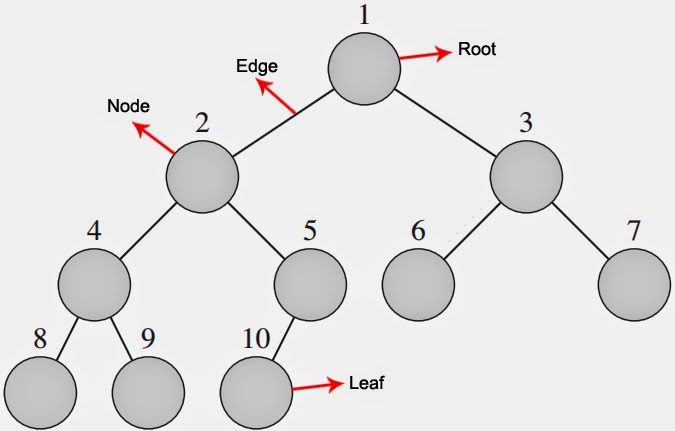
Algoritma Heap Sort

Heap sort adalah sebuah metode sorting (pengurutan) angka pada sebuah array dengan cara menyerupai binary tree, yaitu dengan cara memvisualisasikan sebuah array menjadi sebuah binary tree yang nantinya pada binary tree tersebut nilai pada masing-masing index array akan diurutkan. Pada heap sort terdapat 3 bagian yaitu ***Node***, ***Edge***, dan ***leaf*** dimana node itu adalah setiap index yang berada pada array, edge adalah garis yang menghubungkan tiap node dan leaf adalah setiap node yang tidak memiliki child node (node turunan). Selain itu juga ada yang bernama ***root*** yaitu node awal pada sebuah heap, berikut adalah ilustrasi dari bagian yang dimiliki oleh heap :

[](http://3.bp.blogspot.com/-uu41YU23hqE/U9mtY9nXIjI/AAAAAAAAAH0/x2COR5luFEU/s1600/Heap+Components.jpg)

Heap tree terbagi menjadi 2 jenis yaitu ***Max-Heap*** dan ***Min-Heap***, dimana max-heap adalah kondisi heap tree yang memiliki nilai tertinggi berada di node root dan setiap child node memiliki nilai yang lebih kecil dari nilai yang dimiliki parent nodenya. Sedangkan pada min-heap adalah kondisi kebalikan dengan max-heap, pada min-heap nilai terkecil berada di node root dan setiap child node memiliki nilai yang lebih besar dari nilai yang dimiliki parent nodenya. Pada metode heap sort jenis heap tree yang digunakan adalah ***Max-Heap***.

Dan untuk memvisualisasikan sebuah array menjadi sebuah heap tree adalah dengan cara mencari node root terlebih dahulu yaitu node pertama node pertama sebuah heap tree adalah index pertama di array yaitu index 0 akan tetapi pada heap tree node awal berada di posisi 1 berbeda dengan array yang memiliki index awal yaitu index 0. Setelah node root telah ditemukan maka sekarang tinggal mencari child node dari node root dan child node terbagi menjadi 2 yaitu left child dan right child dan untuk mencari left child, right child, dan parent digunakan rumus sebagai berikut :

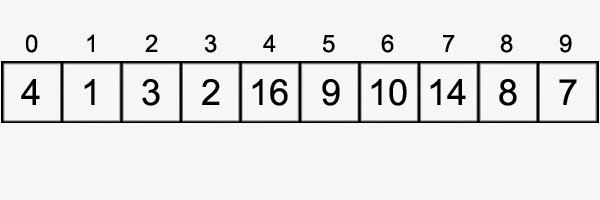
         **Left Child**            : 2i (Contoh : Left child dari 1 adalah 2 x 1 = 2)

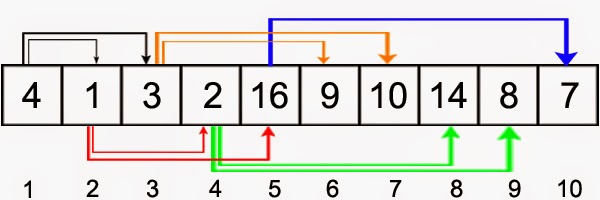
         **Right Child**                     : 2i + 1 (Contoh : Right Child dari 1 adalah (2 x 1) + 1 = 3)

