Nama : Afie Syahrulloh Arridlo

NPM : 140810170040

HEAPSORT

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.

Sourcecode C++ :

```cpp
// Nama : Afie Syahrulloh Arridlo
// NPM : 140810170040
#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
    int largest = i; // Initialize largest as root
    int l = 2*i + 1; // left = 2*i + 1
    int r = 2*i + 2; // right = 2*i + 2

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i)
    {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// main function to do heap sort
void heapSort(int arr[], int n)
{
```

```
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i=n-1; i>=0; i--)
    {
        // Move current root to end
        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

/* A utility function to print array of size n */
void printArray(int arr[], int n)
{
    for (int i=0; i<n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

// Driver program
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is \n";
    printArray(arr, n);
}
```

Hasil dari Program :

```
PS C:\Users\LENOVO\Desktop> & '.\heapshort.exe'
Sorted array is
5 6 7 11 12 13
```

Algoritma :

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

Build-Max-Heap(A)

1. For i = (Elemen - 1) / 2 selama i ≥ 0
2. Max-Heapfy(A, i)
3. End for

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)

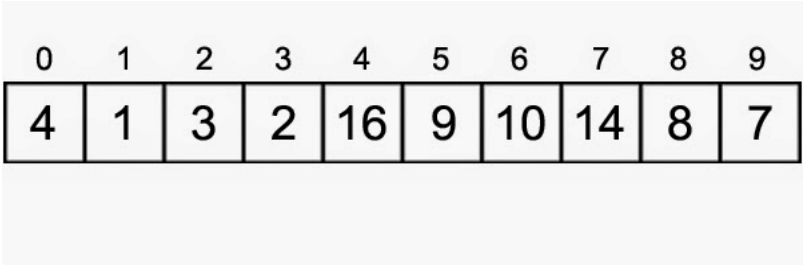
14. Tukar A[i] dengan A[largest]

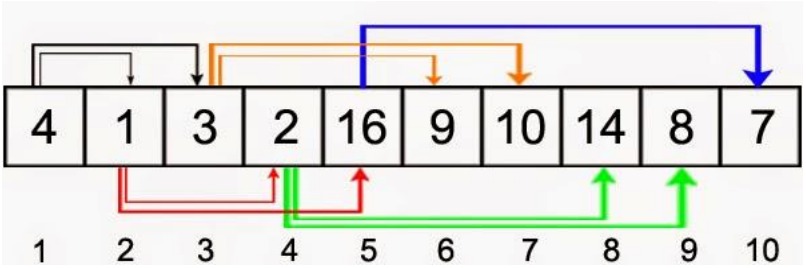
15. Max-Heapfy(A, i)

16. end if

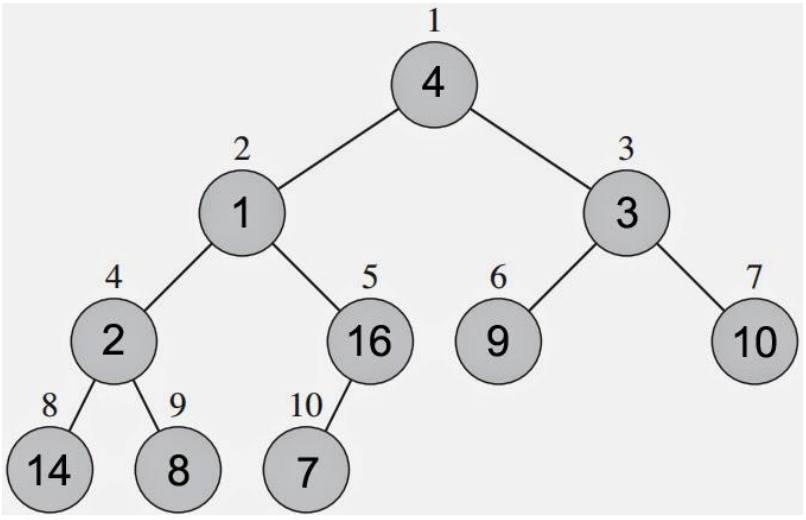
Contoh Soal dan Pembahasan :

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 :

