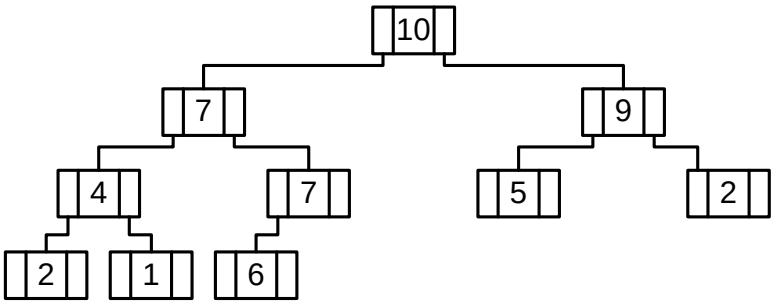
HEAP

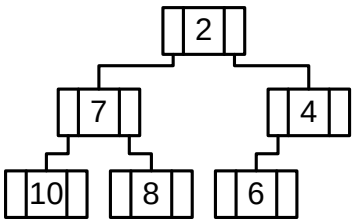
Heap adalah sebuah binary tree dengan ketentuan sebagai berikut :

- Tree harus complete binary tree
  - Semua level tree mempunyai simpul maksimum kecuali pada level terakhir.
  - Pada level terakhir, node tersusun dari kiri ke kanan tanpa ada yang dilewati.
- Perbandingan nilai suatu node dengan nilai node child-nya mempunyai ketentuan berdasarkan jenis heap, diantaranya :
  - Max Heap mempunyai ketentuan bahwa nilai suatu node lebih besar atau sama dengan (>=) dari nilai childnya.
  - Min Heap mempunyai ketentuan bahwa nilai suatu node lebih kecil atau sama dengan (<=) dari nilai childnya.

Contoh :



Gambar 1. Contoh heap maksimum (Max Heap)



Gambar 2. Contoh heap minimum (Min Heap)

Contoh penggunaan heap adalah pada persoalan yang mempertahankan antrian prioritas (priority queue). Dalam antrian prioritas, elemen yang dihapus adalah elemen yang mempunyai prioritas terbesar (atau terkecil, tergantung keperluan), dan elemen inilah yang **selalu** terletak di akar (root). Suatu heap dapat sewaktu-waktu berubah baik itu penambahan elemen (insert) dan penghapusan elemen (delete).

Ada beberapa operasi yang dapat terjadi di sebuah heap, yaitu :

1. Reorganisasi Heap (mengatur ulang heap).
2. Membantuk Heap (mengatur binary tree agar menjadi heap)
3. Penyisipan Heap (menyisipkan node baru)
4. Penghapusan Heap (menghapus node root)
5. Pengurutan Heap (Heap sort)

1. Reorganisasi Heap

Algoritma heap semuanya bekerja dengan prinsip bahwa modifikasi nilai prioritas pada suatu simpul dapat melanggar kondisi heap. Bila kondisi heap dilanggar, maka heap harus diorganisasi kembali.

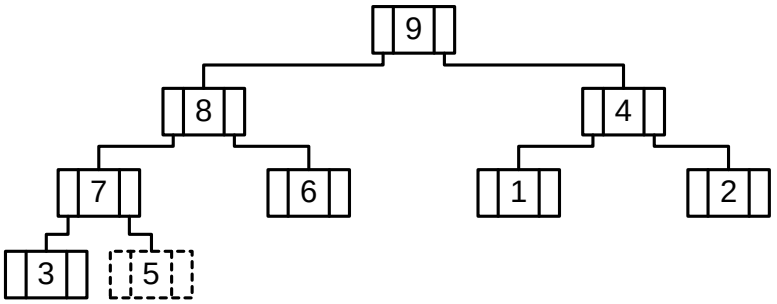
Sebagai contoh kita gunakan pada heap maksimum. Ketika nilai/prioritas suatu node merubah, maka ada 2 kemungkinan yang terjadi yaitu :

- Nilai prioritas node bertambah sehingga nilai prioritasnya lebih besar dari parentnya, maka lakukan langkah berikut :
  a. Tukarkan nilai prioritas node tersebut dengan nilai prioritas parent-nya.
  b. Ulangi langkah a, sampai ditemukan posisi yang tepat (memenuhi kondisi heap)

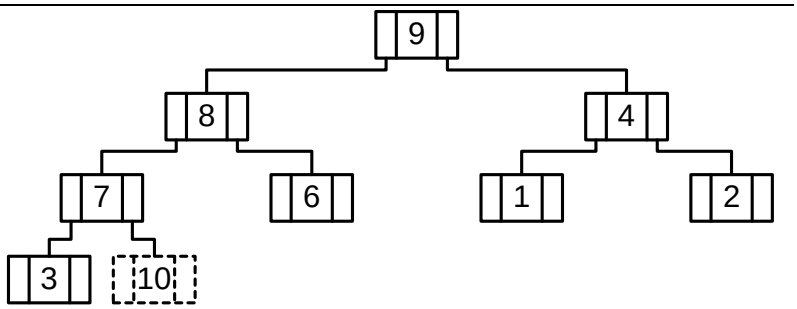
  Proses ini disebut dengan proses ***sift-up***.