         **Parent**                              : └ i/2 ┘ (Contoh : Parent dari 3 adalah 3 / 2 = 1 )

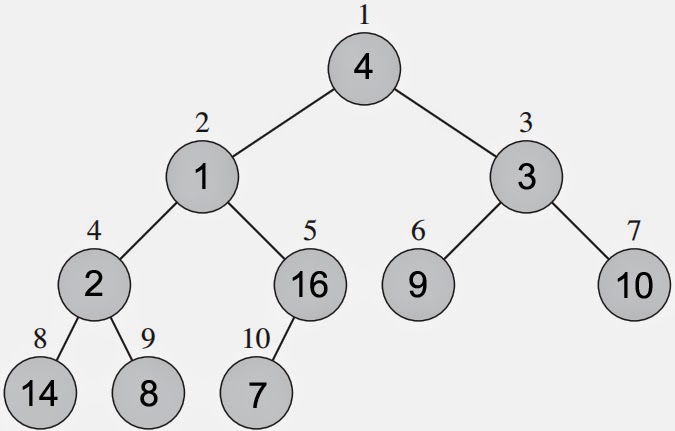
**NB** : Untuk ***i*** adalah posisi node yang ingin dicari left/right childnya atau parent nodenya dan untuk lambing (**└ ┘**) adalah ***floor*** yaitu pembulatan kebawah missal 3 / 2 = 1,5 dibulatkan kebawah menjadi 1. Berikut adalah contoh cara memvisualisasikan sebuh array menjadi sebuah heap tree :

Contoh : Kita memiliki sebuah aray A = 4, 1, 3, 2, 16, 9, 10, 14, 8, 7. Dan untuk memvisualisasikan array tersebut gunakan rumus yang sudah disediakan dan prosesnya akan terlihat seperti ini :

[](http://3.bp.blogspot.com/-DApYQsNORXU/U9mty697qnI/AAAAAAAAAH8/LcOKh8drDEM/s1600/Array+A.jpg)

[](http://2.bp.blogspot.com/-2kkbn3V5nIY/U9muA-SsnZI/AAAAAAAAAIE/hUJhNGGrrSY/s1600/Array+to+Heap.jpg)

Dan hasil heap treenya adalah sebagai berikut :

[](http://2.bp.blogspot.com/-2fYDp1RuNco/U9muXHTRbDI/AAAAAAAAAIM/q0QB7l8SDSk/s1600/HeapSort+1.jpg)

Akan tetapi pada max-heap kondisi heap tree adalah node dengan nilai tertinggi adalah root dan setiap parent node memiliki nilai yang lebih besar dari child nodenya, dan heap tree yang terbentuk dari array A tidak memenuhi kondisi max-heap oleh karena itu dibutuhkan metode untuk membuat heap tree tersebut memiliki kondisi max-heap. Dalam metode sorting heap sort terdapat 2 metode yang digunakan yaitu ***Build-Max-Heap*** dan ***Max-Heapfy***, Build-Max-Heap adalah metode yang digunakan untuk membuat heap tree diatas memenuhi kondisi dari max-heap, berikut ini adalah ilustrasi penggunaan metode ***HeapSort***, ***Build-Max-Heap***, dan ***Max-Heapfy*** pada sebuah heap tree :

**HeapSort(A)**

**1.**    **Deklarasi array A**

**2.**    **Deklarasi Elemen**

**3.**    **Input elemen array A**

**4.**    **Input nilai-nilai elemen array A**

**5.**    **Build-Max-Heap(A)**

**6.**    **For i = Elemen – 1 selama i > 0**

**7.**    **Tukar A[i] dengan A[0]**

**8.**    **Elemen – 1**

**9.**    **Max-Heapfy(A, 1)**

**10.** **End for**

Dapat dilihat pada algoritma HeapSort terdapat 2 metode yang dipanggil yaitu Build-Max-heap dan Max-Heapfy, dan algoritma dari Build-Max-heap adalah :

**Build-Max-Heap(A)**

**1.**    **For i = (Elemen - 1) / 2 selama i ≥ 0**

**2.**    **Max-Heapfy(A, i)**

**3.**    **End for**

Pada metode Build-Max-Heap terdapat for looping yang membagi 2 jumlah elemen, disini elemen – 1 karena pada array index awal adalah 0 sedangkan pada heap tree adalah 1, lalu elemen dibagi 2 dan selama i ≥ 0 maka for looping akan terus berajalan. Dan berikut adalah metode Max-Heapfy :

**Max-Heapfy(A, i)**

**1.**    **Deklarasi left = (i + 1) \* 2 – 1**

**2.**    **Deklarasi right = (i + 1) \* 2**

**3.**    **Deklarasi largest**

**4.**    **if(left < elemen dan A[left] > A[i])**

**5.**    **largest = left**

**6.**    **end if**

**7.**    **else**

**8.**    **largest = i**

**9.**    **end else**

**10.** **if(right < elemen dan A[right] > A[i])**

**11.** **largest = right**

**12.** **end if**

**13.** **if(largest != i)**

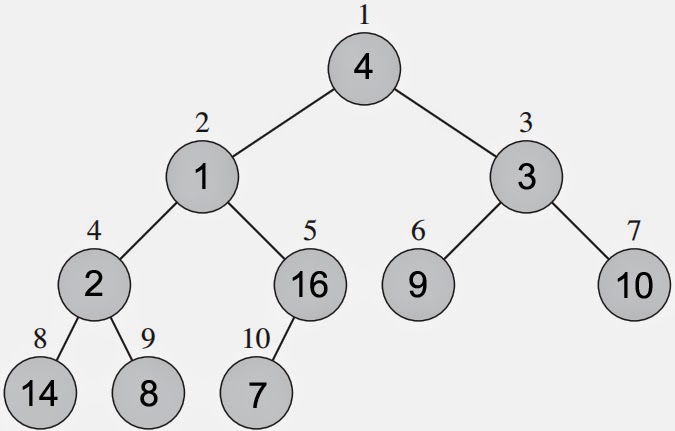
**14.** **Tukar A[i] dengan A[largest]**

