IKI 20100: Struktur Data & Algoritma

Priority Queue & Heap

Ruli Manurung & Ade Azurat

(acknowledgments: Denny, Suryana Setiawan)

Fasilkom UI

Review

- Complete binary tree:
  - sebuah tree adalah terisi penuh (complete), kecuali pada level terbawah yang terisi dari kiri ke kanan.

A	B	C	D	E	F	G	H	I	J			
0	1	2	3	4	5	6	7	8	9	10	11	12

Priority Queue

- Sebuah queue dengan perbedaan aturan sebagai berikut:
  - operasi *enqueue* tidak selalu menambahkan elemen pada akhir *queue*,
  - namun, meletakkannya sesuai urutan prioritas.

Binary Heap

- Hanya diperbolehkan mengakses (membaca) item yang minimum.
- operasi dasar:
  - **menambahkan** item baru dengan kompleksitas waktu worst-case yang logaritmik.
  - **menghapus** item yang minimum dengan kompleksitas waktu worst-case yang logaritmik.
  - **mencari** item yang minimum dengan kompleksitas waktu konstan.

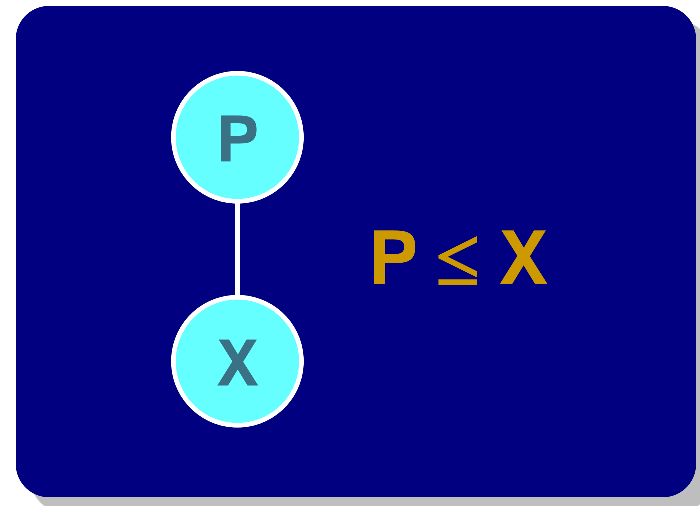
Properties (Aturan)

- Structure Property
  - Data disimpan pada complete binary tree

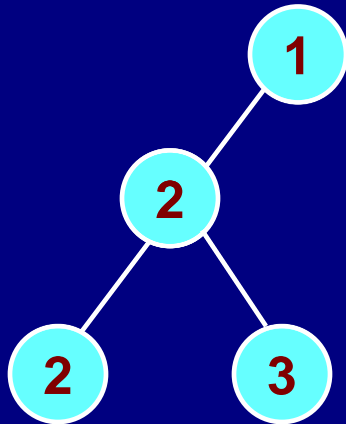
    ⇒ tree selalu balance.

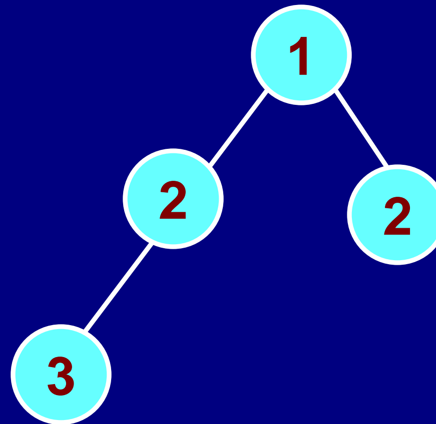
      ⇒ seluruh operasi dijamin O(log n) pada worst case

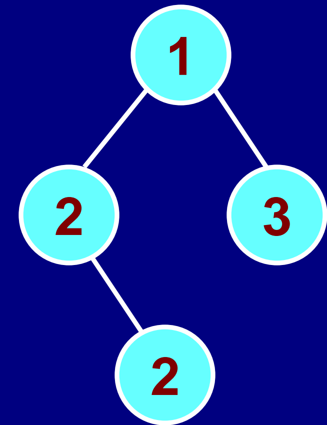
    ⇒ data disimpan menggunakan array atau java.util.Vector
- Ordering Property
  - Heap Order:
    - Parent ≤ Child

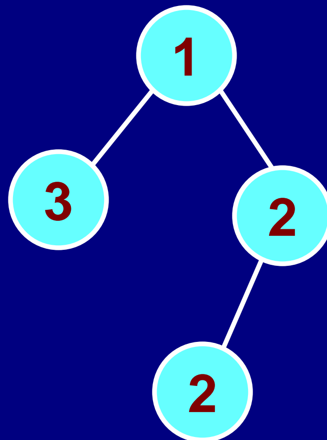


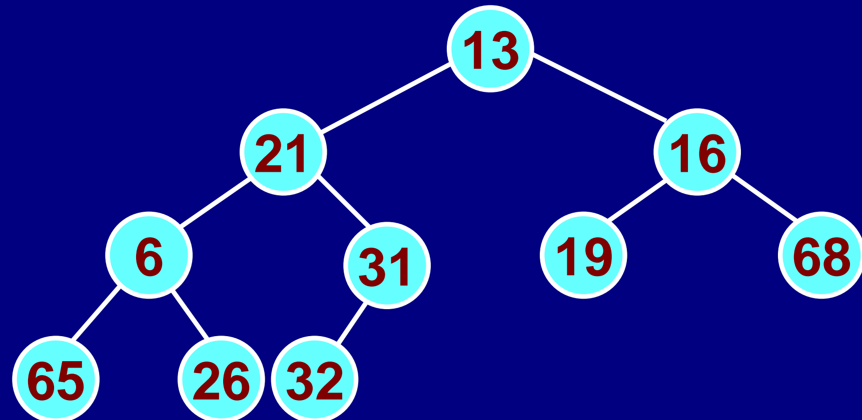
Mana yang Binary Heap?



