HEAP

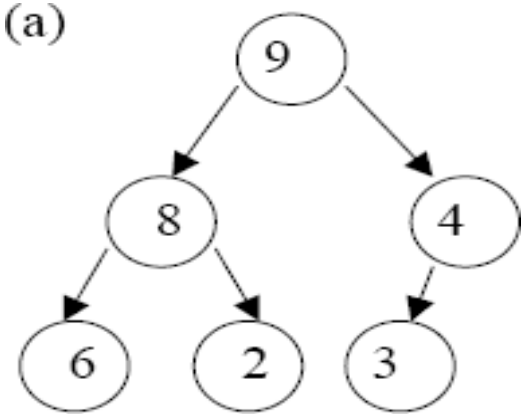
Part of Tree Topic

Binary Heap

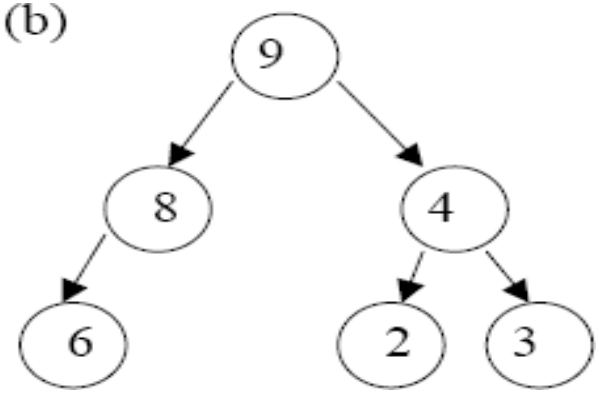
- Heap data structure is a binary tree with the following properties:
  - Heap is a complete binary tree where each level of the tree is filled, except possibly the bottom level. At this level, it is filled from left to right.
  - It satisfies the heap-order property: The data item stored in each node is greater than or equal to the data items stored in its children.

Which one is Heap ?

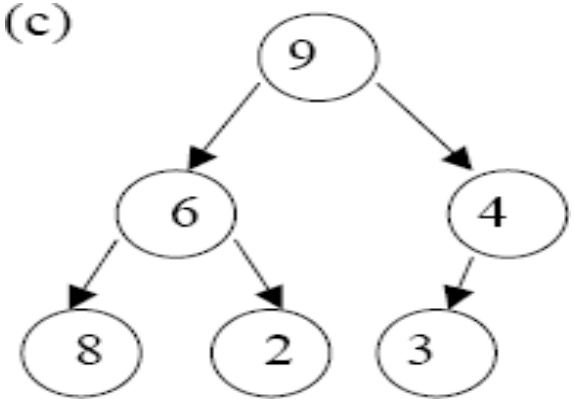
(a)



(b)

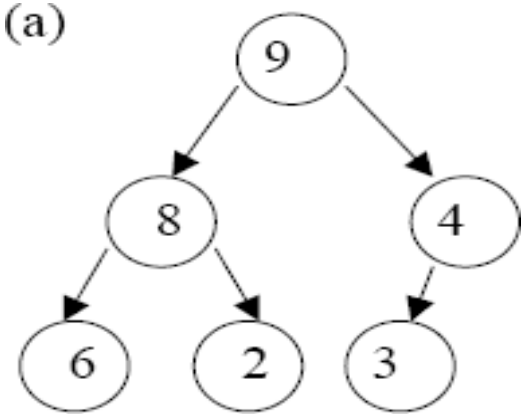


(c)



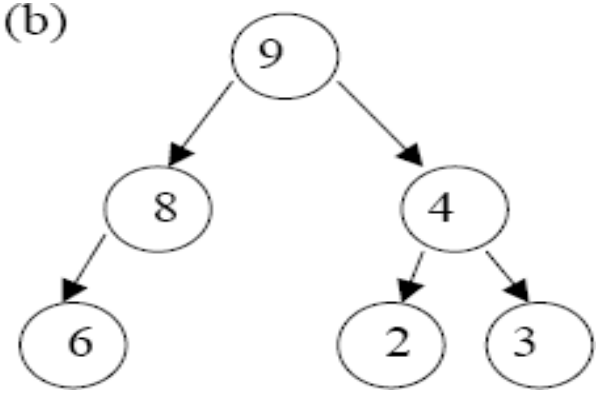
Which one is Heap ?