**15.** **Max-Heapfy(A, i)**

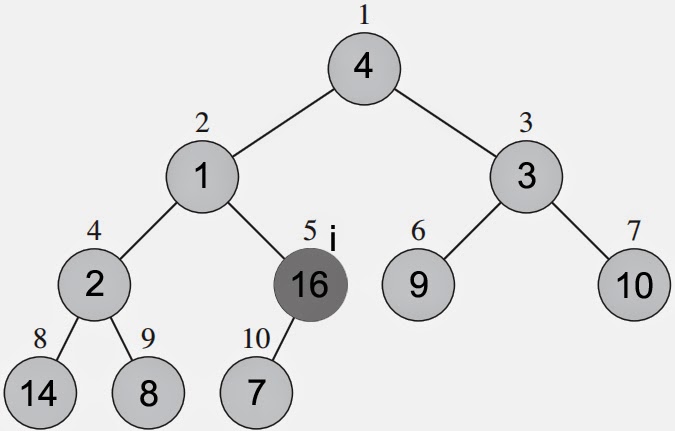
**16.** **end if**

Sebenarnya metode Max-Heapfy digunakan untuk mengkoreksi posisi dari index yang dipilih apakah posisinya sudah memenuhi kondisi Max-Heap yaitu tiap node parent harus memiliki nilai yang lebih tinggi dari nilai yang dimiliki child nodenya, dan dengan metode Max-Heapfy pengaturan posisi node yang memenuhi kondisi Max-Heap dapat dilakukan. Maka pada metode Build-Max-Heap banyaknya elemen array dibagi menjadi 2 dan hasil bagianya itu adalah lokasi index awal yang akan diperiksa apakah sudah memenuhi kondisi Max-Heap atau belum dan proses pemeriksaan dilakukan oleh metode Max-Heapfy. Berikut adalah ilustrasi penggunakan metode Build-Max-Heap :

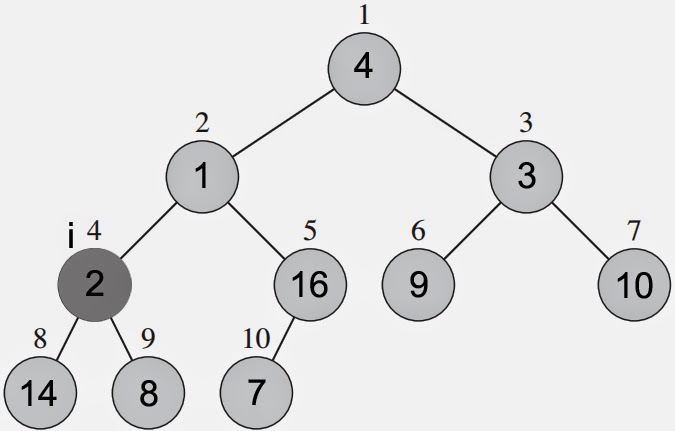
Heap tree awal adalah seperti ini :

[](http://2.bp.blogspot.com/-2fYDp1RuNco/U9muXHTRbDI/AAAAAAAAAIM/q0QB7l8SDSk/s1600/HeapSort+1.jpg)

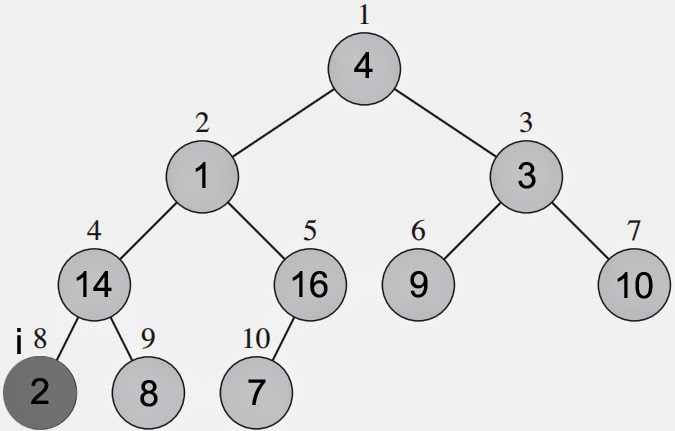
Karena pada heap tree jumlah node/elemen ada 10 (kalo di array adalah 9) maka jumlah elemen di bagi 2 (10 / 2 = 5 atau └ 9 / 2 ┘ = 4) maka yang menjadi variabel i adalah 5 (untuk heap tree) atau 4 (untuk array) maka ilustrasinya adalah sebagai berikut :

[](http://4.bp.blogspot.com/-cJUTBE6p0ls/U9mvTk-Z9RI/AAAAAAAAAIY/0cCAo7I-EBE/s1600/Build+Max+Heap+1.jpg)