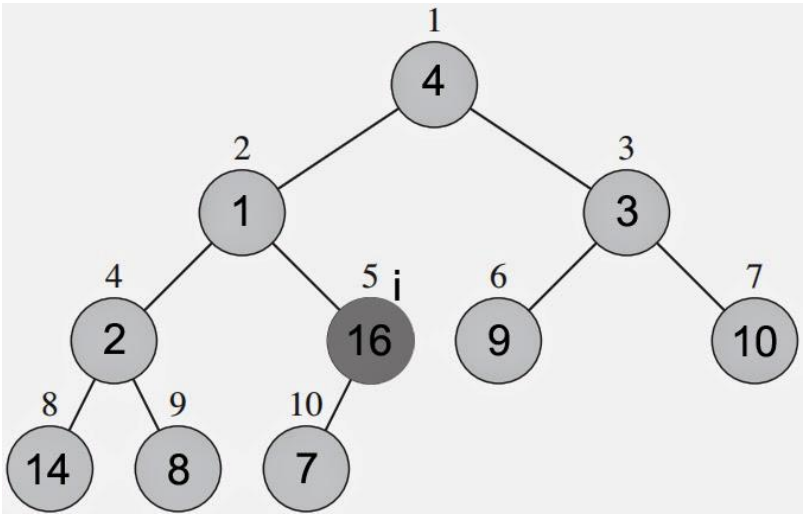




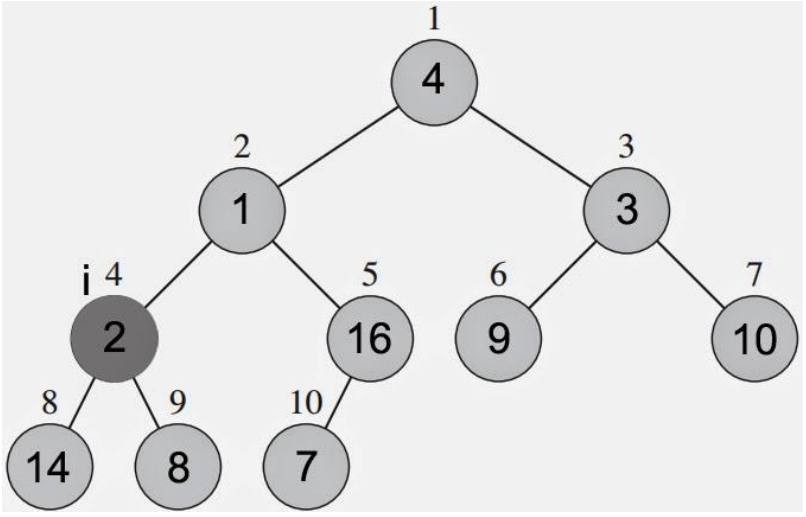
Dan hasil heap treenya adalah sebagai berikut :



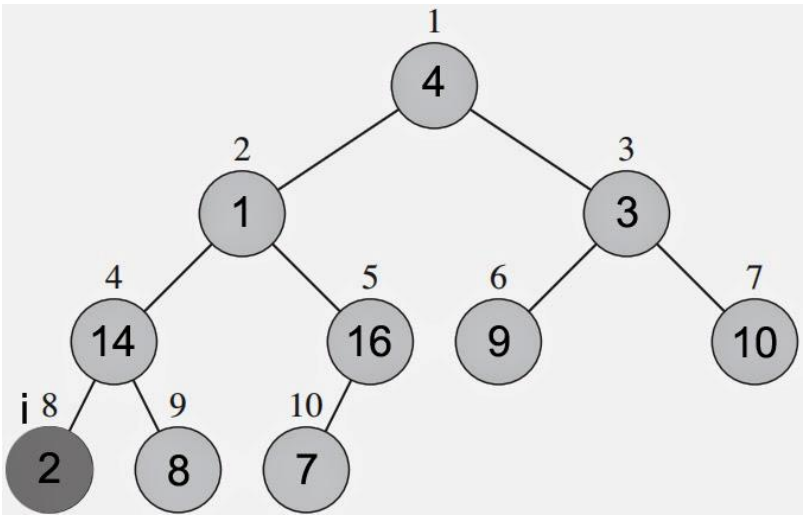
Karena pada heap tree jumlah node/elemen ada 10 (kalo di array adalah 9) maka jumlah elemen di bagi 2 (10 / 2 = 5 atau ⌊9 / 2⌋ = 4) maka yang menjadi variabel i adalah 5 (untuk heap tree) atau 4 (untuk array) maka ilustrasinya adalah sebagai berikut :



