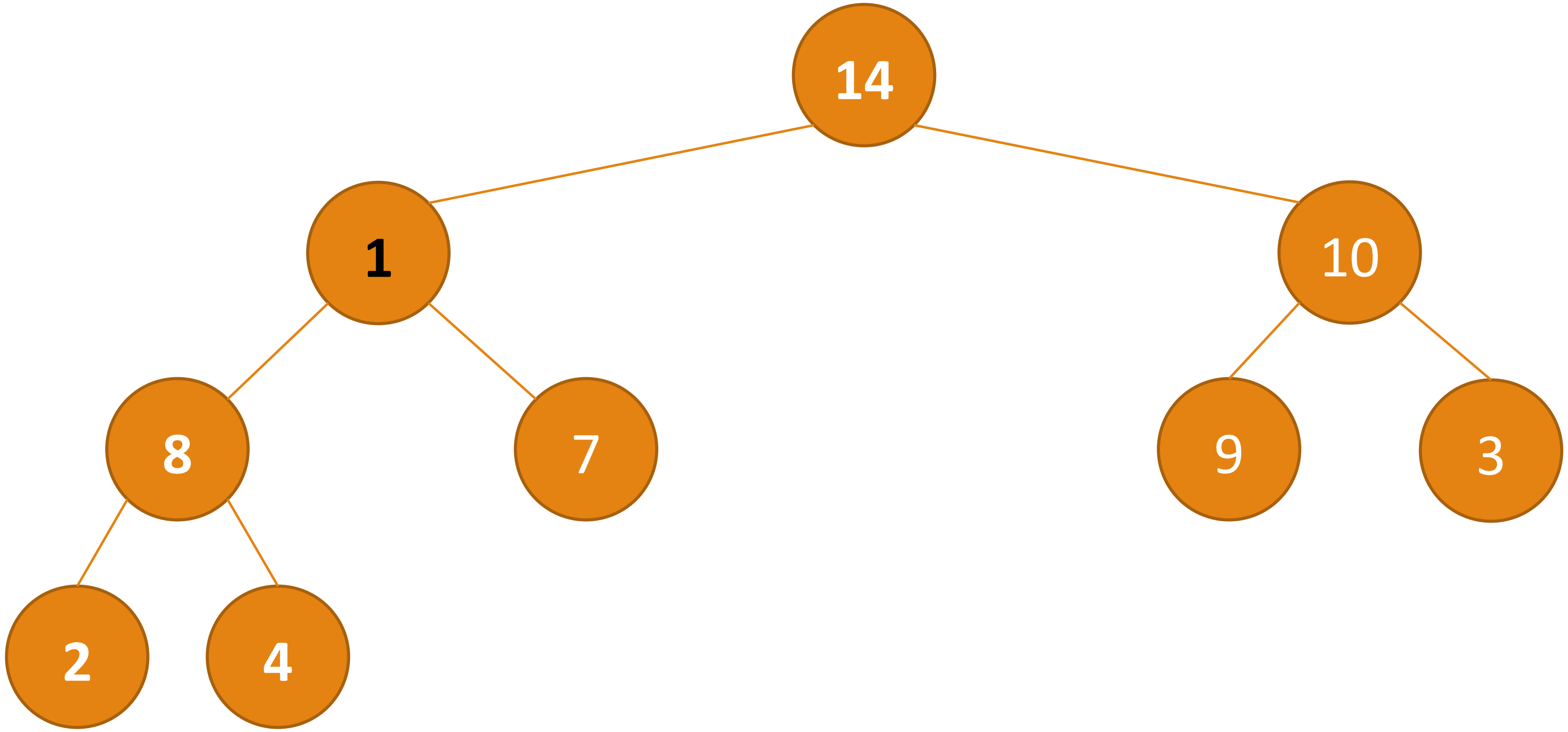
Получение массива

1	14	10	8	7	9	3	2	4	16



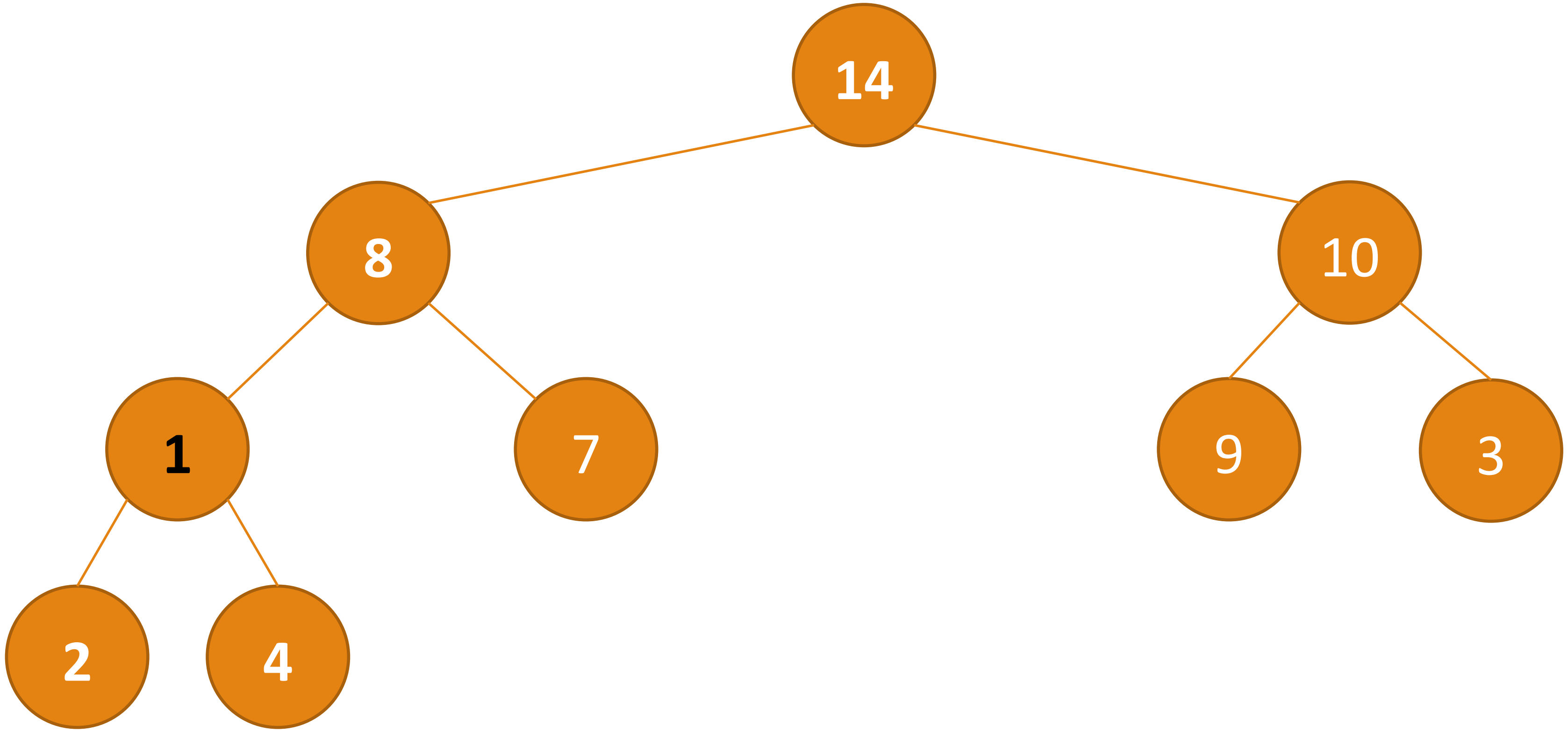
Получение массива

14	1	10	8	7	9	3	2	4	16



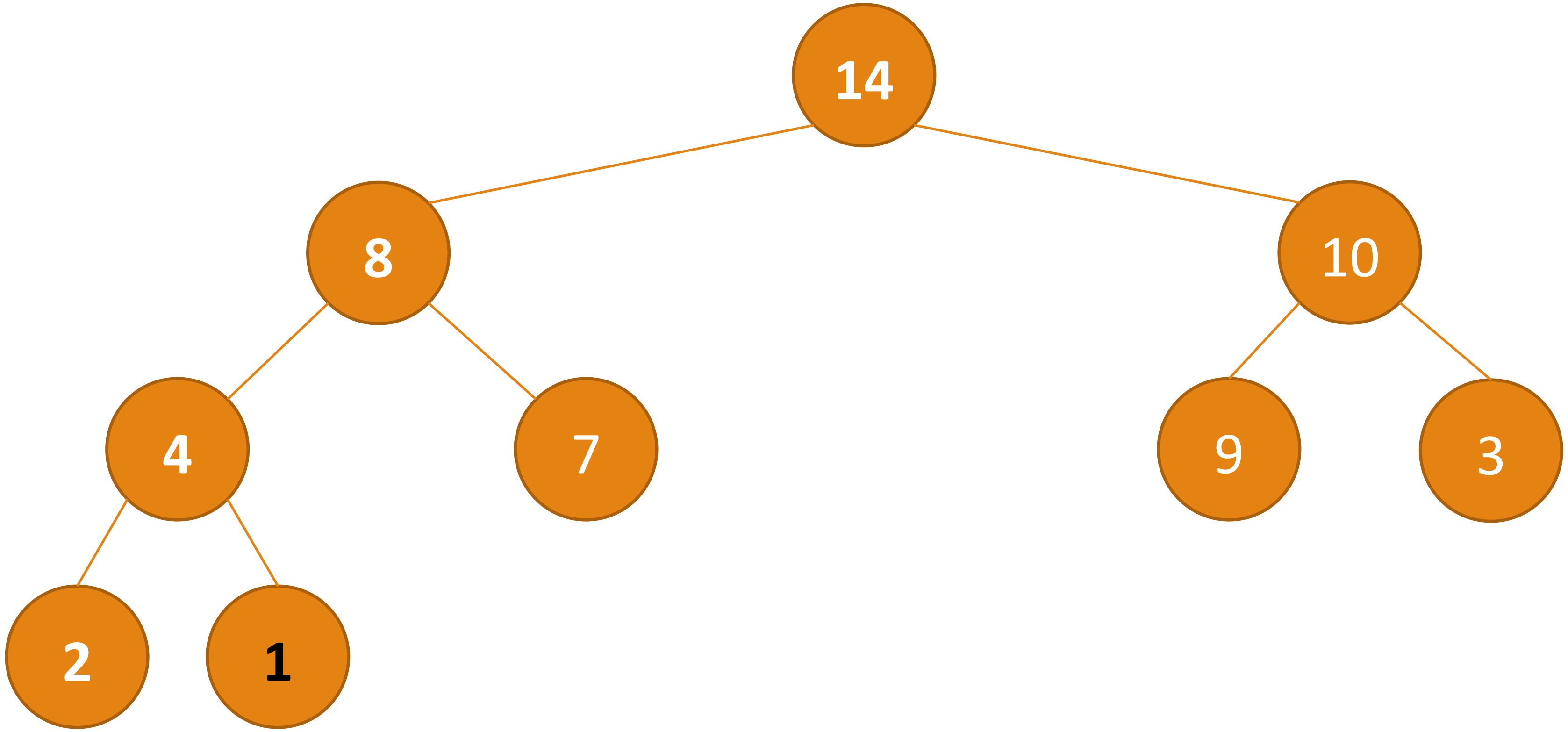
Получение массива

14	8	10	1	7	9	3	2	4	16



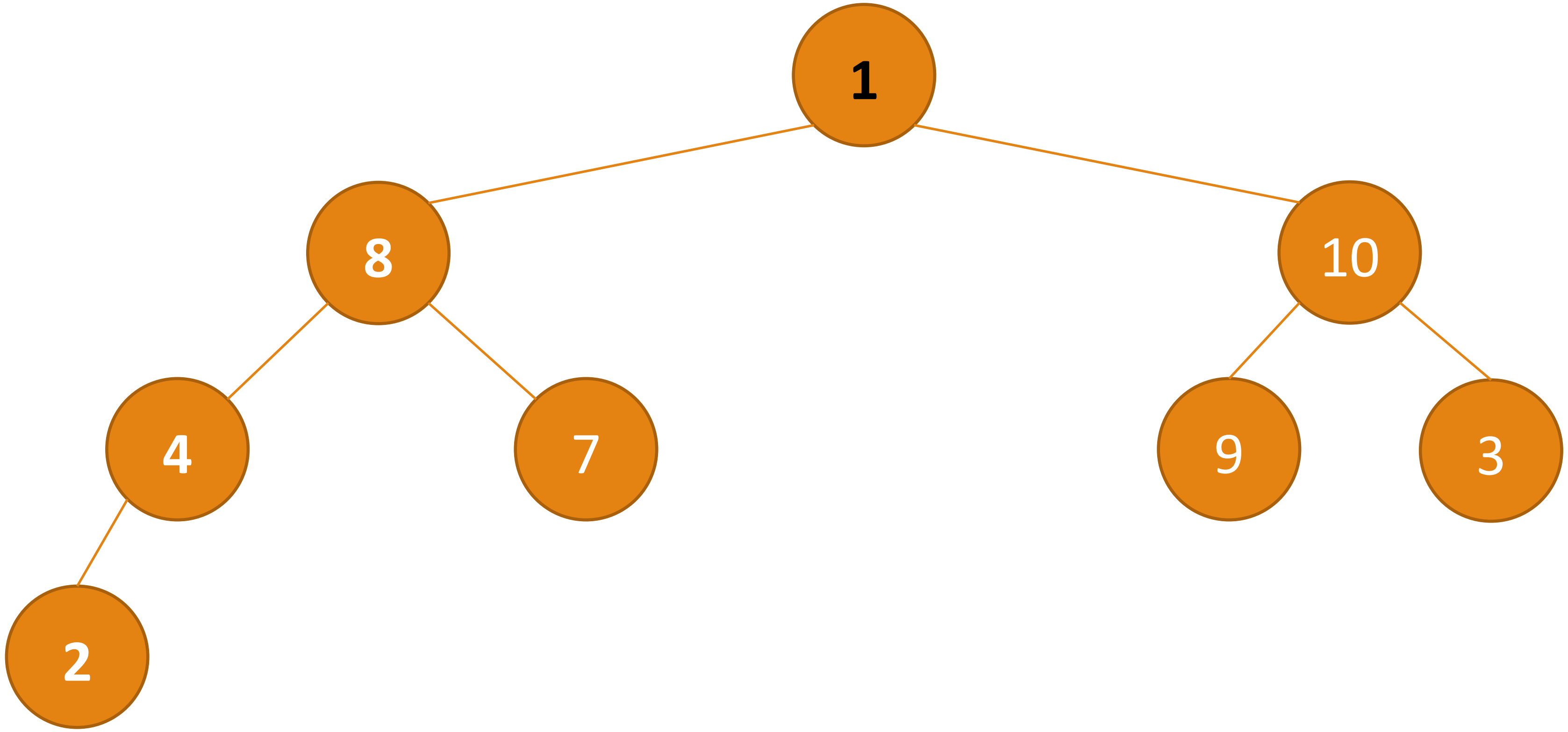
Получение массива

14	8	10	4	7	9	3	2	1	16



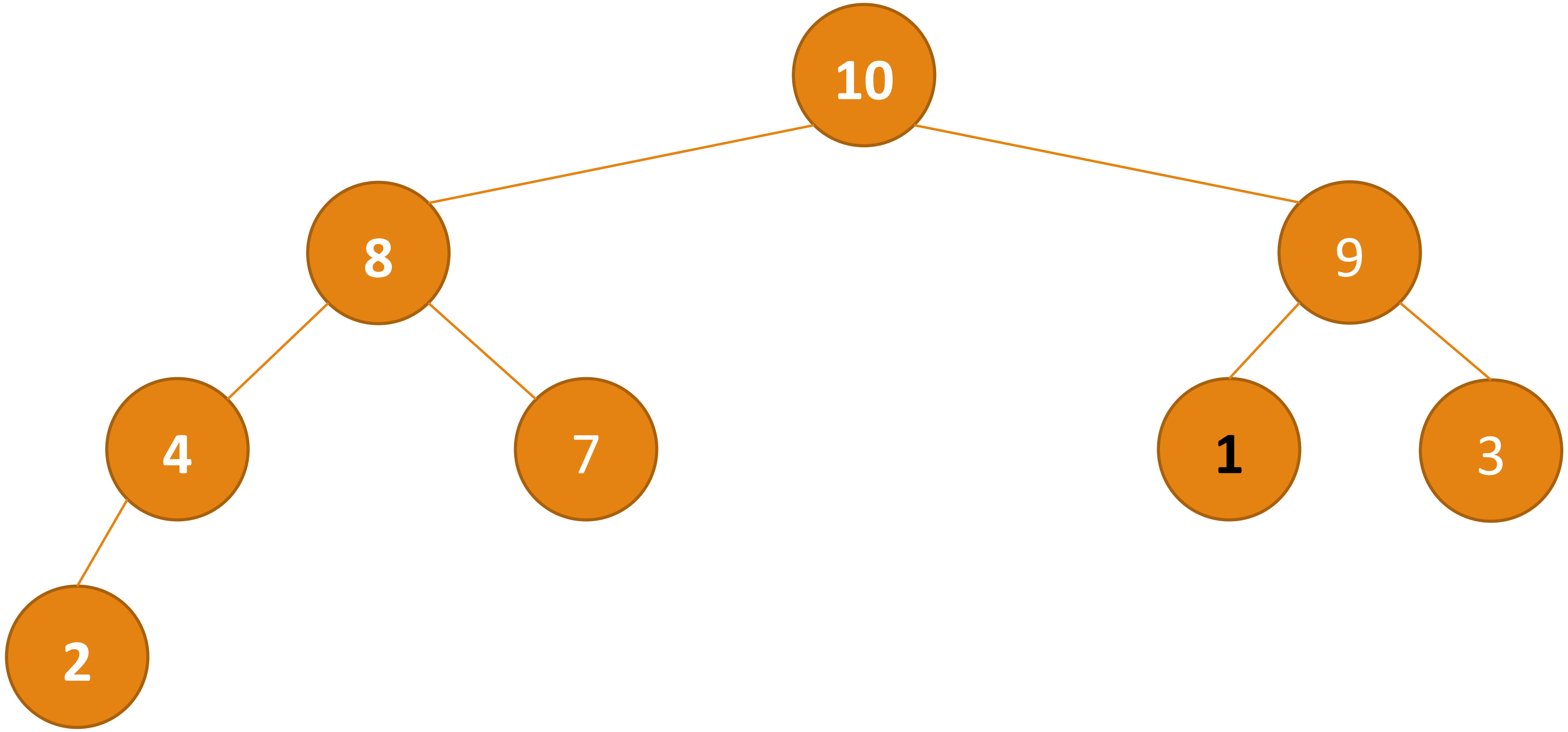
Получение массива

1	8	10	4	7	9	3	2	14	16



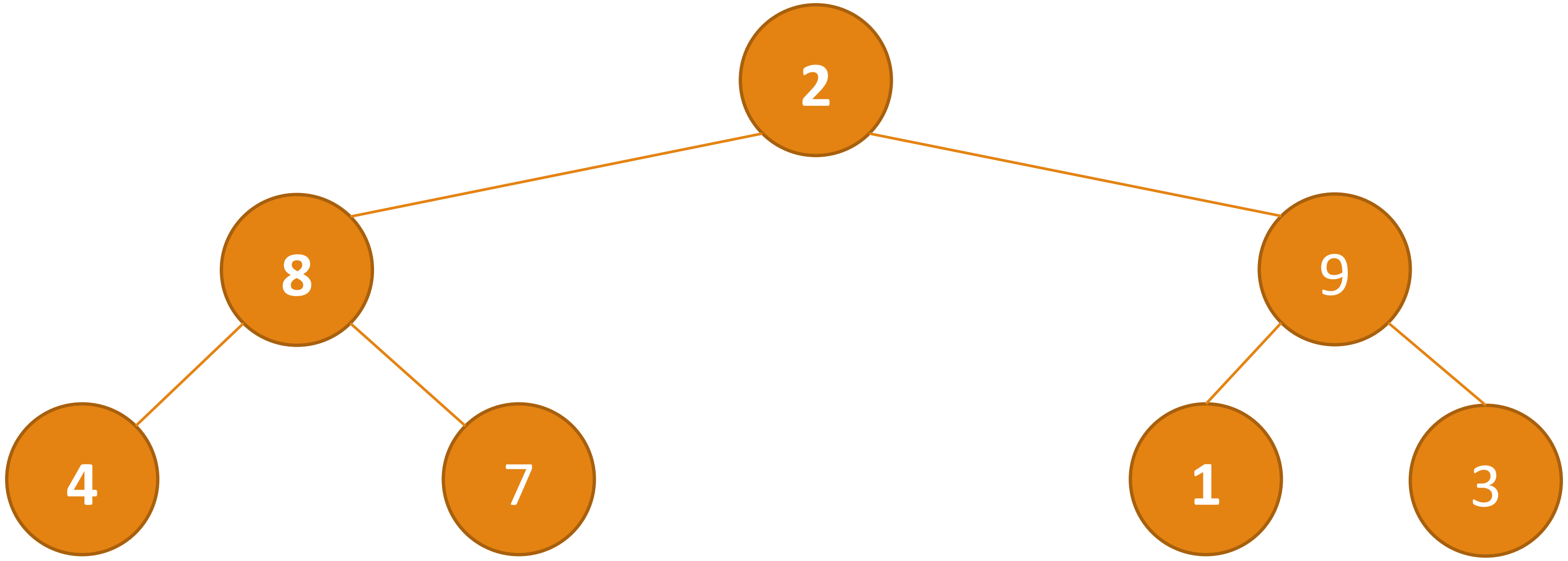