(a)



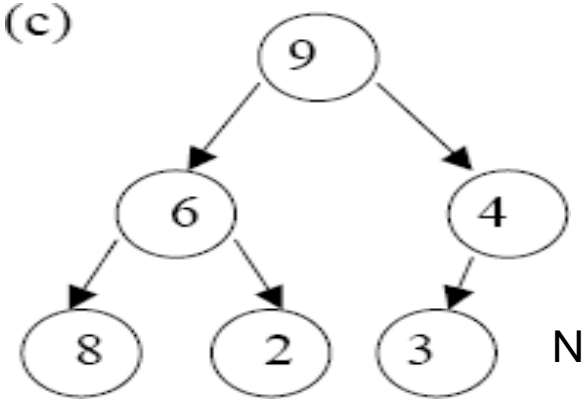
This is a Heap ✅

(b)



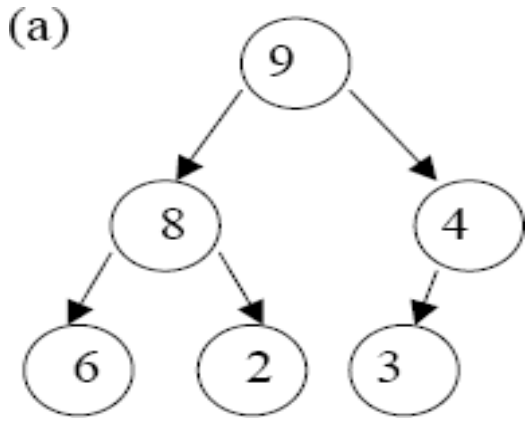
This is a Heap ✅

(c)



Not a Heap ❌

Which one is Heap ?



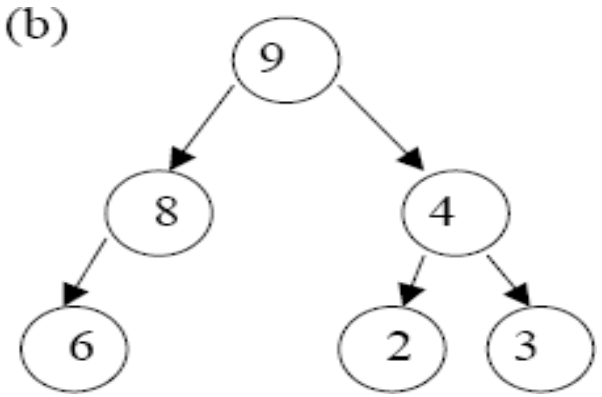
This is a Heap ✅

Root = 9 → lebih besar dari 8 & 4
8 > 6 & 2, 4 > 3
Lengkap dan ikut heap-order property → Betul!

Which one is Heap ?

Sama macam (a), cuma struktur visual sikit berbeza (left-heavy), tapi masih:

- Complete Binary Tree ✅
- Heap-order property ✅



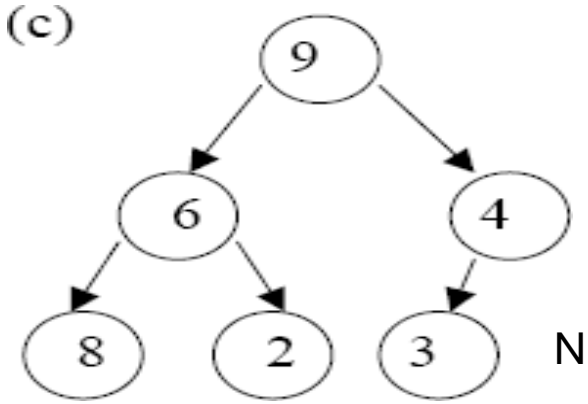
This is a Heap ✅

Which one is Heap ?

Nampak macam lengkap, tapi:

- Node 6 ada anak 8, tapi 6 < 8 → melanggar heap-order property
- 6 sepatutnya ≥ 8 kalau ikut Max-Heap

Jadi memang bukan Heap.