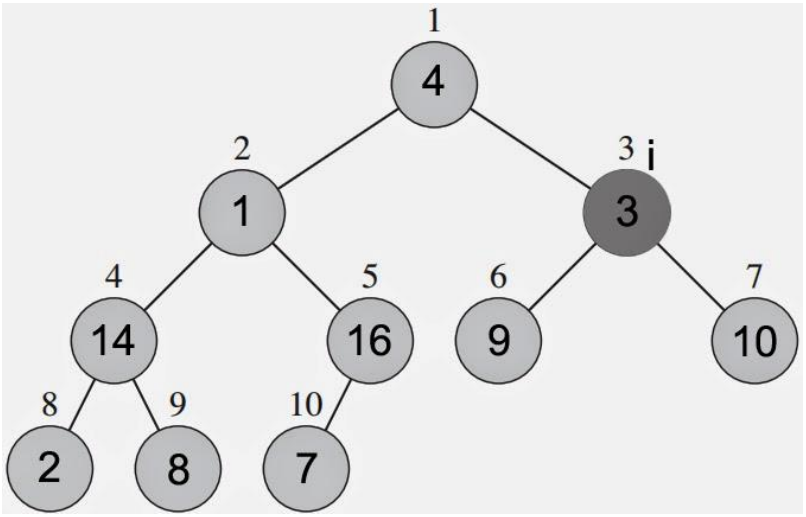
Karena node sebagai parent node sudah memiliki nilai lebih besar dari child nodenya (node 10) maka pertukaran posisi tidak terjadi, selanjutnya adalah lanjut ke index 4 :



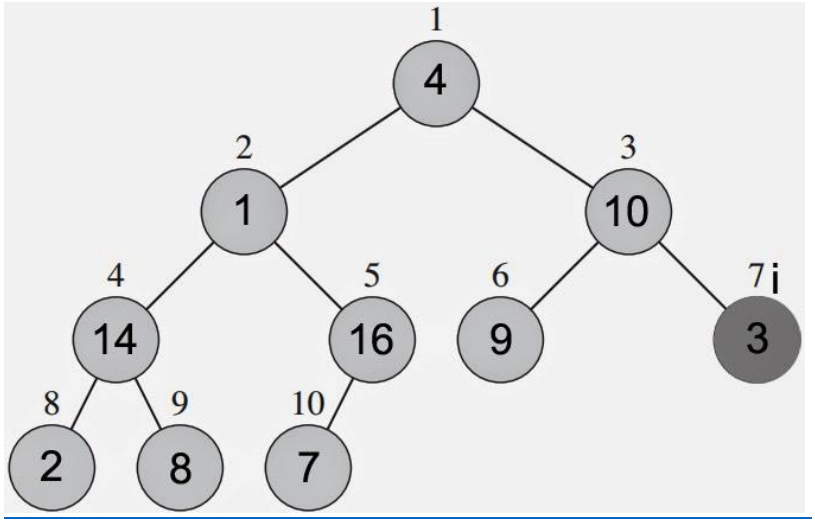
Disini ternyata nilai yang dimiliki node 4 lebih kecil dari nilai yang dimiliki child nodenya (node 8 dan 9) maka pertukaran posisi dilakukan dan hasilnya adalah sebagai berikut :



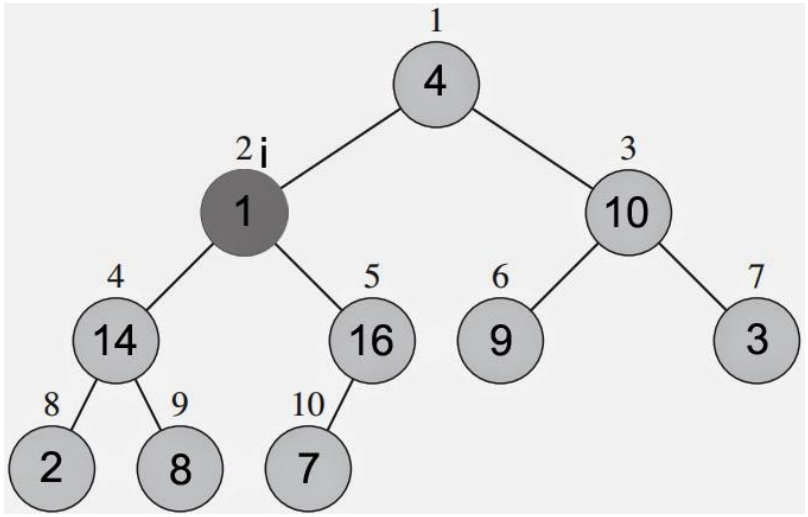
Selanjutnya adalah node 3 :