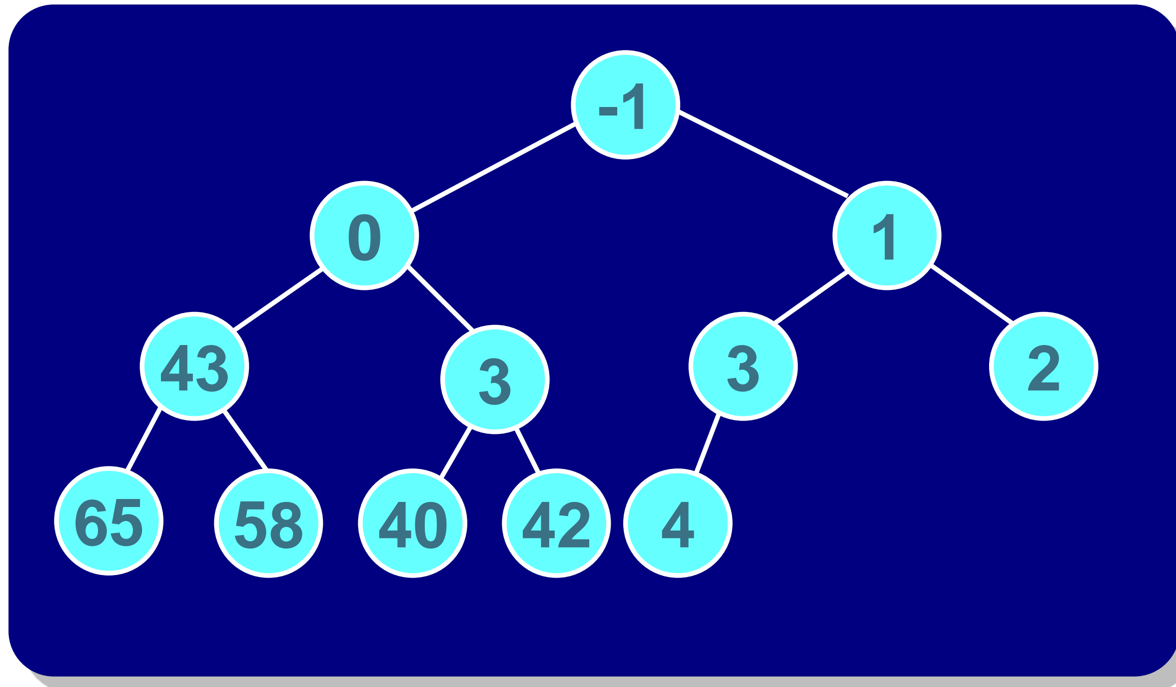




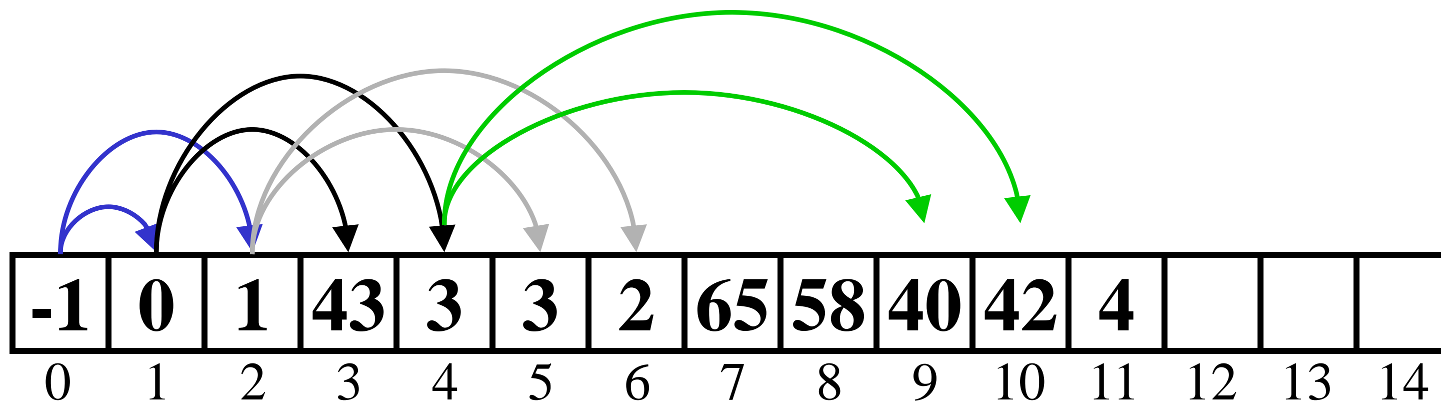




Representasi Heap

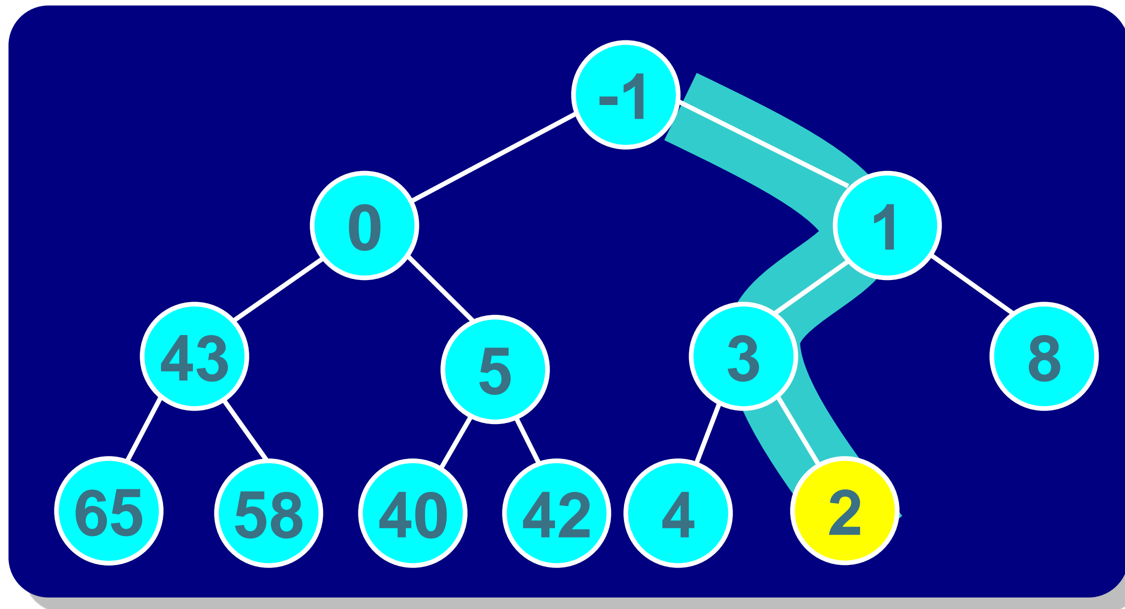


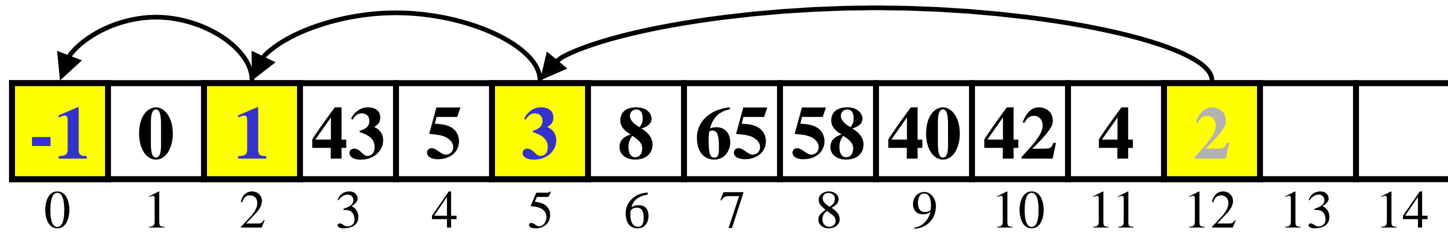
- Root pada indeks 0
- anak kiri dari i pada indeks 2i + 1
- anak kanan dari i pada indeks 2i + 2 = 2(i + 1)
- *Parent* dari i pada indeks (i - 1) / 2