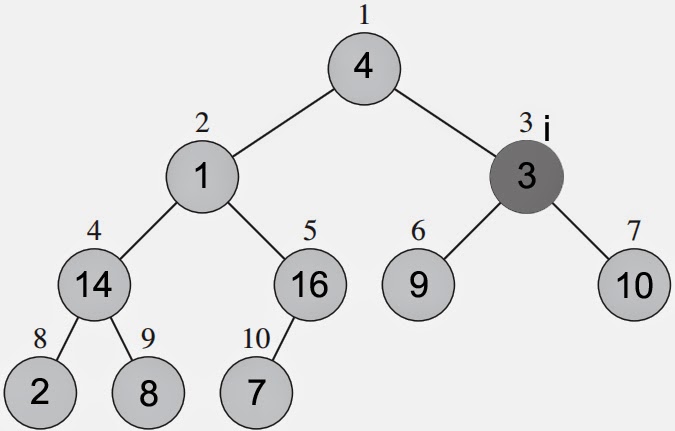
Karena node sebagai parent node sudah memiliki nilai lebih besar dari child nodenya (node 10) maka pertukaran posisi tidak terjadi, selanjutnya adalah lanjut ke index 4 :

[](http://1.bp.blogspot.com/-hYvbKZMCz3w/U9mviURFSwI/AAAAAAAAAIg/YAYAz25jNKA/s1600/Build+Max+Heap+2.jpg)

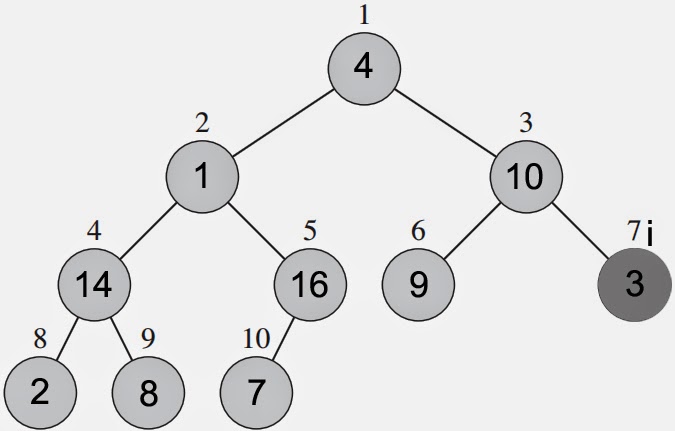
Disini ternyata nilai yang dimiliki node 4 lebih kecil dari nilai yang dimiliki child nodenya (node 8 dan 9)  maka pertukaran posisi dilakukan dan hasilnya adalah sebagai berikut :

[](http://2.bp.blogspot.com/-qDKS9jfuI5w/U9mvyQQ0t-I/AAAAAAAAAIo/NecQuS6yj9o/s1600/Build+Max+Heap+3.jpg)

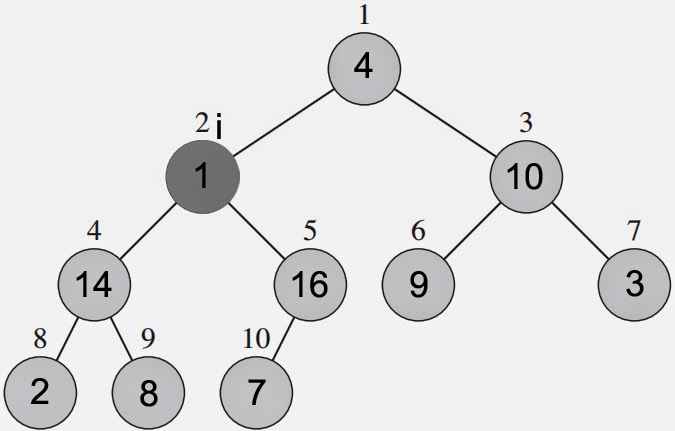
Selanjutnya adalah node 3 :

[](http://2.bp.blogspot.com/-DMdnwW8w80U/U9mwFwtLP9I/AAAAAAAAAIw/S-v4K8i6vH4/s1600/Build+Max+Heap+4.jpg)

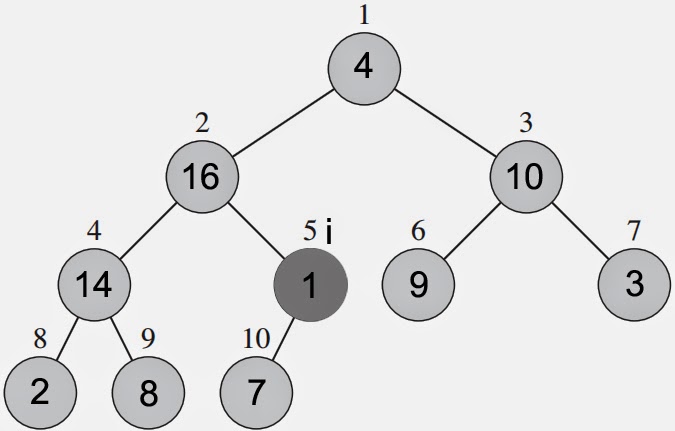
Disni node 3 memiliki nilai yang lebih kecil dari child nodenya (node 6 dan 7) maka pertukaran posisi dilakukan antara node 3 dan 7 karena antara node 3, 6, dan 7 node 7 lah yang memiliki nilai yang paling tinggi. Maka hasilnya adalah sebagai berikut :