Получение массива

10	8	9	4	7	1	3	2	14	16



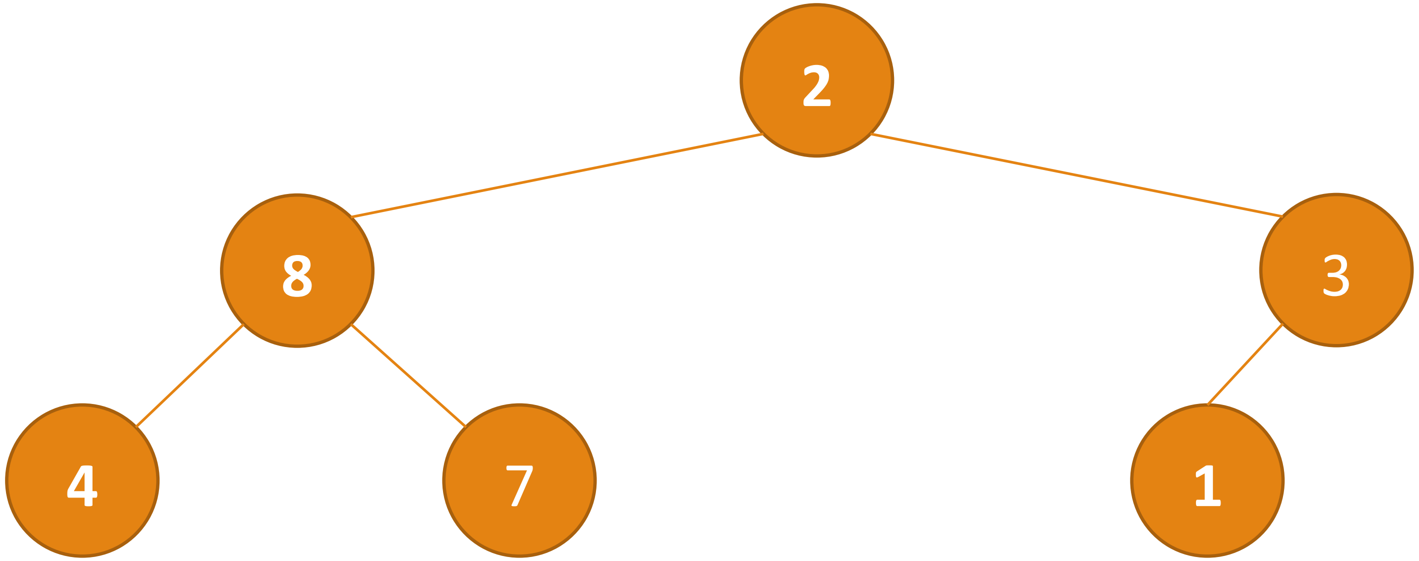
Получение массива

2	8	9	4	7	1	3	10	14	16



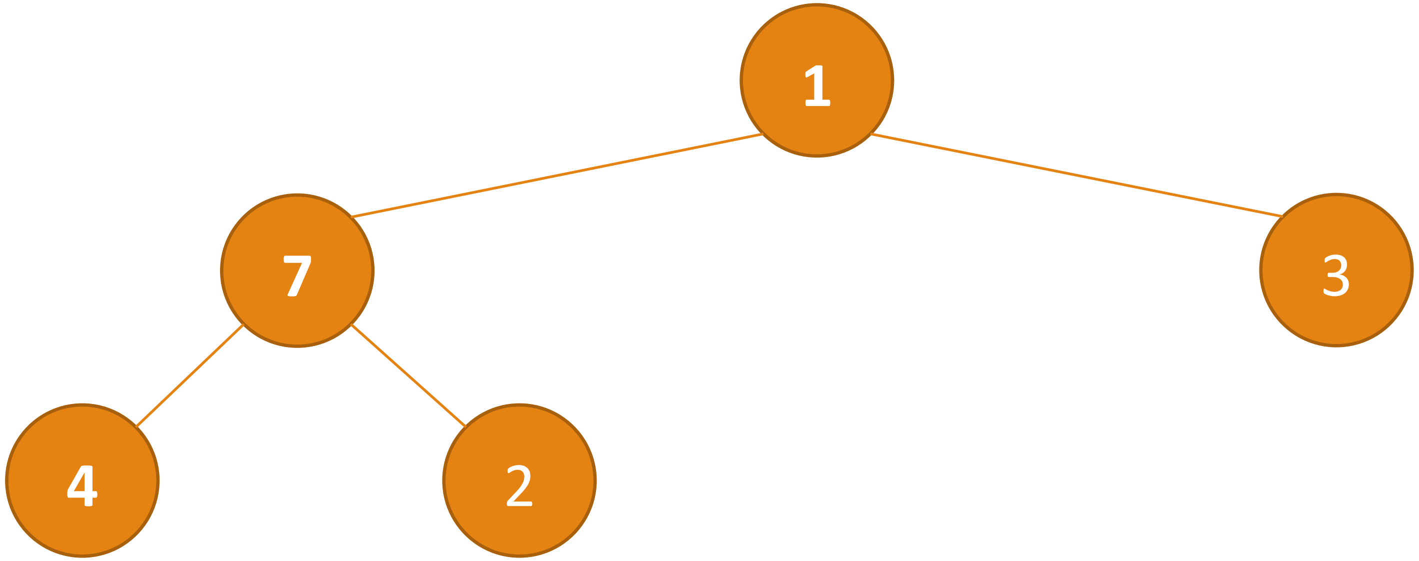
Получение массива

2	8	3	4	7	1	9	10	14	16



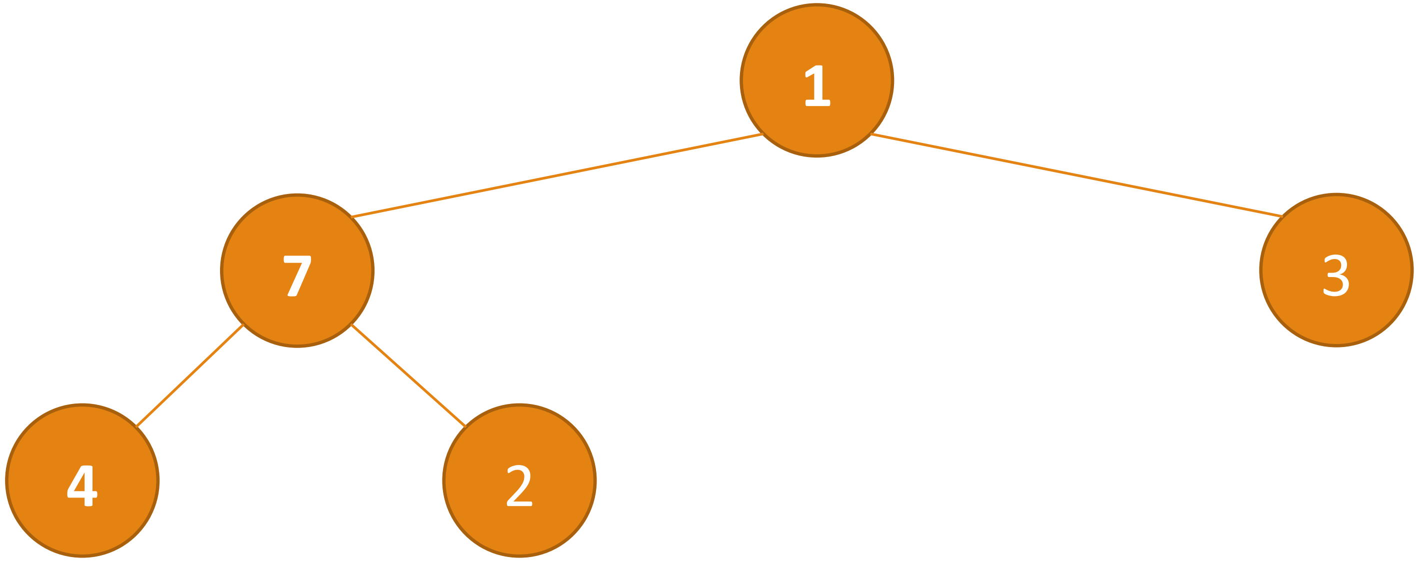
Получение массива

1	7	3	4	2	8	9	10	14	16



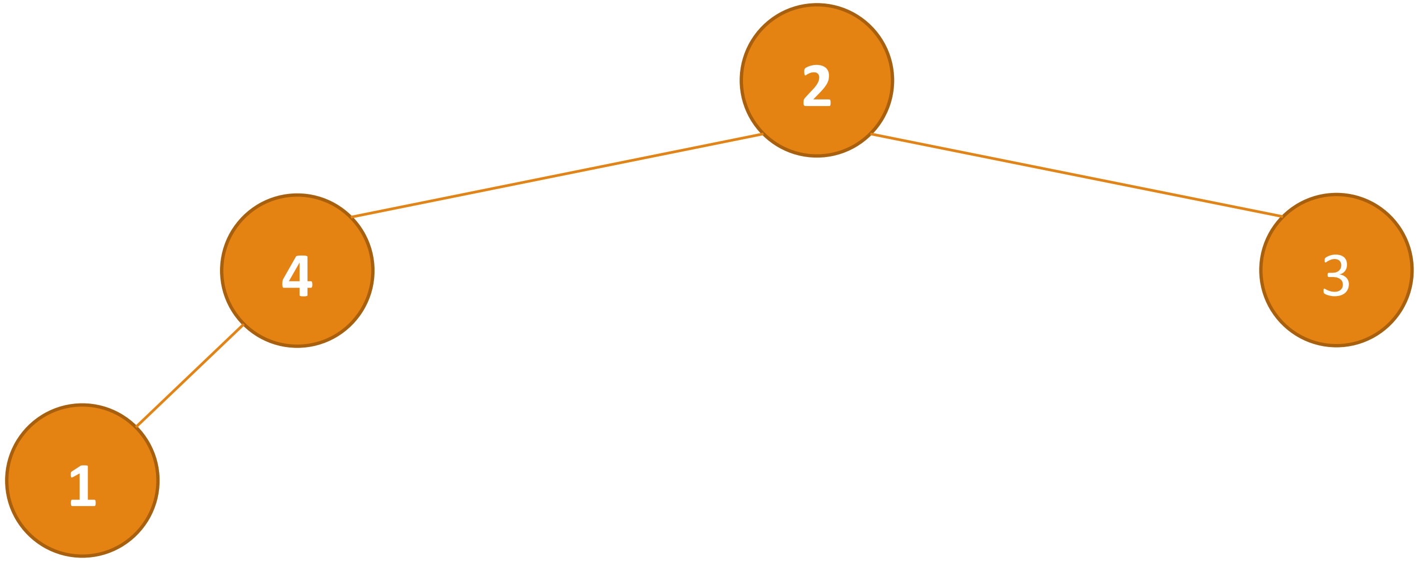
Получение массива

1	7	3	4	2	8	9	10	14	16



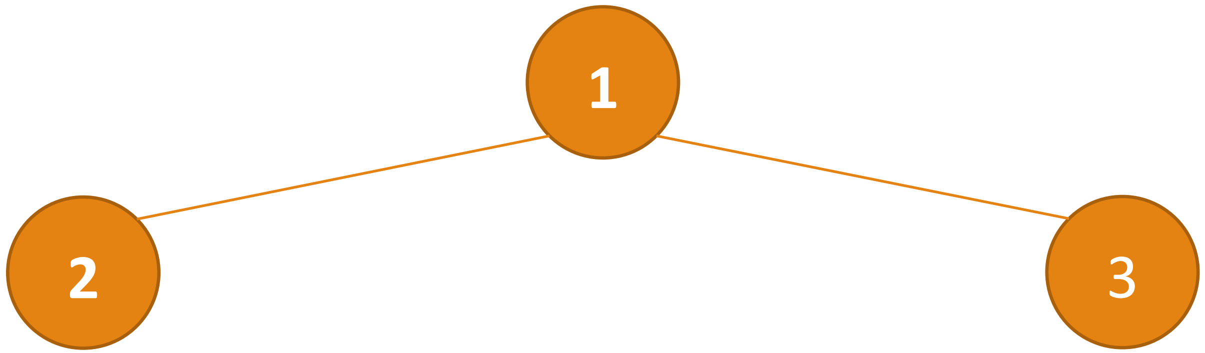
Получение массива

2	4	3	1	7	8	9	10	14	16