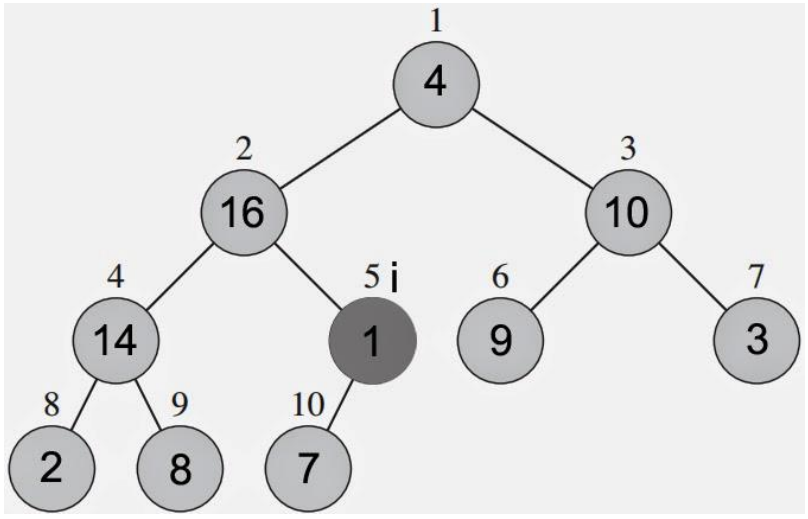
Disni node 3 memiliki nilai yang lebih kecil dari child nodenya (node 6 dan 7) maka pertukaran posisi dilakukan antara node 3 dan 7 karena antara node 3, 6, dan 7 node 7 lah yang memiliki nilai yang paling tinggi. Maka hasilnya adalah sebagai berikut :



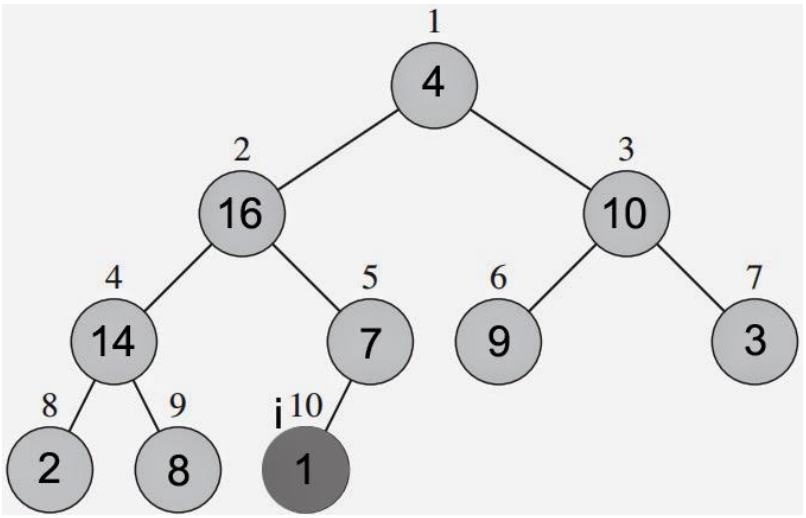
Selanjutnya adalah node 2 :