Insertion

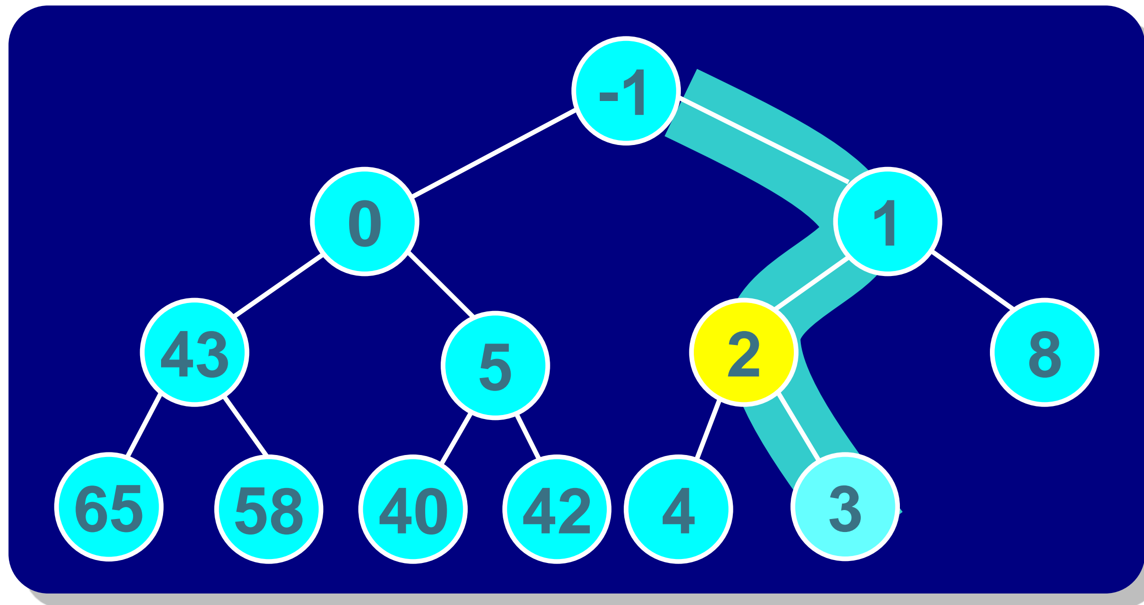
- Insert 2 (Percolate Up)

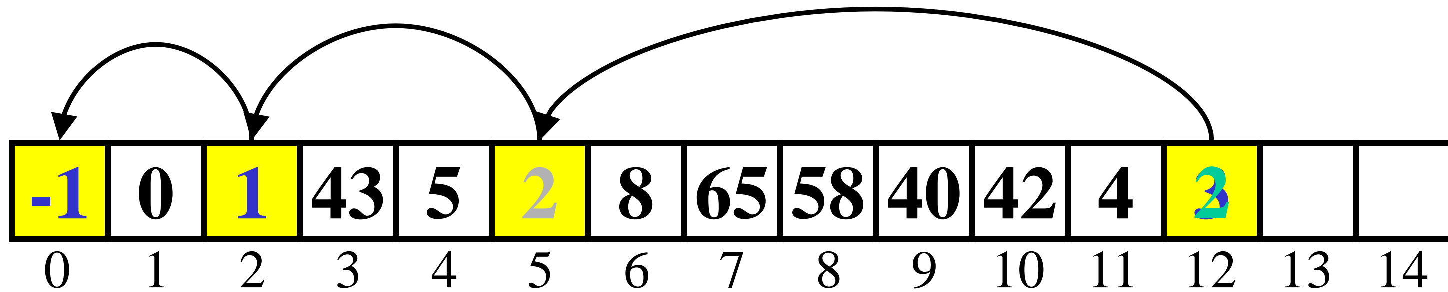




Insertion

- Insert 2 (Percolate Up)