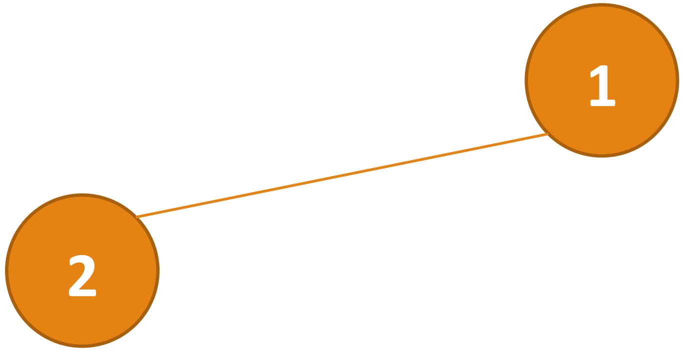
Получение массива

1	2	3	4	7	8	9	10	14	16



Получение массива

1	2	3	4	7	8	9	10	14	16



Получение массива

1	2	3	4	7	8	9	10	14	16

1

Получение массива

- По условию пирамиды наибольшие элемент находится на первом месте. Меняем его местами с последним (где он и должен быть), укорачиваем пирамиду на 1 и восстанавливаем ее, утапливая элемент оказавшийся на вершине.
- Продолжаем пока не останется один элемент в пирамиде.