  Contoh node dengan prioritas 5 menjadi 10.

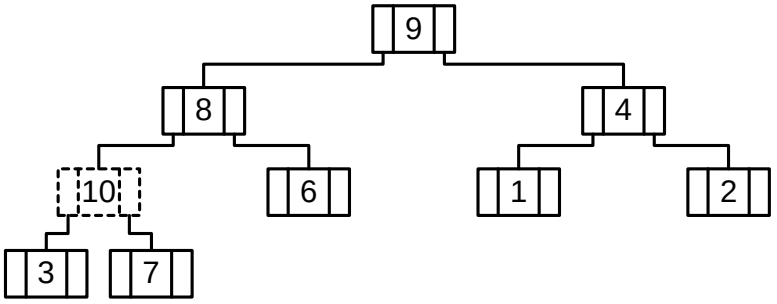
Kondisi Awal



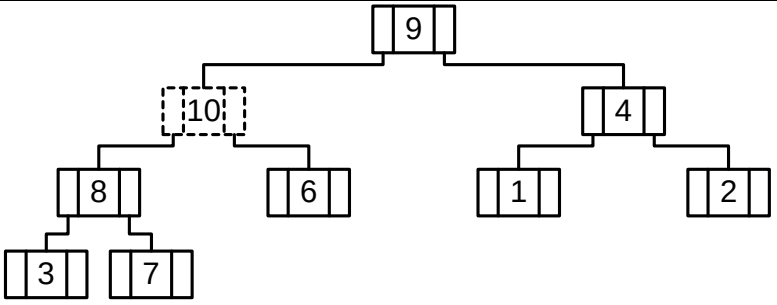
Node dengan prioritas 5 menjadi 10.



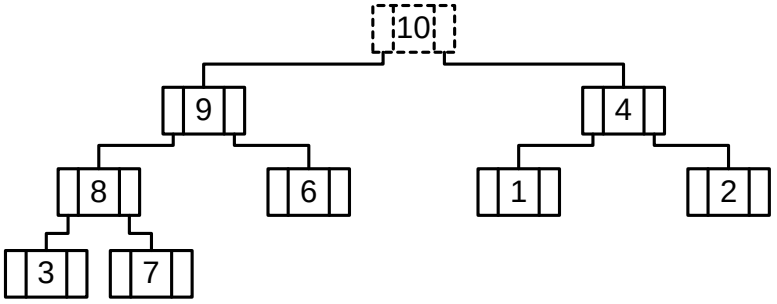
Karena prioritas node lebih besar dari prioritas parent, maka tukarkan prioritasnya.



Karena prioritas node lebih besar dari prioritas parent, maka tukarkan prioritasnya.



Karena prioritas node lebih besar dari prioritas parent, maka tukarkan prioritasnya.

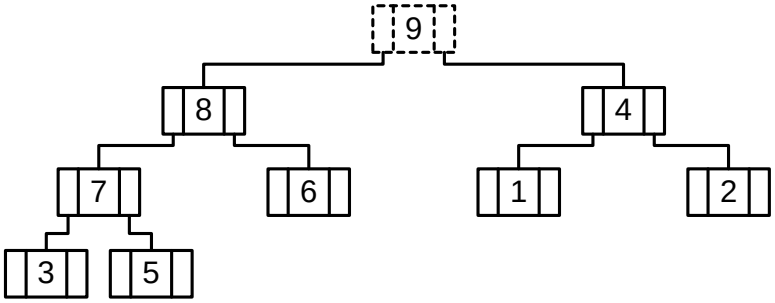


- Nilai prioritas node berkurang sehingga menjadi lebih kecil dari prioritas di antara node child-nya, maka yang harus dilakukan adalah :
  a. Tukarkan nilai prioritas simpul yang berubah dengan nilai prioritas dari child yang mempunyai prioritas terbesar.
  b. Ulangi langkah a, sampai ditemukan posisi yang tepat (memenuhi kondisi heap)

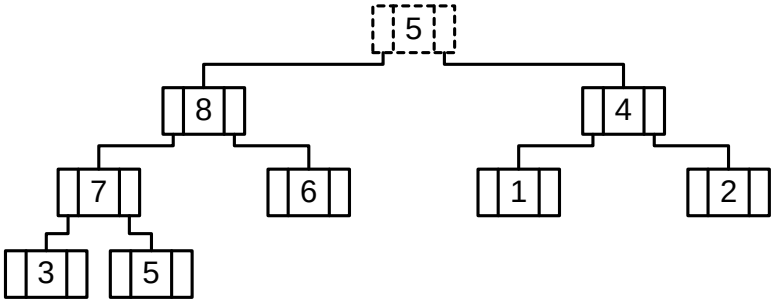
  Proses ini disebut dengan proses ***sift-down***.

  Contoh : Node dengan prioritas 9 menjadi 5.