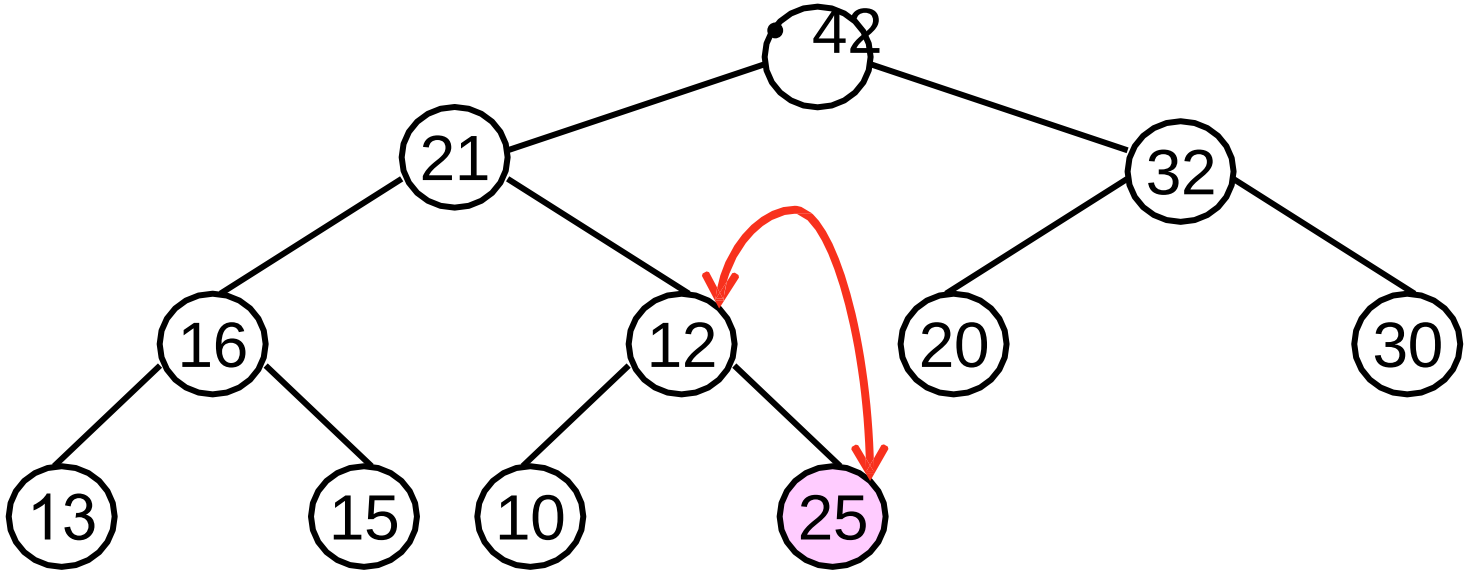
Not a Heap ✗

Basic Heap algorithm

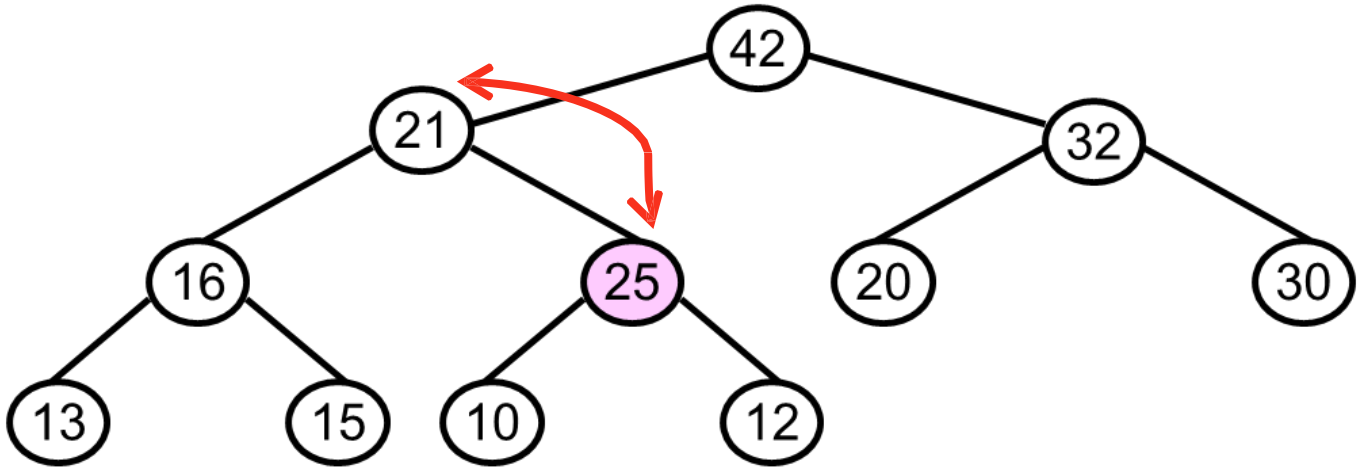
- One of the heap operation is to convert a complete binary tree to a heap.
- The basic algorithm in converting a complete binary tree to a heap are:-
    - ReheapUp
    - ReheapDown