[[](http://1.bp.blogspot.com/-giVgPc8ckxU/U9mwd4GhIgI/AAAAAAAAAI4/fqAGlQIuPkU/s1600/Build+Max+Heap+5.jpg)](http://1.bp.blogspot.com/-giVgPc8ckxU/U9mwd4GhIgI/AAAAAAAAAI4/fqAGlQIuPkU/s1600/Build+Max+Heap+5.jpg)

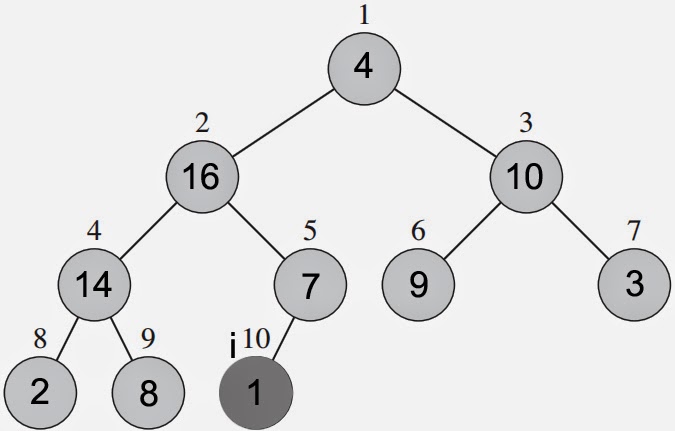
Selanjutnya adalah node 2 :

[](http://4.bp.blogspot.com/-58amQpC016E/U9mwxWURZRI/AAAAAAAAAJA/Z8nxZTcurO4/s1600/Build+Max+Heap+6.jpg)

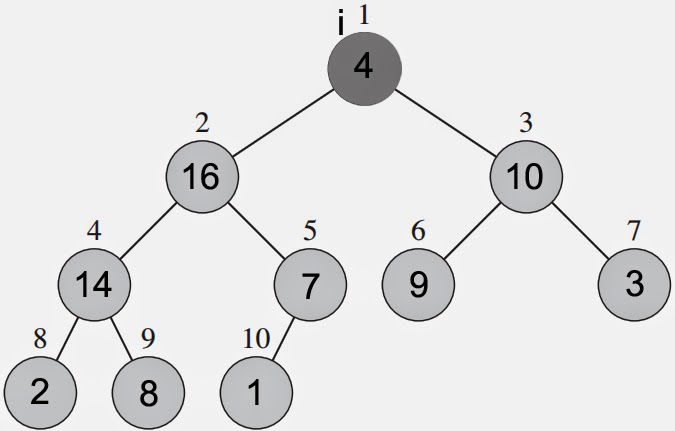
Node 2 memiliki nilai yang lebih kecil dari child nodenya (node 4 dan 5) maka pertukaran posisi dilakukan disini node 5 memiliki nilai yang paling tinggi maka nilai pada index 2 bertukar posisi dengan nilai pada index 5. Maka hasilnya adalah sebagai berikut :

[](http://4.bp.blogspot.com/-3_5t9LfSWhw/U9mw-p_-C4I/AAAAAAAAAJI/9fno4Drw-gs/s1600/Build+Max+Heap+7.jpg)

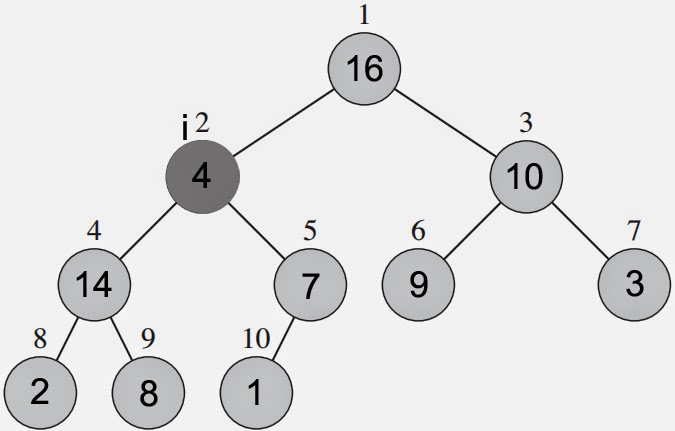
Setelah itu posisi index berubah menjadi pada node 5 dan ternyata disini nilai pada node 5 lebih kecil dari nilai yang berada di child nodenya yaitu node 10 maka perpindahan posisi kembali dilakukan dan hasilnya terlihat seperti ini :