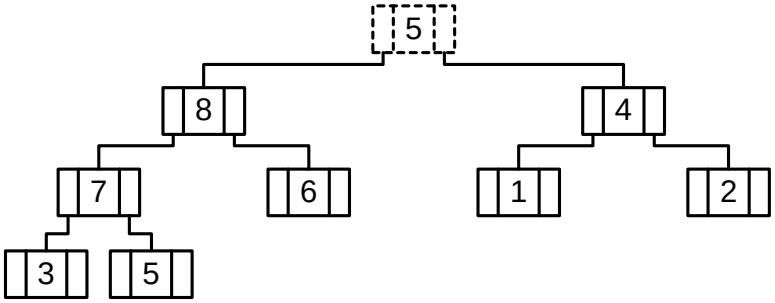
Kondisi Awal



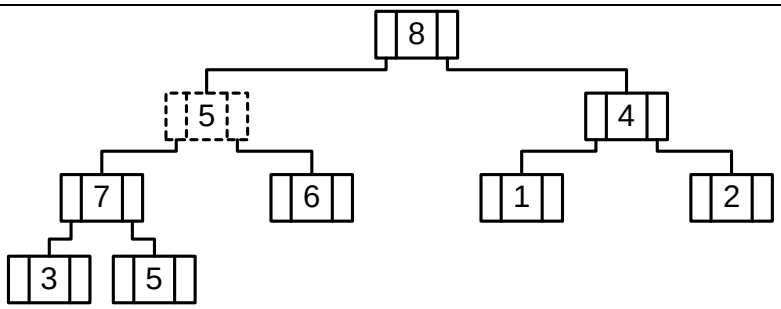
Node dengan prioritas 9 menjadi 5



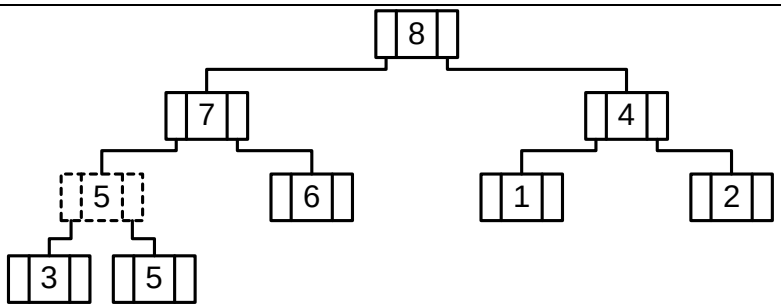
Prioritas node lebih kecil dari prioritas salah satu child, sehingga tukarkan.



Prioritas node lebih kecil dari prioritas salah satu child, sehingga tukarkan.



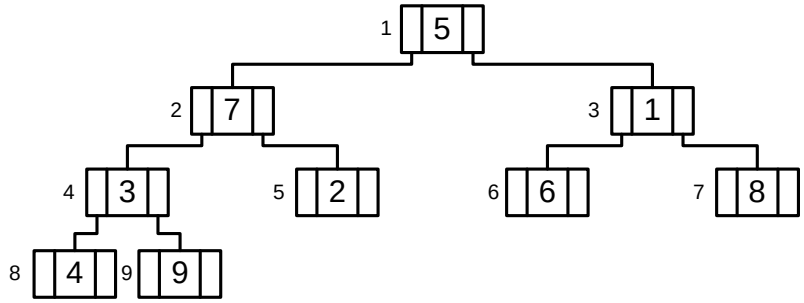
Prioritas node lebih kecil dari prioritas salah satu child, sehingga tukarkan dengan child terbesar.



2. Pembentukan Heap

Pada mulanya jika suatu complete binary tree memiliki prioritas antrian secara acak, maka langkah yang harus dilakukan agar binary tree tersebut dapat disebut sebagai heap adalah dengan melakukan proses sift_down dari node bernomor tengah (banyaknode/2 atau N/2), menurun sampai node pertama.

Kondisi Awal



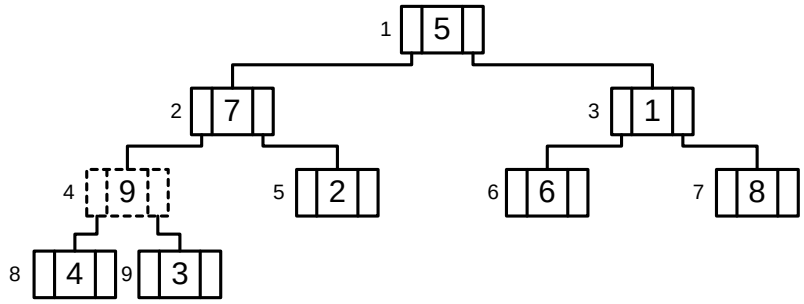
Representasi Array :

5	7	1	3	2	6	8	4	9
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

N=9, Tengah = N/2 = 9 / 2 = 4

Lakukan reorganisasi heap pada node 4 sampai node 1.