Insertion

- Insert 14

Insertion

- Insert 14

13

21 16

24 14 19 68

65 26 32 31

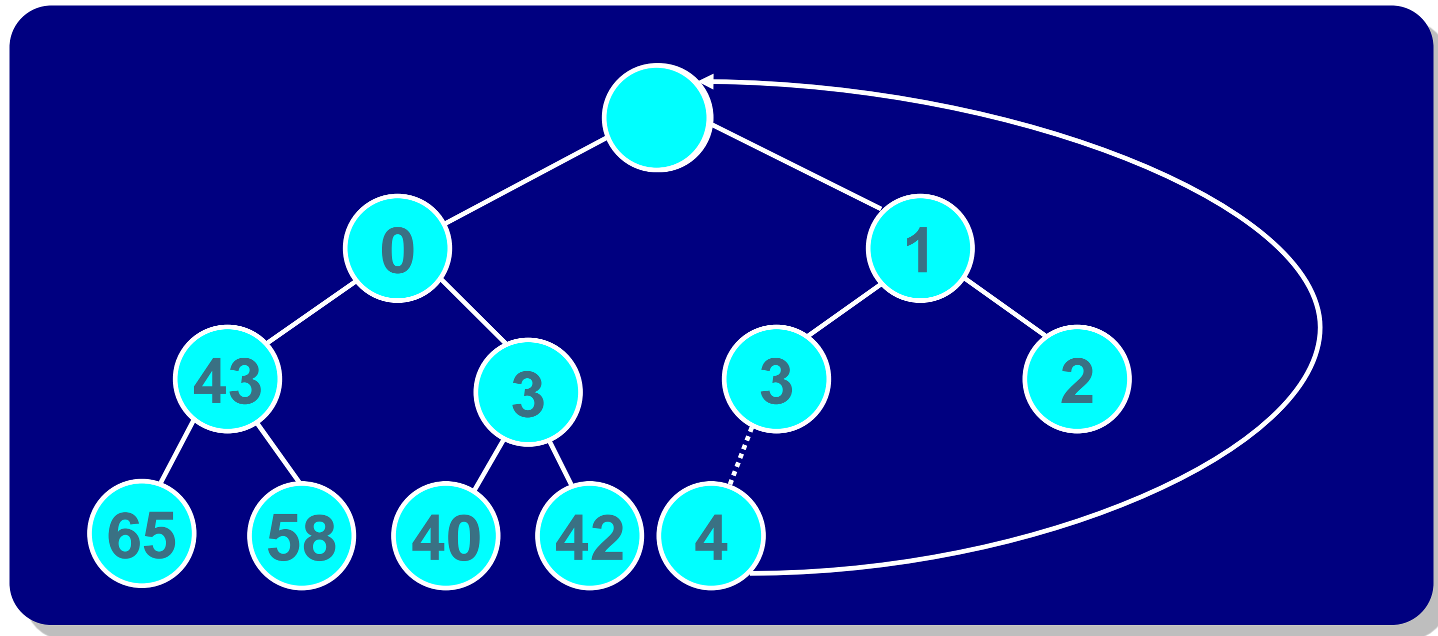
Insertion

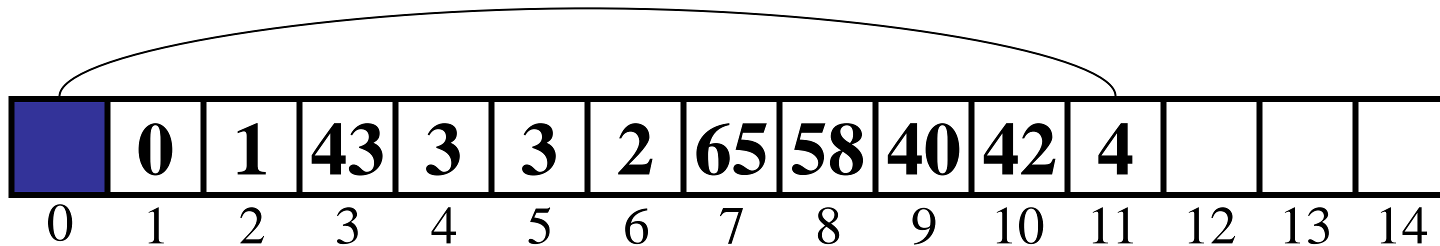
- Insert 14

13
14 16
24 21 19 68
65 26 32 31

Delete Minimum

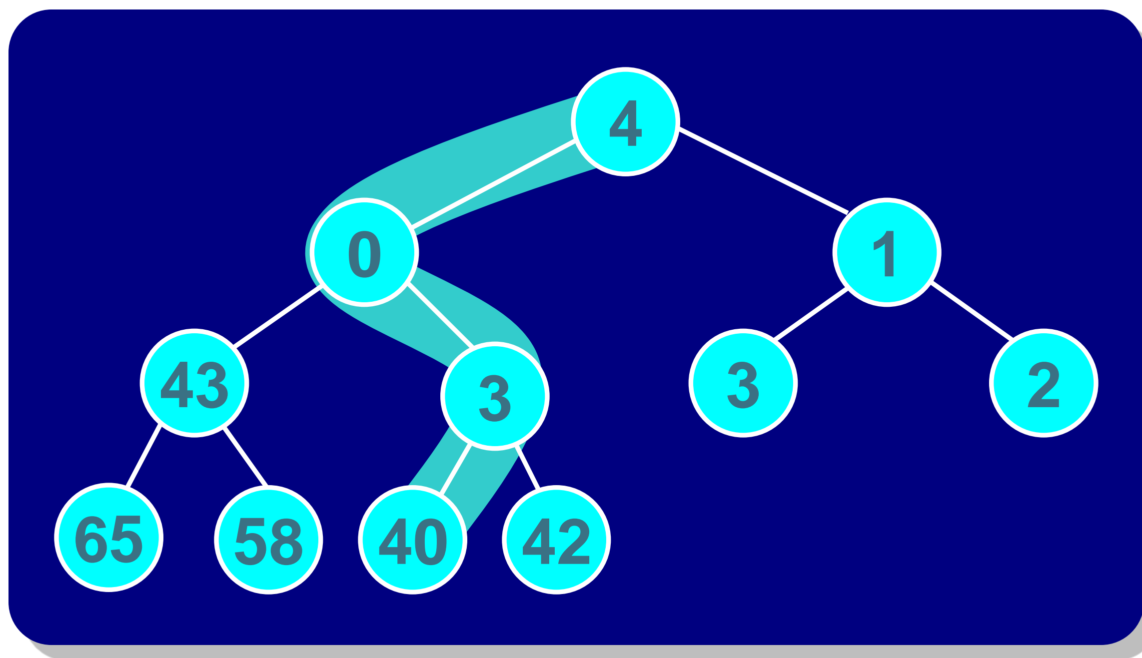
- Percolate Down

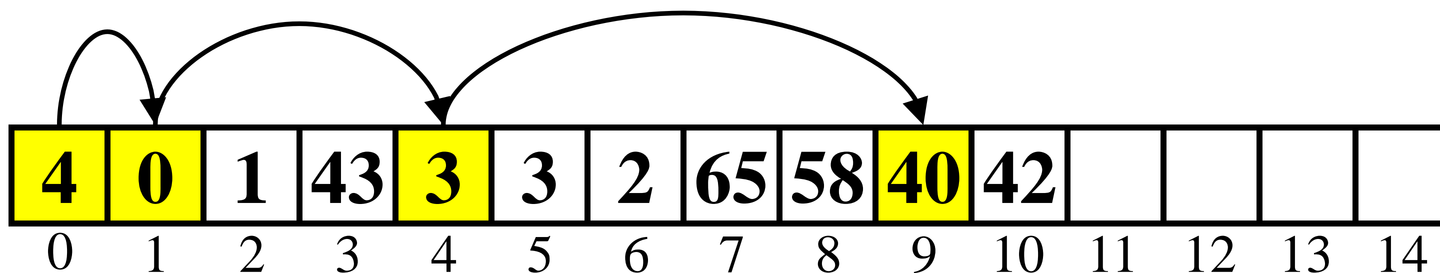




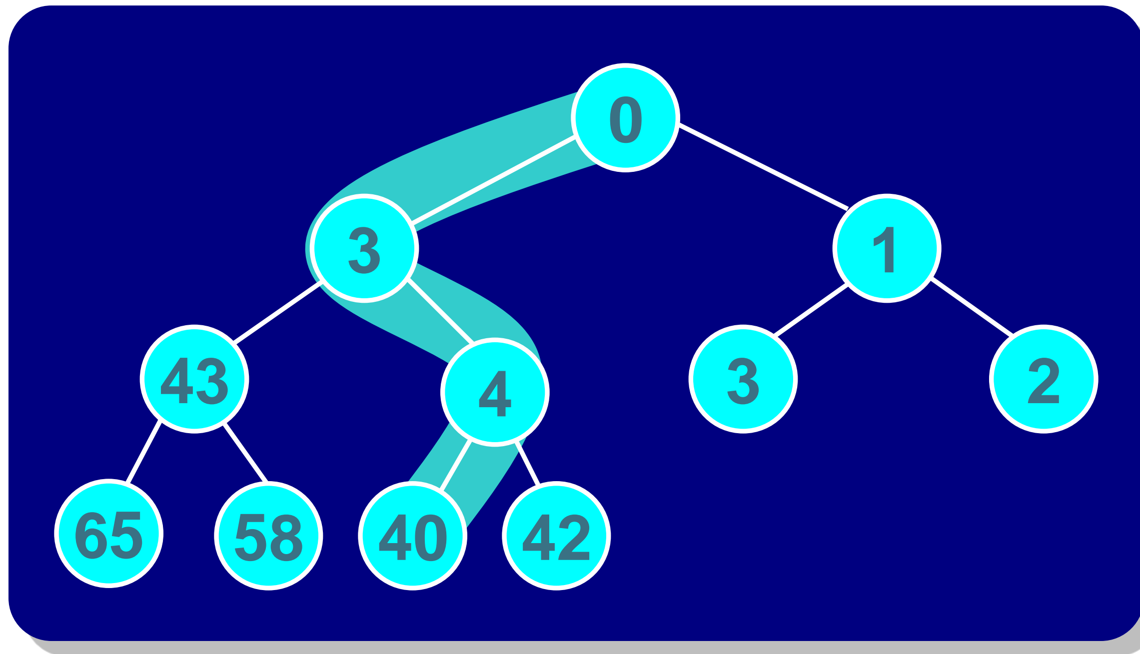
	0	1	43	3	3	2	65	58	40	42	4			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