[](http://3.bp.blogspot.com/-bAMd9fOL9A0/U9mxUHG12AI/AAAAAAAAAJQ/8uwf9HEQYC0/s1600/Build+Max+Heap+8.jpg)

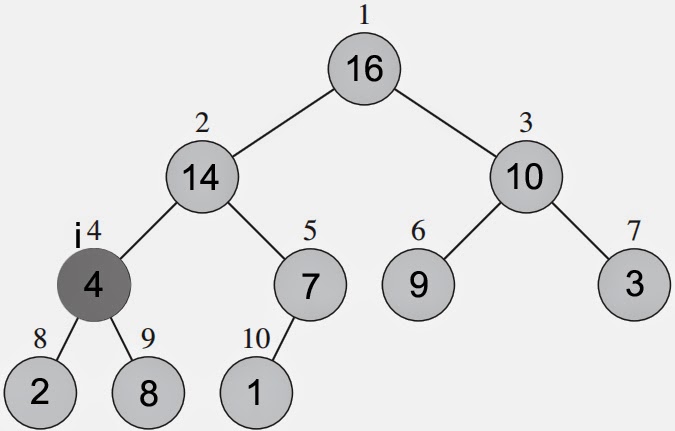
Selanjutnya posisi index berada di node 1 :

[](http://1.bp.blogspot.com/-5zUcnRWUMJ8/U9mxfcQy5aI/AAAAAAAAAJY/14916lgagA0/s1600/Build+Max+Heap+9.jpg)

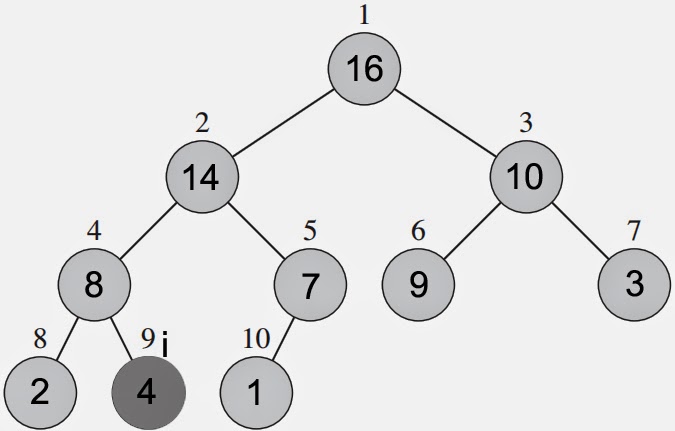
Disini nilai pada index 1 lebih kecil dbandingkan dengan nilai yang dimiliki child nodenya (node 2 dan 3) maka nilai pada node 1 ditukar dengan nilai pada node 2 karena nilai pada node 2 adalah nilai tertinggi dari nilai pada 1 dan 3 dan hasilnya adalah :

[](http://2.bp.blogspot.com/-Z6iAezwMqiw/U9mxrfIOj7I/AAAAAAAAAJg/MgtUPU-gLcE/s1600/Build+Max+Heap+10.jpg)

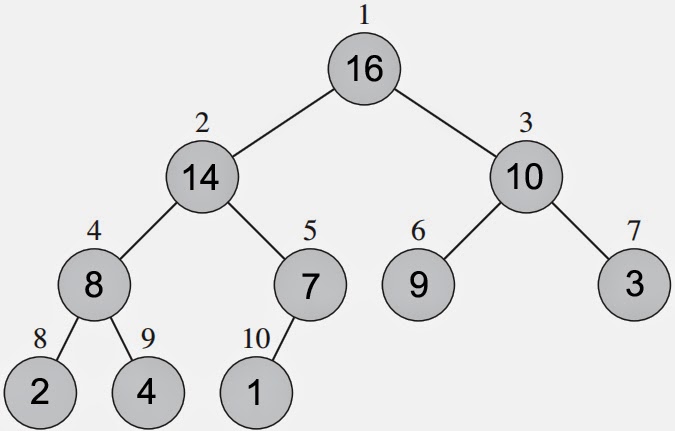
Setelah pertukaran posisi sekarang posisi ***i*** ada di index 2 disini nilai pada index 2 lebih kecil dibandingkan nila yang dimiliki child nodenya (index 4 dan 5) maka pertukaran posisi kembali terjadi disini index 4 memiliki nilai tertinggi maka nilai pada indx i ditukar dengan nilai pada index 4 dan hasilnya adalah sebagai berikut :

[](http://2.bp.blogspot.com/-Syv_kFRlStA/U9mx_2aAdpI/AAAAAAAAAJo/MVhirCSNgOQ/s1600/Build+Max+Heap+11.jpg)