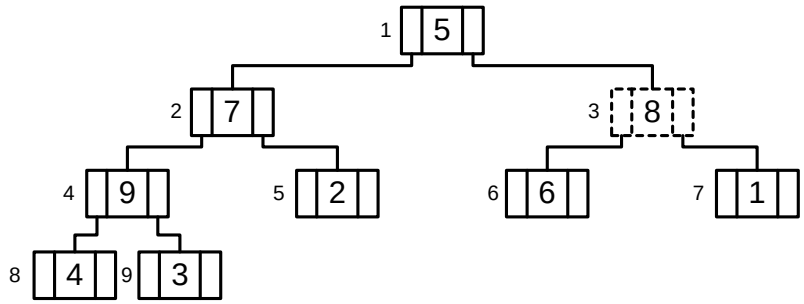
Reorganisasi pada node 4 sehingga tree menjadi



Representasi Array :

5	7	1	9	2	6	8	4	3
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

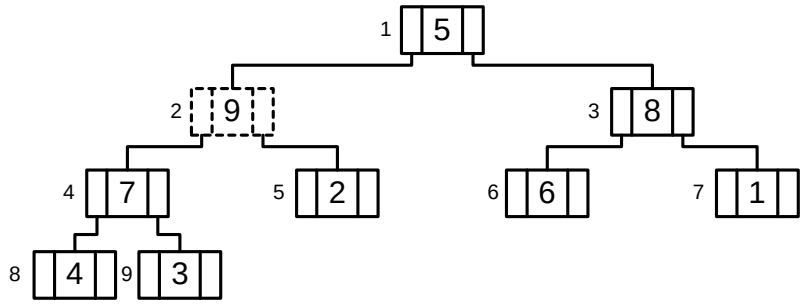
Reorganisasi pada node 3 sehingga tree menjadi



Representasi Array :

5	7	8	9	2	6	1	4	3
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

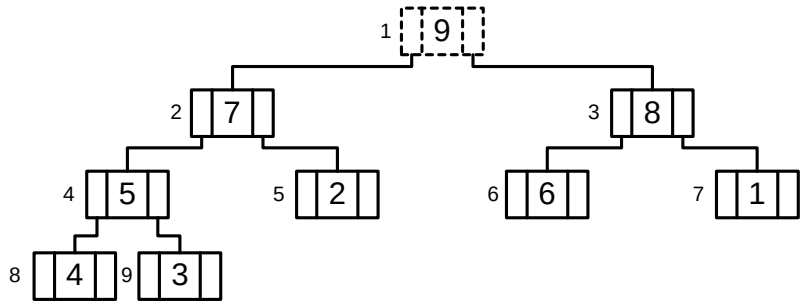
Reorganisasi pada node 2 sehingga tree menjadi



Representasi Array :

5	9	8	7	2	6	1	4	3
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Reorganisasi pada node 1, karena sudah sesuai dengan aturan heap maka node 1 tidak berubah.



Representasi Array :

9	7	8	7	2	6	1	4	3
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

3. Penyisipan Heap (Insert)

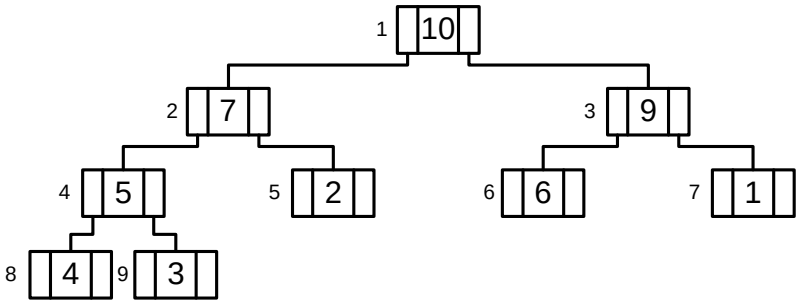
Penyisipan heap dilakukan ketika ada sebuah elemen baru diinsertkan. Algoritma untuk penyisipan data adalah :

- Simpan elemen baru tersebut setelah data paling akhir (tree dengan level paling bawah dan pada posisi sebelah kanan dari data terakhir atau jika level telah penuh, maka simpan data baru tersebut dalam level baru).
- Lakukan reorganisasi heap pada node baru tersebut. Proses yang biasanya dipakai adalah proses sift up.
- Banyak simpul ditambah 1

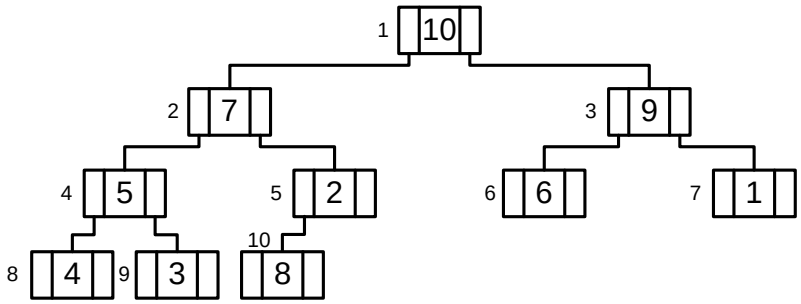
Contoh : Penyisipan Heap dengan prioritas/nilai 8

Kondisi awal :