ReheapUp

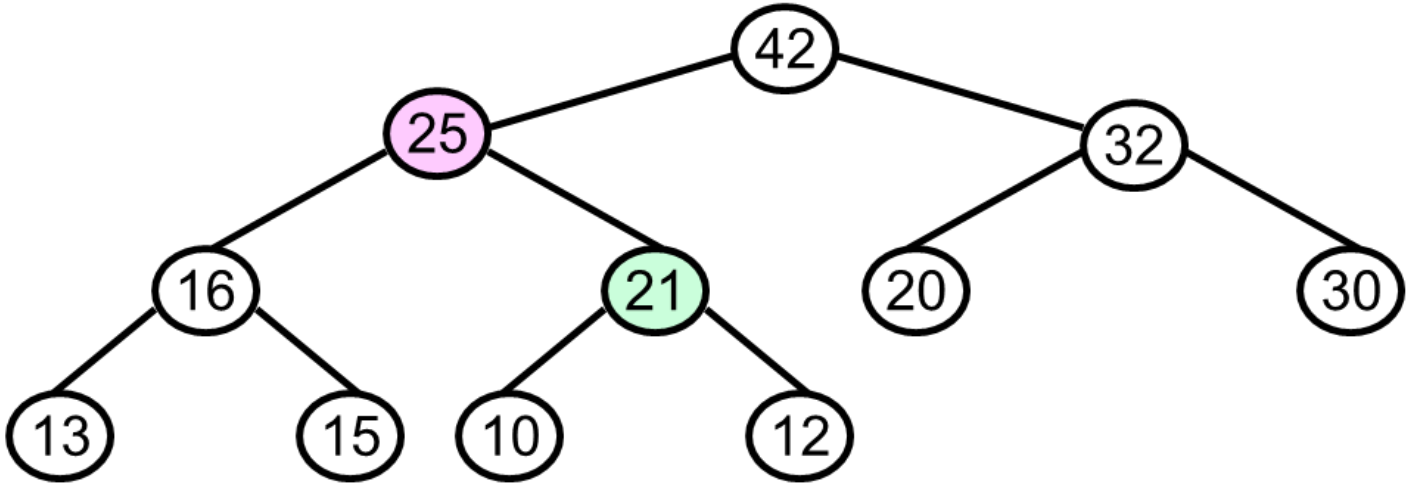
- Repairs a “broken” heap by floating the last element up the tree until it is in its correct location in the heap.
- The tree below is not a heap due to the node 25. It not fulfilling the 2nd properties