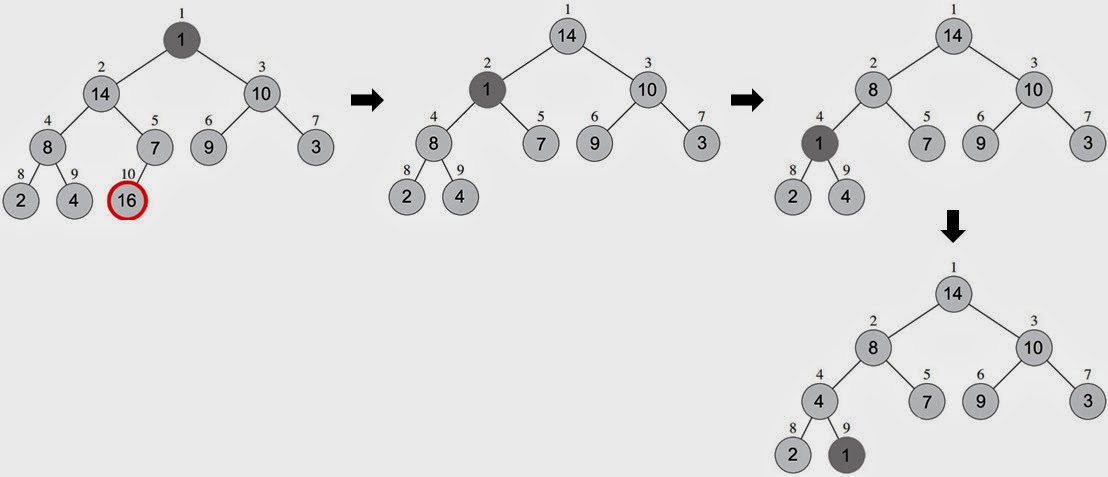
Setelah proses tukar posisi selesai maka posisi i adalah di node 4 akan tetapi disini kembali nilai pada index i lebih kecil dibandingkan dengan nilai yang dimiliki child nodenya (node 9), maka nilai pada index i ditukar dengan nilai pada index 9 dan hasilnya adalah sebagai berikut  :

[](http://1.bp.blogspot.com/-J91hwMgVgAQ/U9myNRVccuI/AAAAAAAAAJw/F2YycxpoIa0/s1600/Build+Max+Heap+12.jpg)

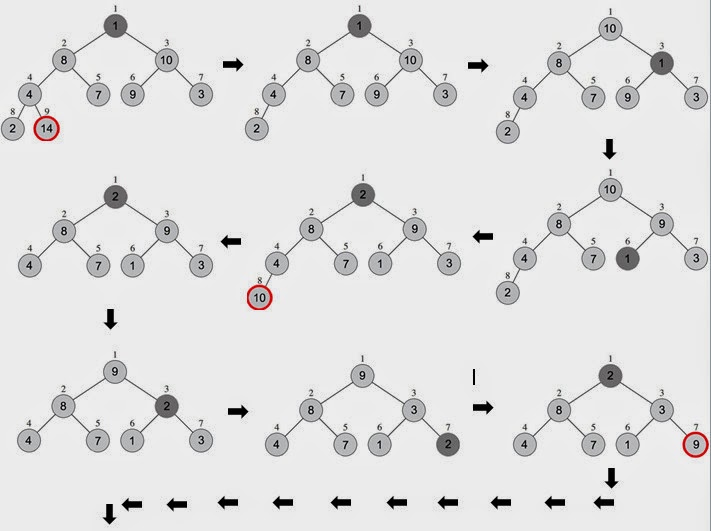
Sampai disini maka proes Build-Max-Heap telah selesai karena masing-masing parent node memiliki nilai yang lebih besar dari child nodenya, yaitu seperti heap tree berikut :

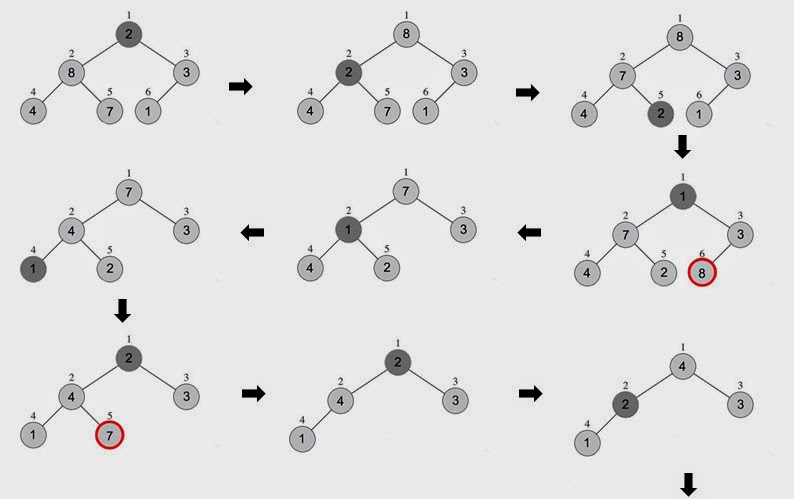
[](http://3.bp.blogspot.com/-ChQ7zjUGEYc/U9myb9YZ2QI/AAAAAAAAAJ4/l5h_I3TQkMM/s1600/Build+Max+Heap+Completed.jpg)