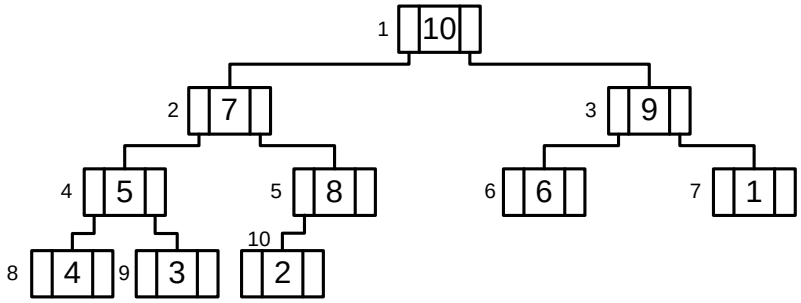
Banyak Node (N) : 9



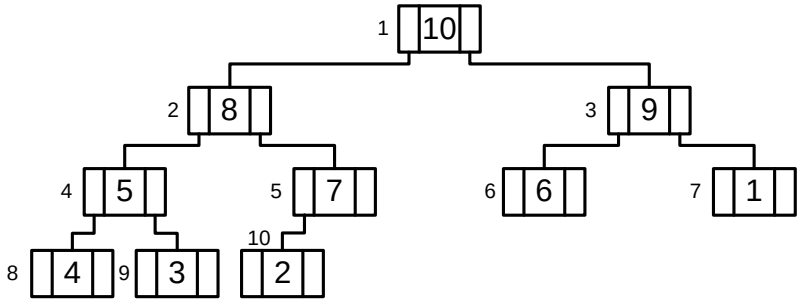
Sisipkan di posisi setelah data terakhir, dan banyak node (N) ditambah dengan 1 menjadi 10



Reorganisasi pada node baru (10)



Reorganisasi pada node 5 (karena ada perubahan prioritas), tukarkan dengan node 7 (node 2).



Karena posisi node 2 telah tepat, maka tidak ada reorganisasi sehingga proses penyisipan pun selesai.

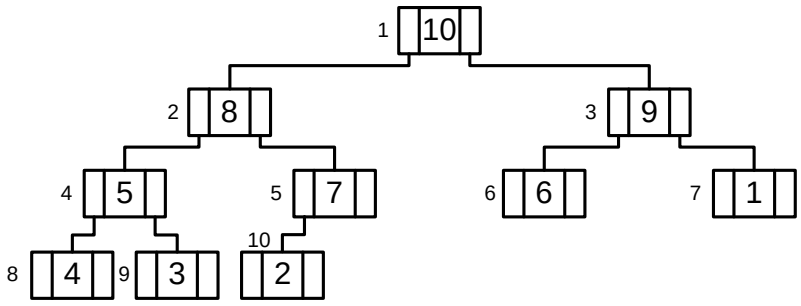
4. Penghapusan Heap (Delete)

Proses penghapusan dilakukan ketika root suatu tree diambil. Algoritma penghapusan heap adalah :

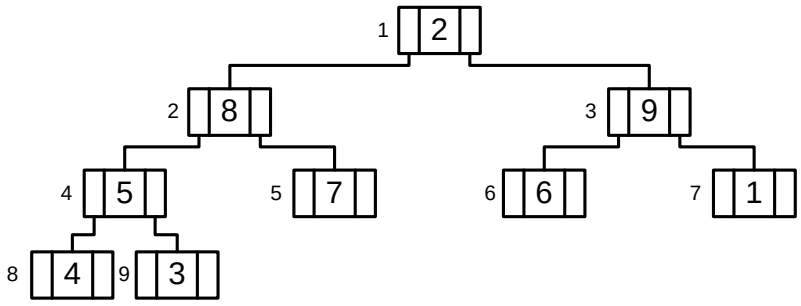
- Ambil Nilai Heap
- Ambil nilai prioritas pada node terakhir, dan dipakai sebagai prioritas root.
- Lakukan proses reorganisasi heap pada root. Umumnya proses yang dilakukan adalah proses sift down.
- Banyak simpul dikurang 1

Contoh : Hapus elemen heap. Elemen yang diambil adalah 10 (root)

N : 10

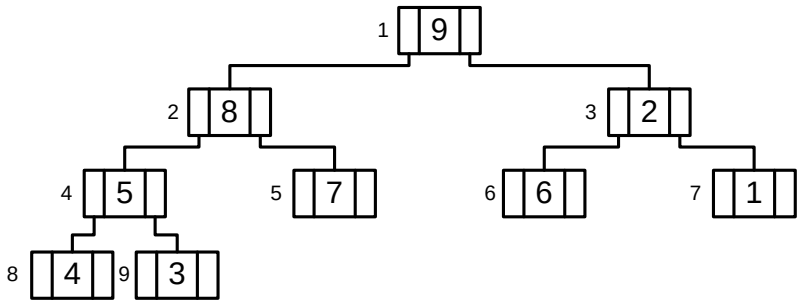


Ambil nilai Root, kemudian ganti dengan prioritas simpul terakhir (simpul 10 dengan prioritas 2). Sehingga tree menjadi :

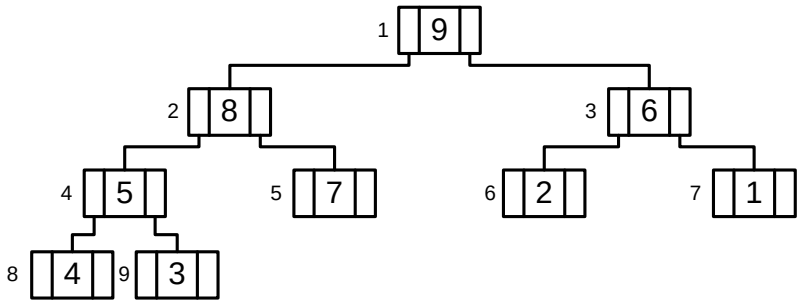


dan Banyak Simpul (N) dikurangi 1 menjadi 9.