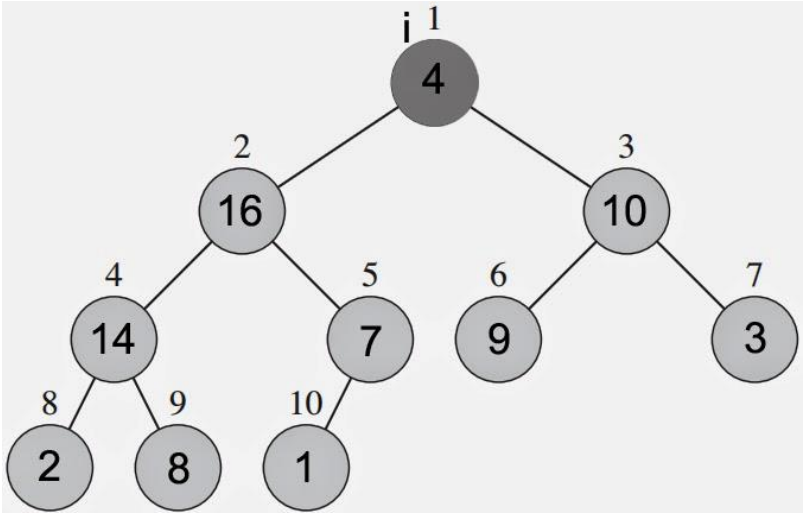
Node 2 memiliki nilai yang lebih kecil dari child nodenya (node 4 dan 5) maka pertukaran posisi dilakukan disini node 5 memiliki nilai yang paling tinggi maka nilai pada index 2 bertukar posisi dengan nilai pada index 5. Maka hasilnya adalah sebagai berikut :



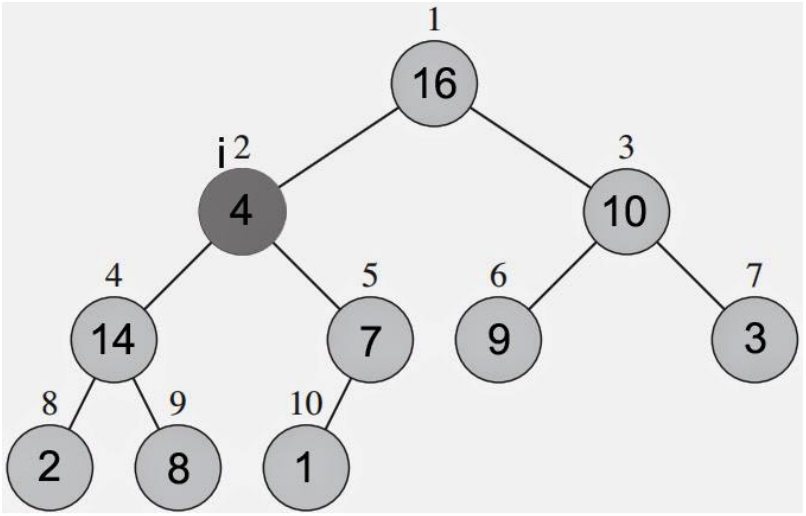
Setelah itu posisi index berubah menjadi pada node 5 dan ternyata disini nilai pada node 5 lebih kecil dari nilai yang berada di child nodenya yaitu node 10 maka perpindahan posisi kembali dilakukan dan hasilnya terlihat seperti ini :



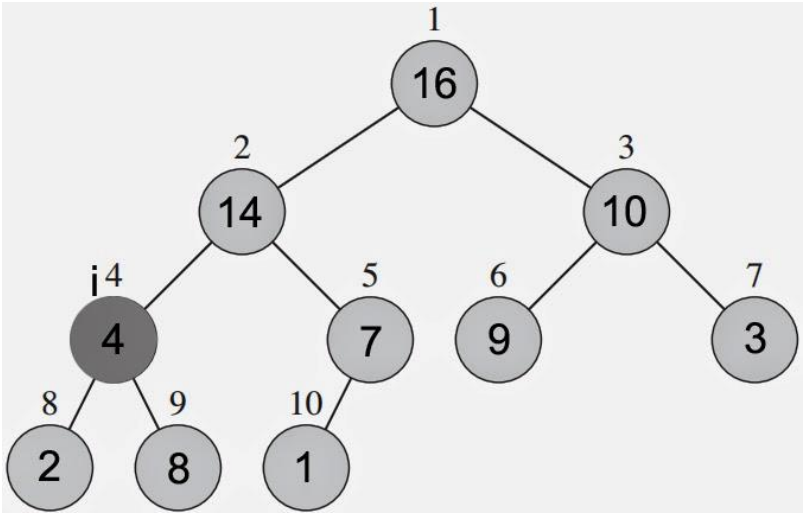
Selanjutnya posisi index berada di node 1 :