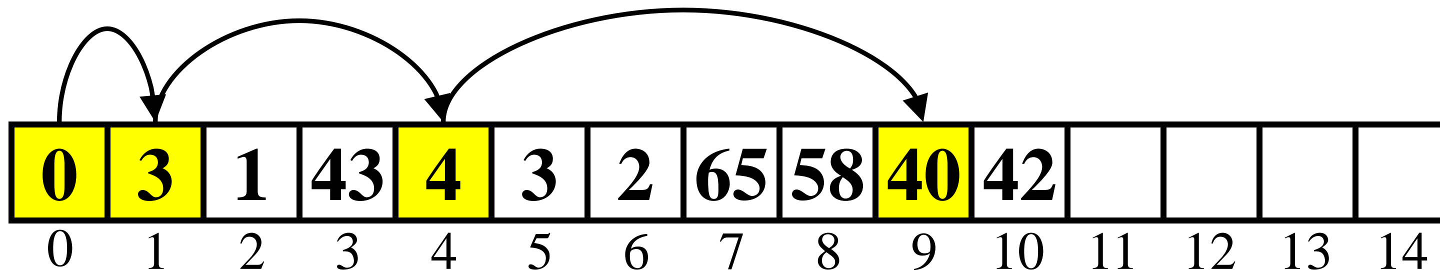
Delete Minimum



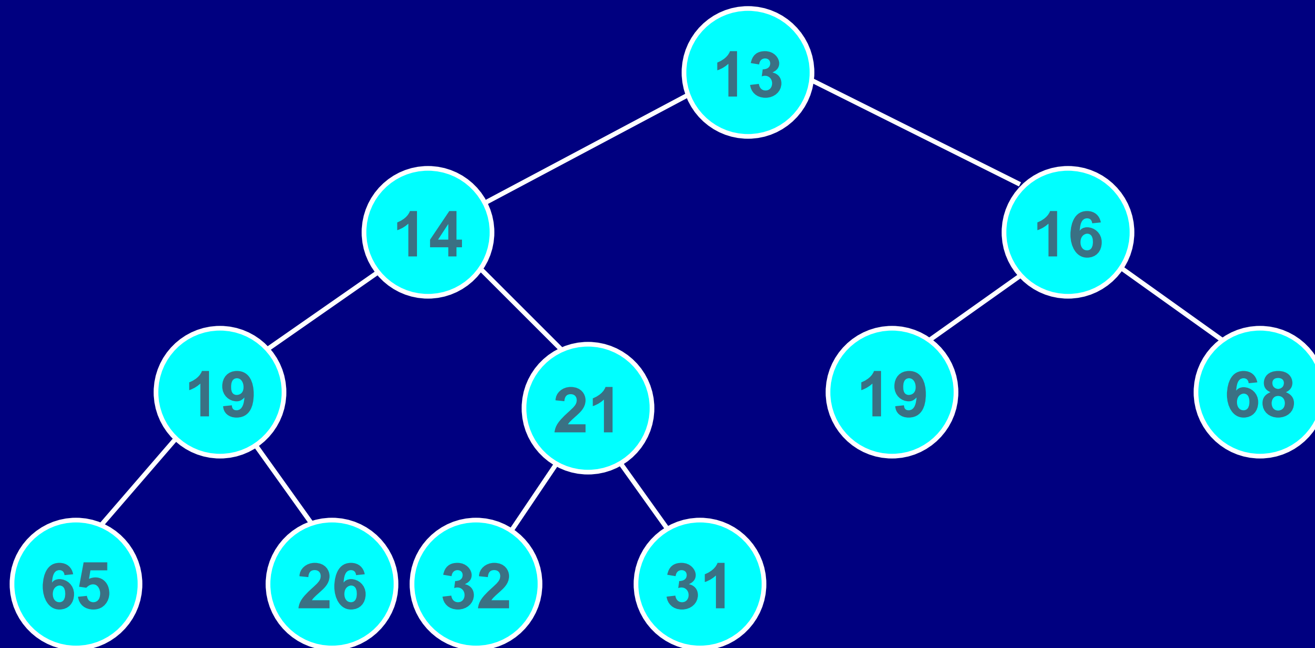


Delete Minimum: Completed



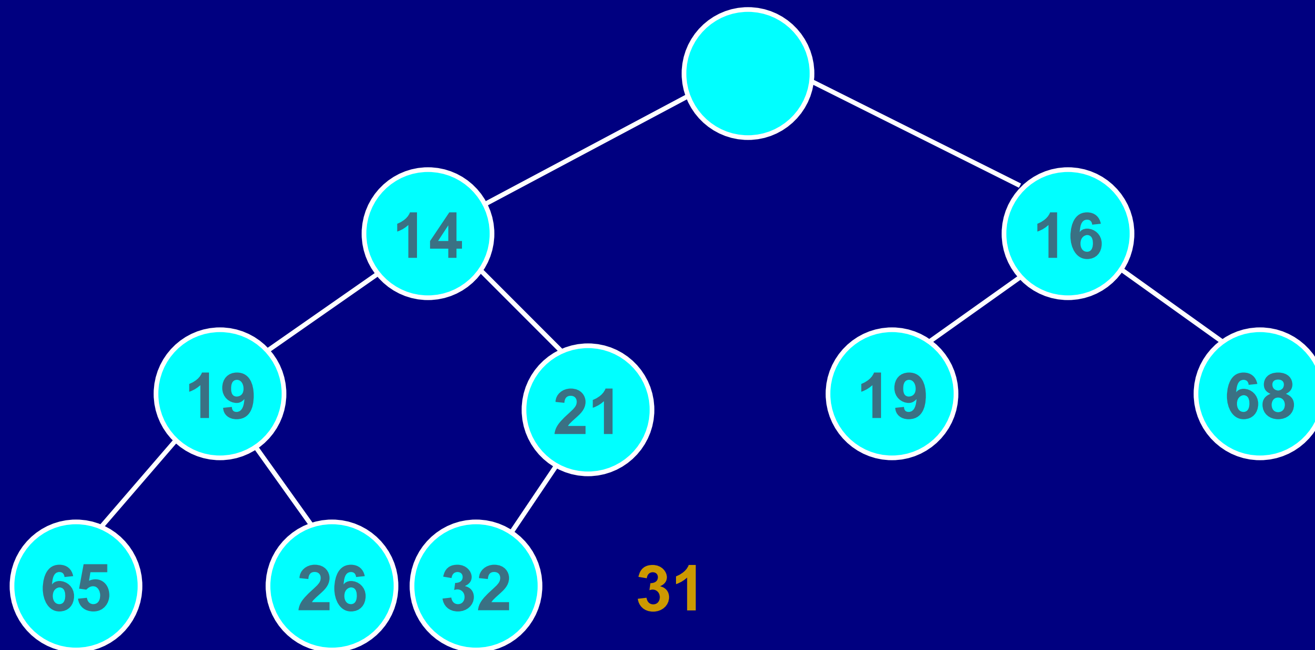


Delete Min (2)



Delete Min (2)

- Percolate Down



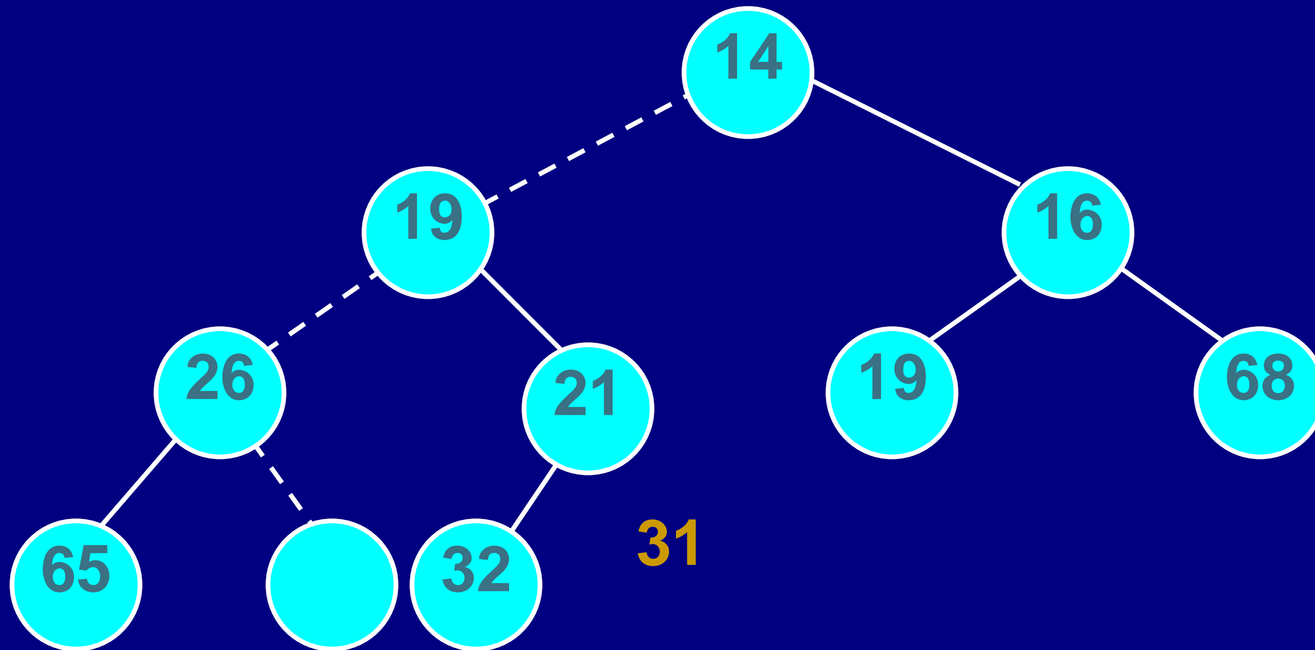
Delete Min (2)

14
16
19
21
19
68
65
26
32
31

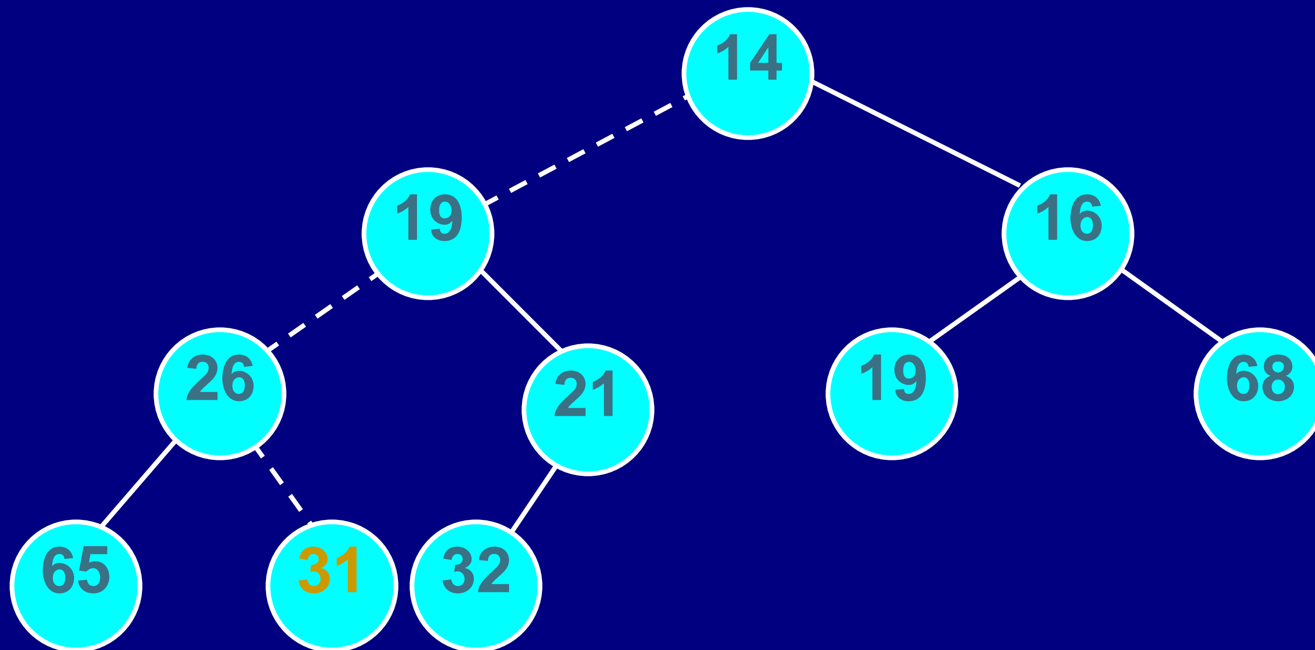
Delete Min (2)

14
19 16
21 19 68
65 26 32
31

Delete Min (2)



Delete Min (2)



Latihan

- Simulasi operasi-operasi berikut pada *Min Binary Heap*:
  - Insert
    - 40, 20, 5, 55, 76, 31, 3
  - Delete Min
  - Delete Min
  - Insert
    - 10, 22
  - Delete Min
  - Delete Min
- Gambarkan isi Binary Heap, setelah operasi terakhir.