Kemudian lakukan reorganisasi heap dengan proses sift down pada posisi 1 (root) sehingga tree menjadi :



Karena terjadi pertukaran dengan node 3, maka node 3 direorganisasikan kembali sehingga tree menjadi :



Karena posisi heap telah benar, maka proses penyisipan seleasi.

5. Pengurutan Heap (Heap Sort)

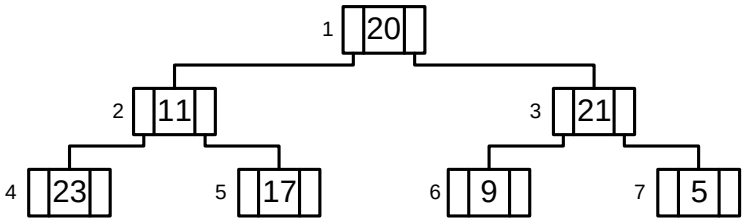
Pengurutan heap dapat dilakukan dengan algoritma seperti di bawah ini :

a. Buat Heap Maksimum
b. Jika N lebih besar dari 1 maka tukarkan Nilai/Prioritas root dengan prioritas simpul terakhir (simpul ke-N) tetapi jika N sama dengan 1 maka ambil nilai yang ada di root.
c. Kemudian nilai banyak simpul (N) dikurangi 1.
d. Jika N > 1 maka lakukan reorganisasi heap yaitu proses sift down terhadap root.
e. Lakukan langkah b sampai d sampai simpul habis (N=0).

Contoh :

Data yang akan diurutkan adalah : 20 11 21 23 17 9 5

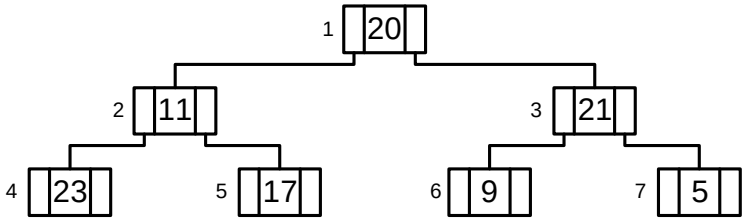
- Dari data di atas, dapat disusun tree seperti di bawah ini :



- Langkah pertama adalah mengkonversi tree di atas menjadi sebuah **heap**.

Kondisi Awal

N : 7

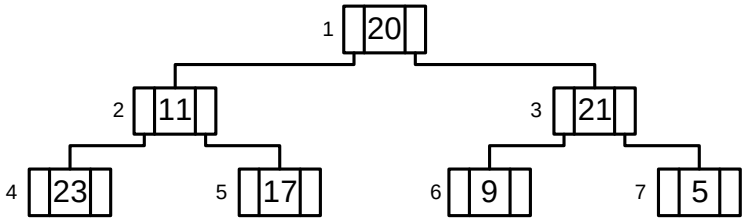


Representasi Array :

20	11	21	23	17	9	5
[1]	[2]	[3]	[4]	[5]	[6]	[7]

Lakukan reorganisasi heap untuk node N/2 (7/2) yaitu 3 sampai node 1.

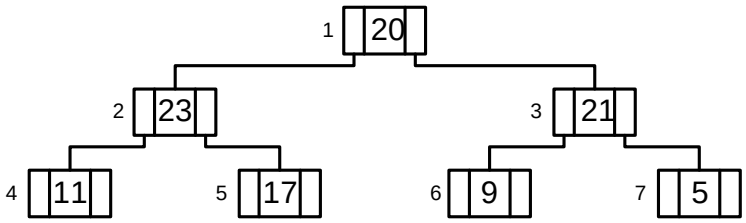
Node 3 Child telah sesuai sehingga tidak ada sift down.



Representasi Array :

20	11	21	23	17	9	5
[1]	[2]	[3]	[4]	[5]	[6]	[7]

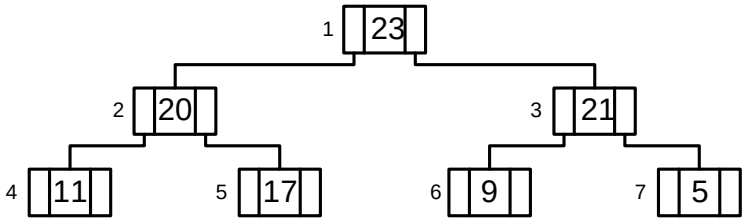
Node 2 Lakukan reorganisasi sift down pada node 2. Karena child dari node 2 ada yang memiliki prioritas lebih besar yaitu 23 dan 17. Tukarkan prioritas node 2 dengan child yang memiliki prioritas paling besar (node 4). Sehingga tree menjadi :



Representasi Array :

20	23	21	11	17	9	5
[1]	[2]	[3]	[4]	[5]	[6]	[7]