ReheapUp



Last element (25)
Moved up

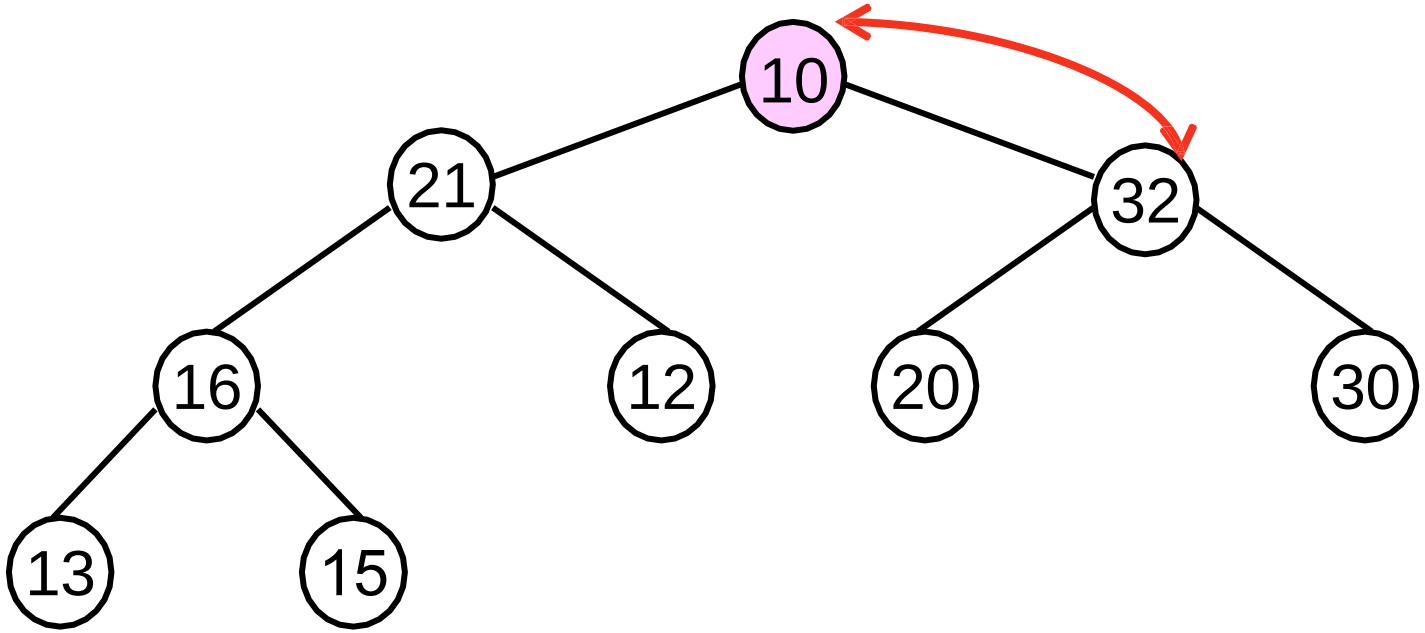


Moved up again
Tree now is a heap

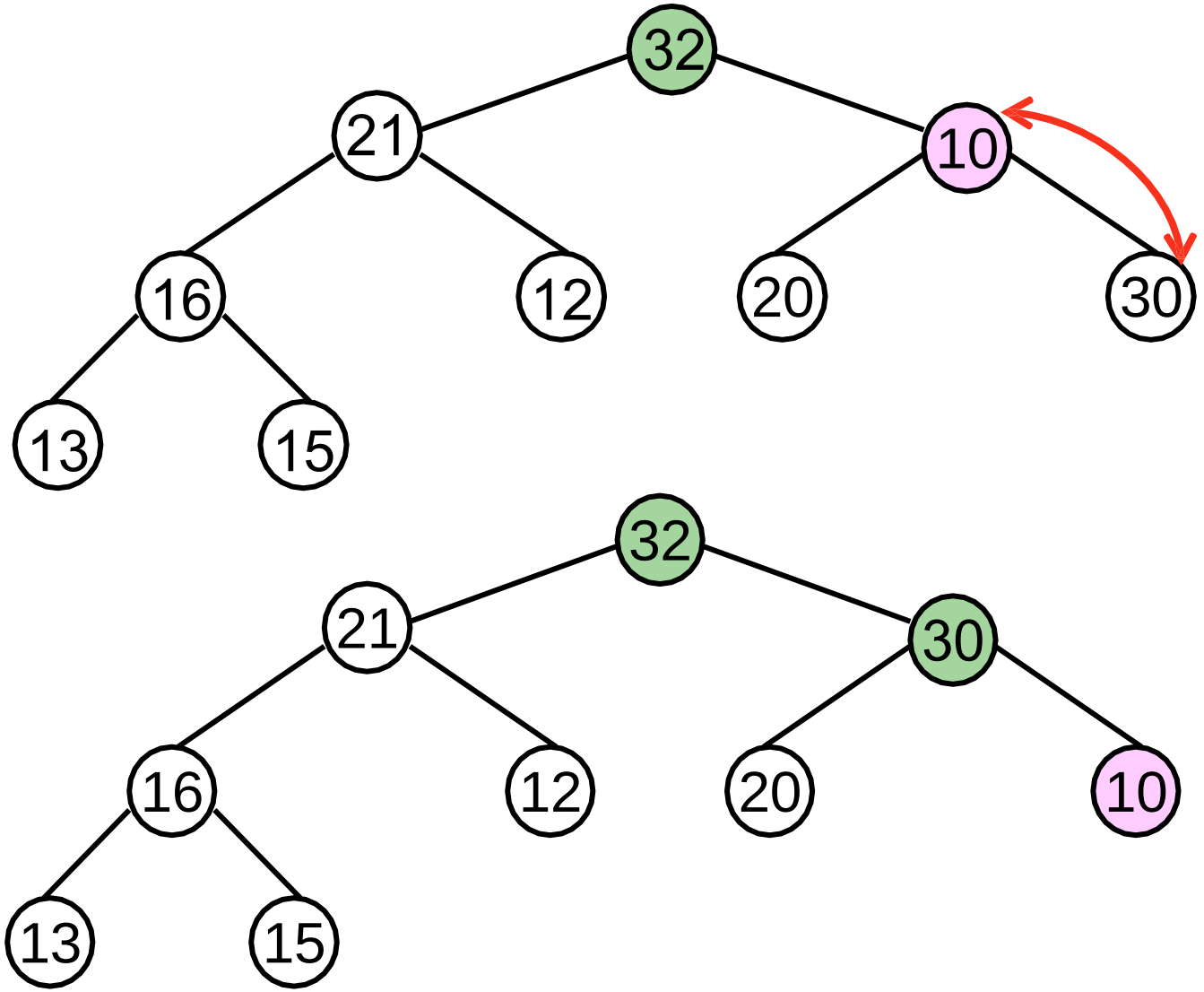
ReheapDown

Repairs a broken heap by pushing the root down the tree until it is in its correct position in the heap

The tree below is not a heap due to the node 10. It not fulfilling the 2nd properties



ReheapDown



Root moved down (right)

Moved down again
Tree now is a heap

Summary

- A heap is a complete binary tree, where the entry at each node is greater than or equal to the entries in its children.
- To add an entry to a heap, place the new entry at the next available spot, and perform a **reheapification** upward.
- To remove the biggest entry, move the last node onto the root, and perform a **reheapification** downward.

Coding

```cpp
#include <iostream>
#include <vector>
using namespace std;

class MaxHeap {
    vector<int> heap;

    void reheapUp(int index) {
        int parent = (index - 1) / 2;
        if (index > 0 && heap[index] > heap[parent]) {
            swap(heap[index], heap[parent]);
            reheapUp(parent);
        }
    }

    void reheapDown(int index) {
        int left = 2 * index + 1, right = 2 * index + 2, largest = index;
        if (left < heap.size() && heap[left] > heap[largest]) largest = left;
        if (right < heap.size() && heap[right] > heap[largest]) largest = right;