Heap Constructor

```
public class VectorHeap implements PriorityQueue
{
    protected Vector data;

    public VectorHeap() { data = new Vector(); }

    public VectorHeap(Vector v) {
        int i;
        data = new Vector(v.size());
        // we know ultimate size
        for (i = 0; i < v.size(); i++) {
            // add elements to heap
            add((Comparable) v.elementAt(i));
        }
    }
```

Heap's Methods

```
protected static int parentOf(int i)  {
    return (i - 1) / 2;
}

protected static int leftChildOf(int i) {
    return 2 * i + 1;
}

protected static int rightChildOf(int i) {
    return 2 * (i + 1);
}

public Comparable peek() {
    // findMin
    return (Comparable) data.elementAt(0);
}
```

Removal & Insertion

```
public Comparable remove()
{
     Comparable minVal = peek();
     data.setElementAt (
          data.elementAt (data.size() - 1), 0);
     data.setSize (data.size() - 1);
     if (data.size() > 1) pushDownRoot(0);
     return minVal;
 }

 public void add(Comparable value) {
     data.addElement (value);
     percolateUp (data.size() - 1);
 }
```

Percolate Down

```
protected void pushDownRoot (int root)
{
    int heapSize = data.size();
    Comparable value = (Comparable)
        data.elementAt (root);
    while (root < heapSize) {
        int childpos = leftChildOf(root);
        if (childpos < heapSize) {
            // choose the bigger child
            if ((rightChildOf(root) < heapSize) &&
                (((Comparable) (data.elementAt(
                childpos+1))).compareTo
                ((Comparable) (data.elementAt(
                childpos))) < 0))
            {
                childpos++;
            }
```

Percolate Down

```
            if (((Comparable)(data.elementAt(
                childpos))).compareTo (value) < 0)
            {
                data.setElementAt (
                    data.elementAt(childpos), root);
                root = childpos; // keep moving down
            } else { // found right location
                data.setElementAt(value, root);
                return;
            }
        } else { // at a leaf! insert and halt
            data.setElementAt (value, root);
            return;
        }
    }
}
```

Percolate Up

```
protected void percolateUp (int leaf)
{
    int parent = parentOf(leaf);
    Comparable value =
        (Comparable)(data.elementAt(leaf));
    while (leaf > 0 && (value.compareTo
        ((Comparable)(data.elementAt(parent)))
        < 0))
    {
        data.setElementAt (data.elementAt
            (parent), leaf);
        leaf = parent;
        parent = parentOf(leaf);
    }
    data.setElementAt(value,leaf);
}
```

Heap's Methods

```
public boolean isEmpty()
{
  return data.size() == 0;
}
public int size()
{
    return data.size();
}
public void clear()
{
    data.clear();
}
public String toString()
{
    return "<VectorHeap: " + data + ">";
}
```

Fix Heap / Heapify

- Operasi fixHeap menerima *complete tree* yang tidak memenuhi *heap order* dan memperbaikinya.
- penambahan sebanyak N dapat dilakukan dengan O(n log n)
- Operasi fix heap dapat dilakukan dengan O(n) !

92
47 21
20 12 45 63
61 17 55 37 25 64 83 73

Fix Heap / Heapify

- Percolate down 6

92

47 21

20 12 45 63

61 17 55 37 25 64 83 73

Fix Heap / Heapify

- Percolate down 5

92
47 21
20 12 25 63
61 17 55 37 45 64 83 73

Fix Heap / Heapify

- Percolate down 4

92

Tree nodes (level order): 92; 47, 21; 20, 12, 25, 63; 61, 17, 55, 37, 45, 64, 83, 73

Fix Heap / Heapify

- Percolate down 3

92

47 21

17 12 25 63

61 20 55 37 45 64 83 73

Fix Heap / Heapify

- Percolate down 2

92
47 21
17 12 25 63
61 20 55 37 45 64 83 73

Fix Heap / Heapify

- Percolate down 1

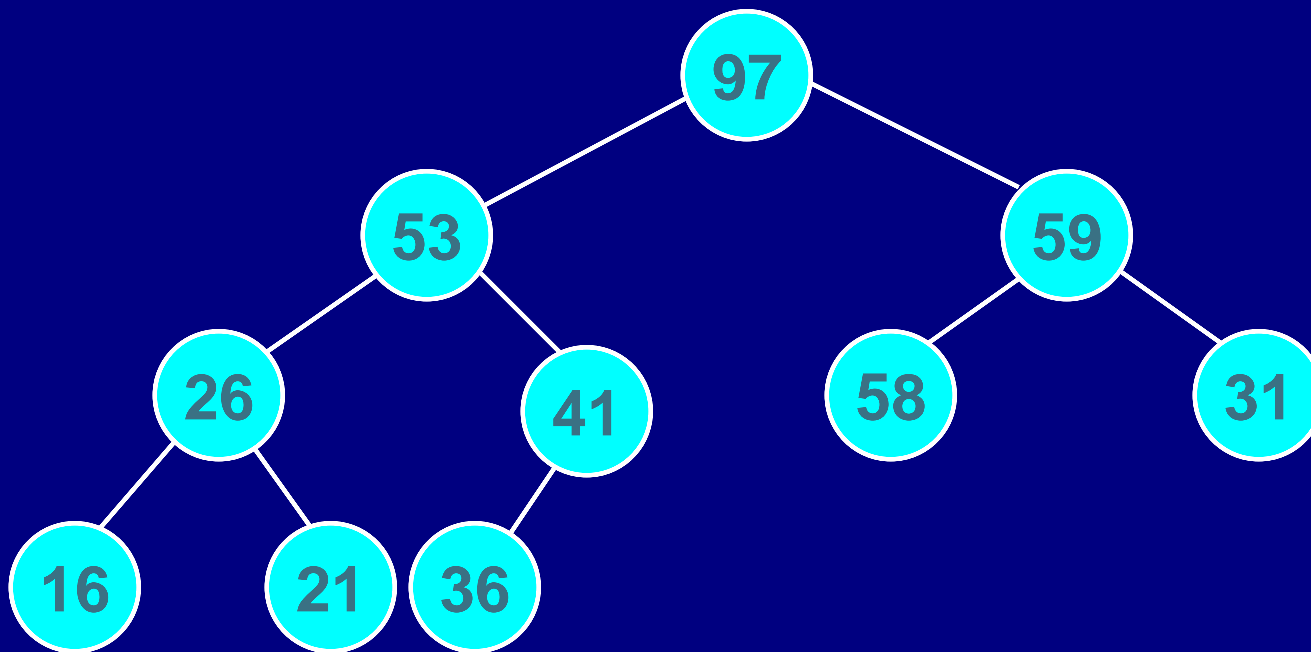
92
12 21
17 37 25 63
61 20 55 47 45 64 83 73