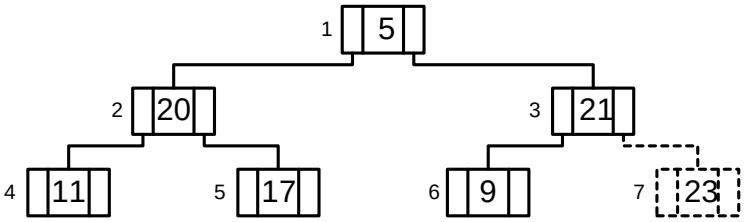
Node 1 Lakukan proses sift down karena nilai prioritas child lebih besar dari prioritas node 1. Tukarkan prioritas node 1 dengan prioritas node yang memiliki prioritas terbesar (node 2). Sehingga tree menjadi :



Representasi Array :

23	20	21	11	17	9	5
[1]	[2]	[3]	[4]	[5]	[6]	[7]

- Tukarkan nilai simpul root (1) dengan nilai simpul terakhir (N=7) sehingga tree menjadi seperti di bawah ini :

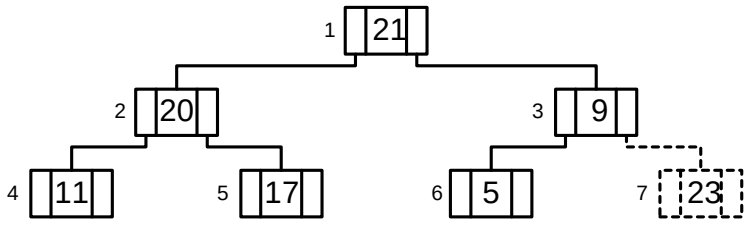


Representasi Array :

5	20	21	11	17	9	**23**
[1]	[2]	[3]	[4]	[5]	[6]	[7]

Banyak elemen (N) dikurangi 1 menjadi 7-1 = 6 Elemen. Simpul dengan nomor di atas N diabaikan karena sudah tidak termasuk tree lagi.

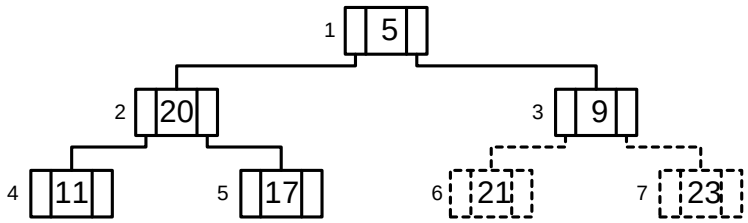
- Reorganisasi sift down terhadap node root (1) sehingga tree menjadi :



Representasi Array :

21	20	9	11	17	5	**23**
[1]	[2]	[3]	[4]	[5]	[6]	[7]

- Tukarkan kembali nilai node root (1) dengan data terakhir (N=6) sehingga tree menjadi :

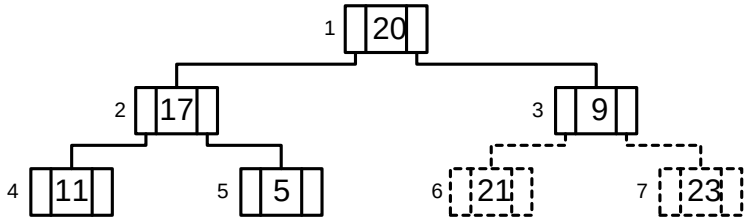


Representasi Array :

5	20	9	11	17	**21**	**23**
[1]	[2]	[3]	[4]	[5]	[6]	[7]

Kemudian banyak simpul (N) dikurangi 1 menjadi (6 – 1) = 5.

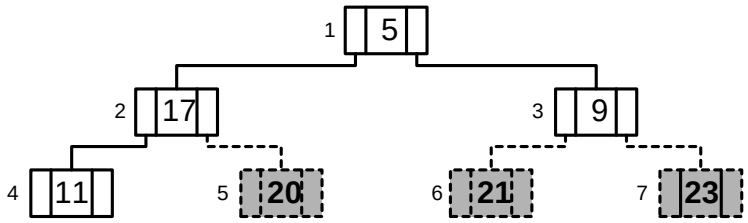
- Reorganisasikan sift down terhadap node root (1) sehingga tree menjadi :



Representasi Array :

20	17	9	11	5	**21**	**23**
[1]	[2]	[3]	[4]	[5]	[6]	[7]

- Tukarkan nilai/prioritas node root dengan node terakhir (N=5) maka tree akan terbentuk seperti di bawah ini :

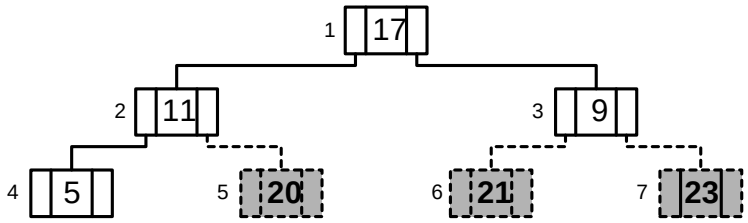


Representasi Array :

5	17	9	11	**20**	**21**	**23**
[1]	[2]	[3]	[4]	[5]	[6]	[7]

Kemudian banyak simpul (N) dikurangi 1 menjadi (5 – 1) = 4.