        if (largest != index) {
            swap(heap[index], heap[largest]);
            reheapDown(largest);
        }
    }

public:
    void insert(int val) {
        heap.push_back(val);
        reheapUp(heap.size() - 1);
    }

    int removeMax() {
        if (heap.empty()) return -1;
        int maxVal = heap[0];
        heap[0] = heap.back();
        heap.pop_back();
        reheapDown(0);
        return maxVal;
    }

    void printHeap() {
        for (int val : heap) cout << val << " ";
        cout << endl;
    }
};

int main() {
    MaxHeap h;
    h.insert(10); h.insert(4); h.insert(8); h.insert(15); h.insert(3);
    h.printHeap();          // Heap in array form
    cout << h.removeMax(); // Print and remove max (15)
}
```

1. insert(int val)

```
void insert(int val) {
        heap.push_back(val);
        reheapUp(heap.size() - 1);
    }
```

✅ Proses:

1. Masukkan nilai ke **belakang array** (bawah pokok).
2. Jalankan **reheapUp()** untuk pastikan susunan heap betul **(bubble up).**

🧠 Contoh:

Masukkan: 10, 4, 8, 15

Selepas 10: [10]
Selepas 4: [10, 4] // 4 < 10 → ok
Selepas 8: [10, 4, 8] // 8 < 10 → ok
Selepas 15:
Masuk belakang: [10, 4, 8, 15]
reheapUp:
15 > parent 4 → swap
15 > parent 10 → swap

Hasil: [15, 10, 8, 4]

Fungsi 1: insert(int val)

- Masukkan elemen baru ke dalam heap:
- Tambah elemen ke akhir vector.
- Reheap dari bawah ke atas (reheapUp).