Fix Heap / Heapify

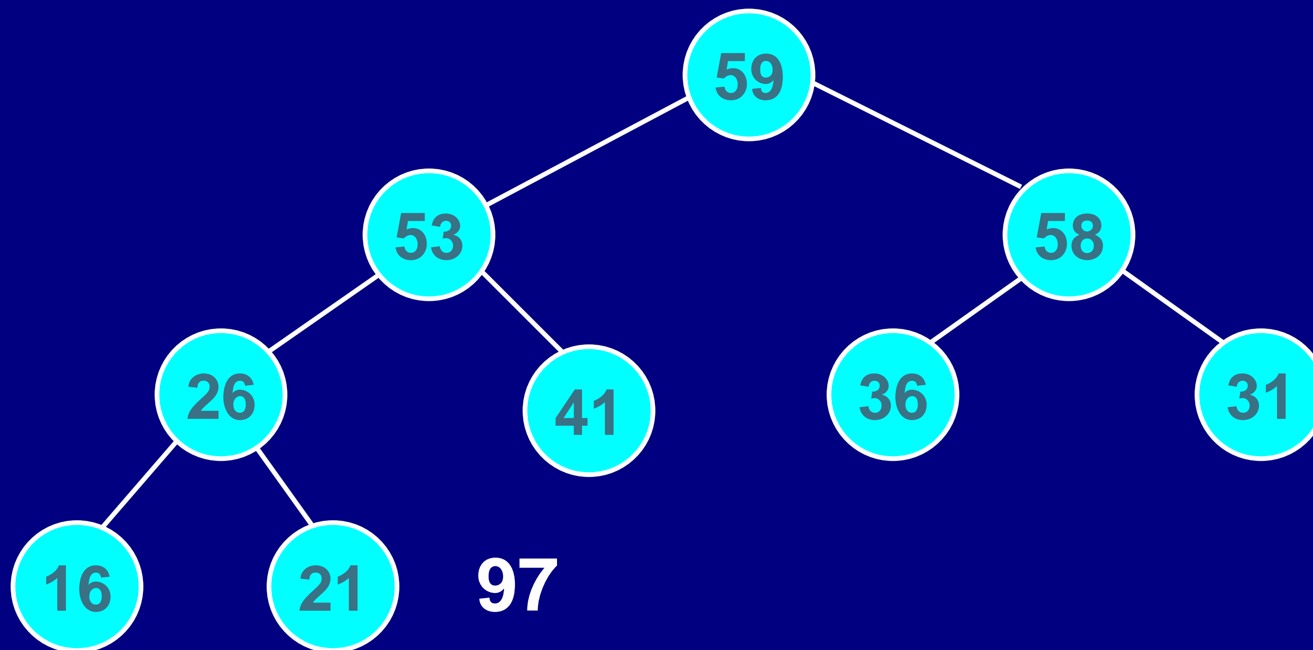
- Percolate down 0

12
17 21
20 37 25 63
61 92 55 47 45 64 83 73

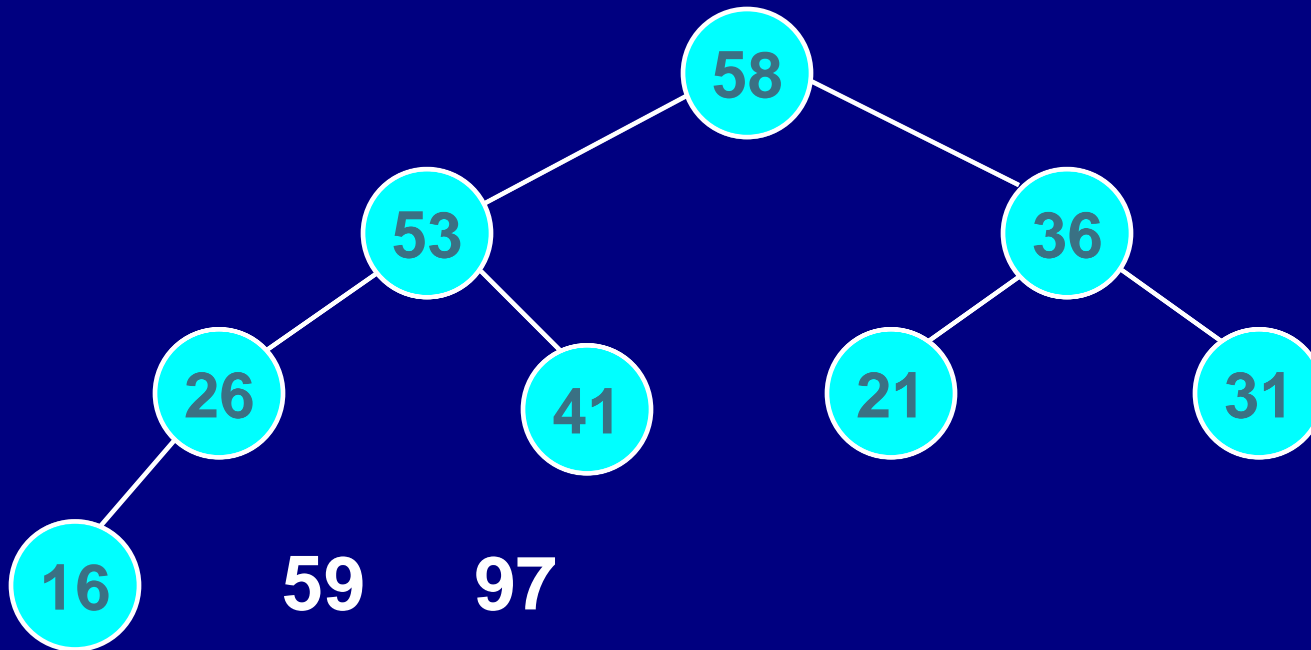
Max Heap



Heap setelah *deleteMax* pertama



Heap setelah *deleteMax* kedua



58	53	36	26	41	21	31	16	59	97				
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Heap Sort

1. Buat sebuah heap tree
2. ambil elemen pada posisi root dari heap setiap pengambilan elemen, dan lakukan heapify.

Rangkuman

- Priority queue dapat diimplementasi kan menggunakan binary heap
- Aturan-aturan pada binary heap
  - structure property
    - complete binary tree
  - ordering property
    - Heap order: Parent ≤ Child
- Operasi pada binary heap
  - insertion: kompleksitas waktu *O(log n)* pada worst case
  - find min: kompleksitas waktu *O(1)*
  - delete min: kompleksitas waktu *O(log n)* pada worst case
- Binary heap dapat digunakan untuk *sorting*