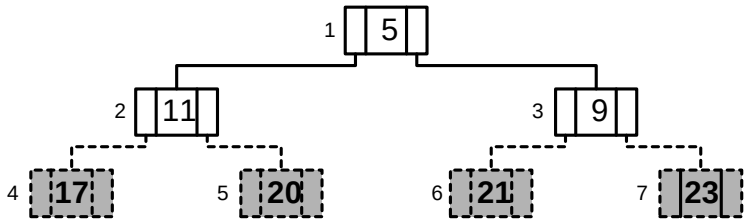
- Reorganisasikan secara sift down terhadap node root sehingga menjadi :



Representasi Array :

17	11	9	5	**20**	**21**	**23**
[1]	[2]	[3]	[4]	[5]	[6]	[7]

- Tukarkan kembali nilai node root (1) dengan node terakhir (N=4) sehingga tree menjadi :

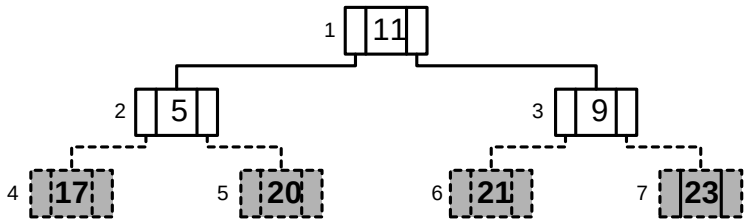


Representasi Array :

5	11	9	**17**	**20**	**21**	**23**

Kemudian banyak simpul (N) dikurangi 1 menjadi (4 – 1) =3.

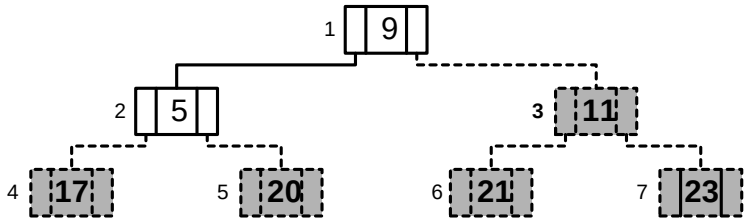
- Reorganisasi secara sift down kepada node root sehingga tree menjadi



Representasi Array :

11	5	9	**17**	**20**	**21**	**23**

- Tukarkan nilai node root dengan nilai node ke-N sehingga tree menjadi :



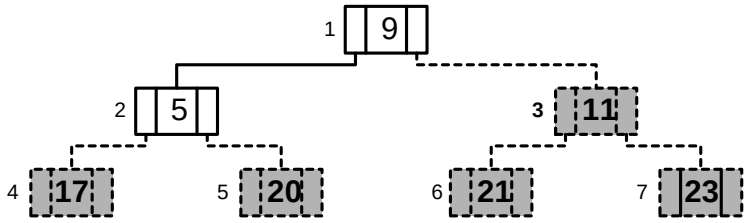
Representasi Array :

9	5	**11**	**17**	**20**	**21**	**23**

Kemudian N dikurangi 1 menjadi (3 – 1) = 2

- Reorganisasi secara sift down terhadap node root sehingga tree menjadi sebagai berikut :

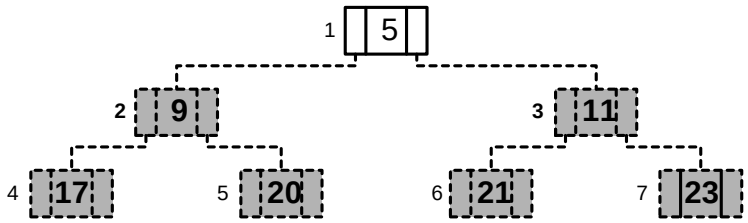
  Karena node root telah sesuai ketentuan heap maka proses reorganisasi tidak dilakukan sehingga tree tetap.



Representasi Array :

9	5	**11**	**17**	**20**	**21**	**23**

- Tukarkan nilai root dengan nilai pada node ke-N (N=2). Sehingga tree menjadi :

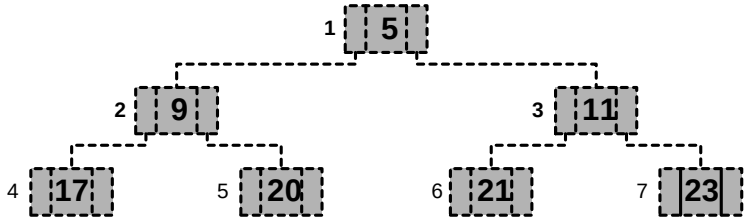


Representasi Array :

5	**9**	**11**	**17**	**20**	**21**	**23**

Kemudian nilai N dikurangi 1 sehingga menjadi (2 -1) = 1.

- Karena N sama dengan 1 maka tidak perlu ada reorganisasi
- Karena N sama dengan 1 maka ambil data yang ada di root, sehingga tree menjadi kosong, seperti di bawah ini :



Representasi Array :

5	**9**	**11**	**17**	**20**	**21**	**23**

Kemudian nilai N dikurangi 1 sehingga menjadi (1 -1) = 0.

- Karena N telah mencapai 0 maka pengurutan selesai.