- Contoh:
- Input: 10, 4, 8, 15
- → [10] → [10, 4] → [10, 4, 8] → [15, 10, 8, 4]

2. reheapUp(int index)

```
void reheapUp(int index) {
        int parent = (index - 1) / 2;
        if (index > 0 && heap[index] > heap[parent]) {
            swap(heap[index], heap[parent]);
            reheapUp(parent);
        }
    }
```

Fungsi ini gerakkan elemen ke atas jika lebih besar dari parent (untuk jaga Max-Heap).

🧠 Contoh ringkas:

insert(20) ke dalam [15, 10, 8, 4]

Masuk: [15, 10, 8, 4, 20]

Index 4 → parent(1) = 10

20 > 10 → swap → [15, 20, 8, 4, 10]

Index 1 → parent(0) = 15

20 > 15 → swap → [20, 15, 8, 4, 10]

Fungsi 2: reheapUp(int index)

- Naikkan elemen jika lebih besar dari parent:
- Bandingkan dengan parent.
- Swap dan teruskan naik.

- Contoh:
- Masukkan 20 → [20, 15, 8, 4, 10]

🗑️ 3. removeMax()

```
int maxVal = heap[0];
heap[0] = heap.back();
heap.pop_back();
reheapDown(0);
```

Fungsi untuk keluarkan elemen terbesar (root), dan betulkan semula heap.

✅ **Proses:**

- Simpan nilai root (max).
- Ganti root dengan node terakhir.
- Buang node terakhir.
- Jalankan reheapDown() dari root → betulkan heap.

🧠 Contoh:
Heap awal: [20, 15, 8, 4, 10]

removeMax() → buang 20
Ganti root dengan 10: [10, 15, 8, 4]

reheapDown:
10 < 15 → swap → [15, 10, 8, 4]

Fungsi 3: removeMax()

- Buang elemen terbesar (root) dan betulkan susunan:
- Ganti root dengan elemen terakhir.
- Buang elemen terakhir.
- Reheap dari atas ke bawah.

- Contoh:
- Sebelum: [20, 15, 8, 4, 10]
- Selepas: [15, 10, 8, 4]

4. reheapDown(int index)

```
void reheapDown(int index) {
       int left = 2 * index + 1, right = 2 * index + 2, largest = index;
       if (left < heap.size() && heap[left] > heap[largest]) largest = left;
       if (right < heap.size() && heap[right] > heap[largest]) largest = right;
       if (largest != index) {
           swap(heap[index], heap[largest]);
           reheapDown(largest);
       }
   }
```

Turunkan elemen dari root jika lebih kecil dari child.

🧠 Contoh:
Heap: [10, 4, 8]

Remove 10 → replace root dengan 8 → [8, 4]
No need to swap → already valid Max-Heap

Kalau:
Heap: [10, 4, 15]
removeMax() → 15 naik ke root → [4, 10]
4 < 10 → swap → [10, 4]

Fungsi 4: reheapDown(int index)

- Turunkan elemen jika lebih kecil dari anak:
- Bandingkan dengan anak kiri dan kanan.
- Swap dengan anak yang terbesar.
- Teruskan ke bawah jika perlu.

- Contoh:
- Sebelum: [2, 4, 8]
- Selepas: [8, 4, 2]

📋 Ringkasan Pantas

Fungsi	Tujuan
insert()	Masuk elemen baru dan reheap naik (reheapUp)
reheapUp()	Naikkan elemen jika lebih besar dari parent
removeMax()	Keluarkan root dan betulkan heap (reheapDown)
reheapDown()	Turunkan elemen jika lebih kecil dari anak