Пирамидальная сортировка

Анализ пирамидальной сортировки

- Утопляем элемент в худшем случае за **O(logN)**

Анализ пирамидальной сортировки

- Утопляем элемент в худшем случае за **O(logN)**
- Для построения пирамиды утопляем **N/2** элементов

Анализ пирамидальной сортировки

- Утопляем элемент в худшем случае за **O(logN)**
- Для построения пирамиды утопляем **N/2** элементов
- Для получение отсортированного массива **N** раз утопляем вершину.

Анализ пирамидальной сортировки

- Утопляем элемент в худшем случае за **O(logN)**
- Для построения пирамиды утопляем **N/2** элементов
- Для получение отсортированного массива **N** раз утопляем вершину.
- Итого **O(NlogN)** в худшем случае.

Сортировки сравнениями

Сортировка основанная на сравнениях – сортировка, оперирующая сравнением двух элементов и их перестановкой.

Может ли такая сортировка в худшем случае работать быстрее чем за **O(NlogN)**?

Нет, время работы в худшем случае любой сортировки, основанной на сравнениях: **Ω(NlogN)**

Ω(N log N)

- Зафиксируем некоторую сортировку и входной массив длины **N**, содержащий числа **1,2, … N** в некотором порядке.
- Вариантов входного массива **N!**.
- Предположим, что сортировка во время своей работы делает операций сравнения.
- Значит, у алгоритма возможно максимум различных путей исполнения.

$\Omega(N \log N)$

- Так как алгоритм корректно сортирует все входные массивы за **k** сравнений, то должно быть не меньше **N!**

(формула Стирлинга)

Сортировка за линейное время

- Для произвольных ключей доказали: количество сравнений не менее
- А что, если ключи не произвольные?
- Сколько операций нужно, чтобы отсортировать последовательность битов?

Сортировка подсчетом (Counting sort)

- Предположим, что все ключи – целые числа **0…K-1**.
- Заведем массив счетчиков **C[k]** размером **K**, изначально заполненный нулями.
- Подсчитаем количество вхождений каждого ключа.
- Добавим к каждому элементу **C[k]** все предыдущие, получая: **C[k]** = количество ключей не превышающих **k**.
- Идем с конца массива элементов, ставя элемент A[n] на позицию C[Key(A[n])]-1 и уменьшаем счетчик.

Свойства сортировки подсчетом

- Временная сложность **O(N+K)**
- Стабильность
- Дополнительная память:

Вспомогательный массив длины N (если стабильность не нужна, можно обойтись без него);

Массив счетчиков.

На сегодня все…