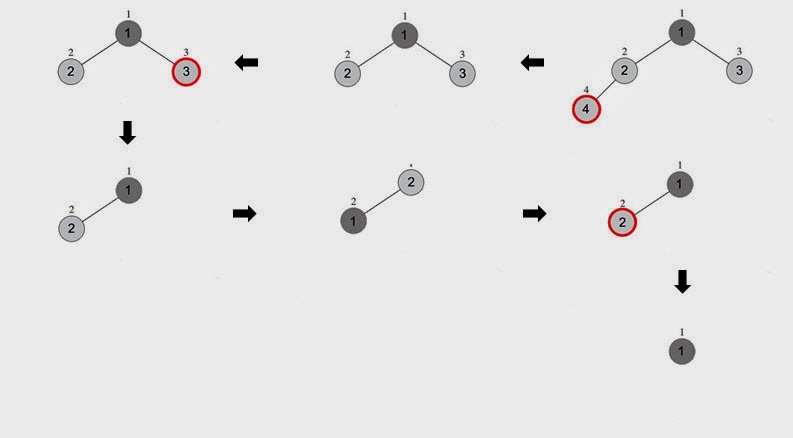
Setelah proses Build-Max-Heap telah selesai baru lah kita dapat menggunakan metode HeapSort untuk mengurutkan nilai pada array A. Pada algoritma heapsort setelah melakukan algoritma Build-Max-Heap nilai pada index terakhir i akan ditukar dengan node 1 atau root selama i > 0 disini nilai 16 akan ditukar dengan 1 dan jumlah elemen akan dikurangi 1 akan tetapi setelah perkukaran posisi dilakukan tree heap tidak memenuhi kondisi Max-Heap maka algoritma Max-Heapfy digunakan dan ilustrasinya adalah sebagai berikut :

[](http://4.bp.blogspot.com/-j1rIthGhCh0/U9my12N5W6I/AAAAAAAAAKA/xyu7YQrwA8g/s1600/Heap+Sort+Complete+01.jpg)

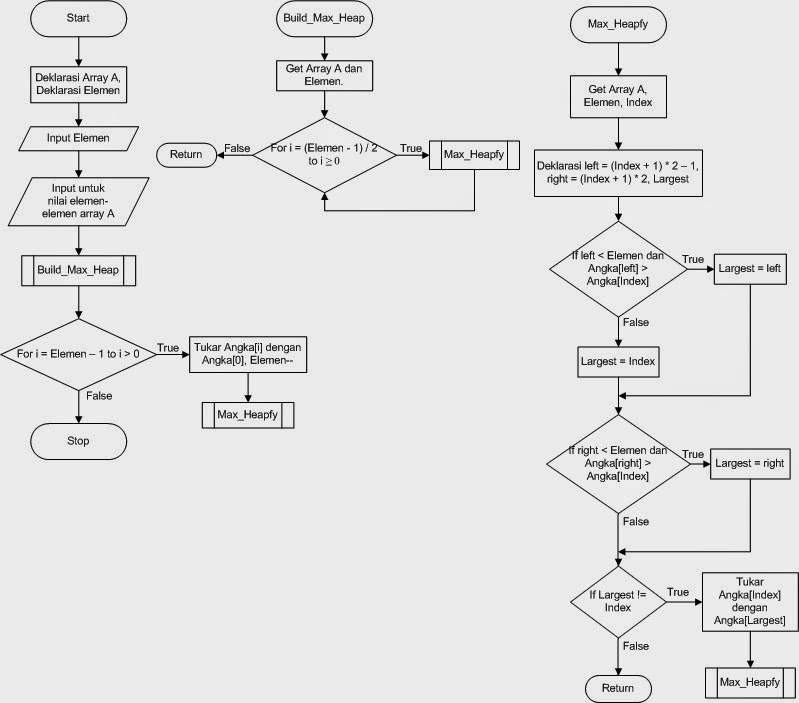
Sampai pada tahap ini nilai tertinggi sudah berada di index yang benar index terakhir 10 pada heap tree dan 9 di array, langkah selanjutnya adalah mengulang cara yang sama dengan for looping selama  i > 0. Berikut adalah ilustrasi lengkapnya :

[](http://1.bp.blogspot.com/-oX95obOwr1s/U9m3KAD0KtI/AAAAAAAAAKg/Gu25Gbuvqac/s1600/Heap+Sort+Complete+02_1.jpg)

[](http://1.bp.blogspot.com/-wKr0sw5M0uQ/U9mzol2LpMI/AAAAAAAAAKU/82H-7SqIC7c/s1600/Heap+Sort+Complete+03.jpg)



Maka setelah algoritma HeapSort dilakukan nilai-nilai pada array akan terurut dari nilai terkecil sampai terbesar. A = 1, 2, 3, 4, 7, 8, 9, 10, 14, 16. Dan berikut adalah flowchart untuk algoritma Heapsort :

[](http://1.bp.blogspot.com/-LMxuKijsiig/U9m5U7JcxWI/AAAAAAAAALA/4N1tiCIzMAA/s1600/Heap+Sort+Flowchart.jpg)