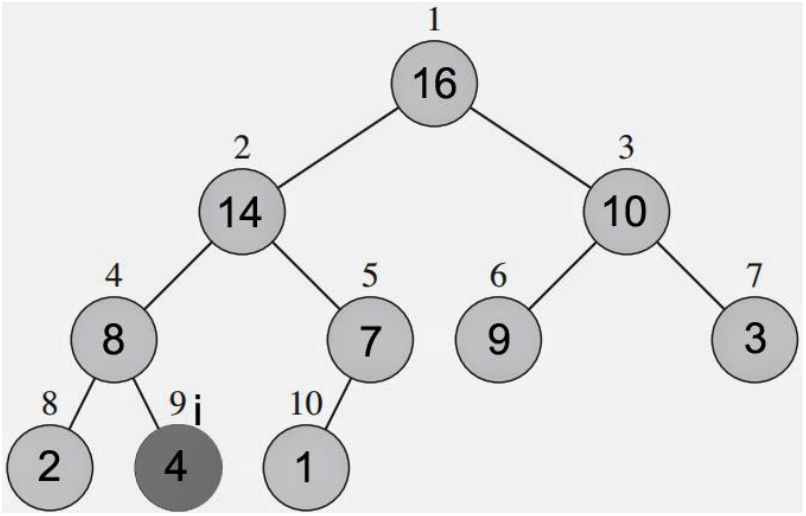
Disini nilai pada index 1 lebih kecil dbandingkan dengan nilai yang dimiliki child nodenya (node 2 dan 3) maka nilai pada node 1 ditukar dengan nilai pada node 2 karena nilai pada node 2 adalah nilai tertinggi dari nilai pada 1 dan 3 dan hasilnya adalah :



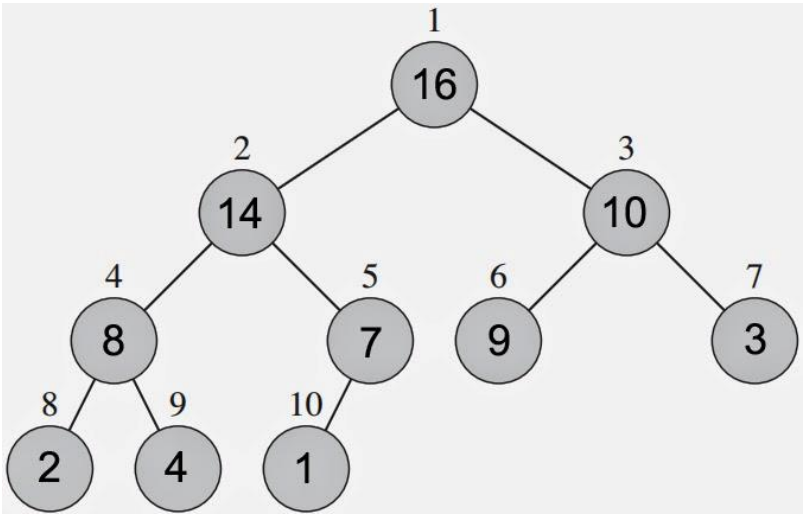
Setelah pertukaran posisi sekarang posisi ***i*** ada di index 2 disini nilai pada index 2 lebih kecil dibandingkan nila yang dimiliki child nodenya (index 4 dan 5) maka pertukaran posisi kembali terjadi disini index 4 memiliki nilai tertinggi maka nilai pada indx i ditukar dengan nilai pada index 4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



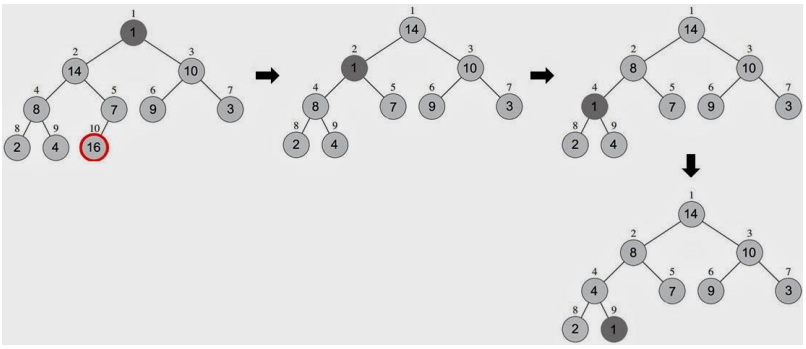
Setelah proses tukar posisi selesai maka posisi i adalah di node 4 akan tetapi disini kembali nilai pada index i lebih kecil dibandingkan dengan nilai yang dimiliki child nodenya (node 9), maka nilai pada index i ditukar dengan nilai pada index 9 dan hasilnya adalah sebagai berikut :



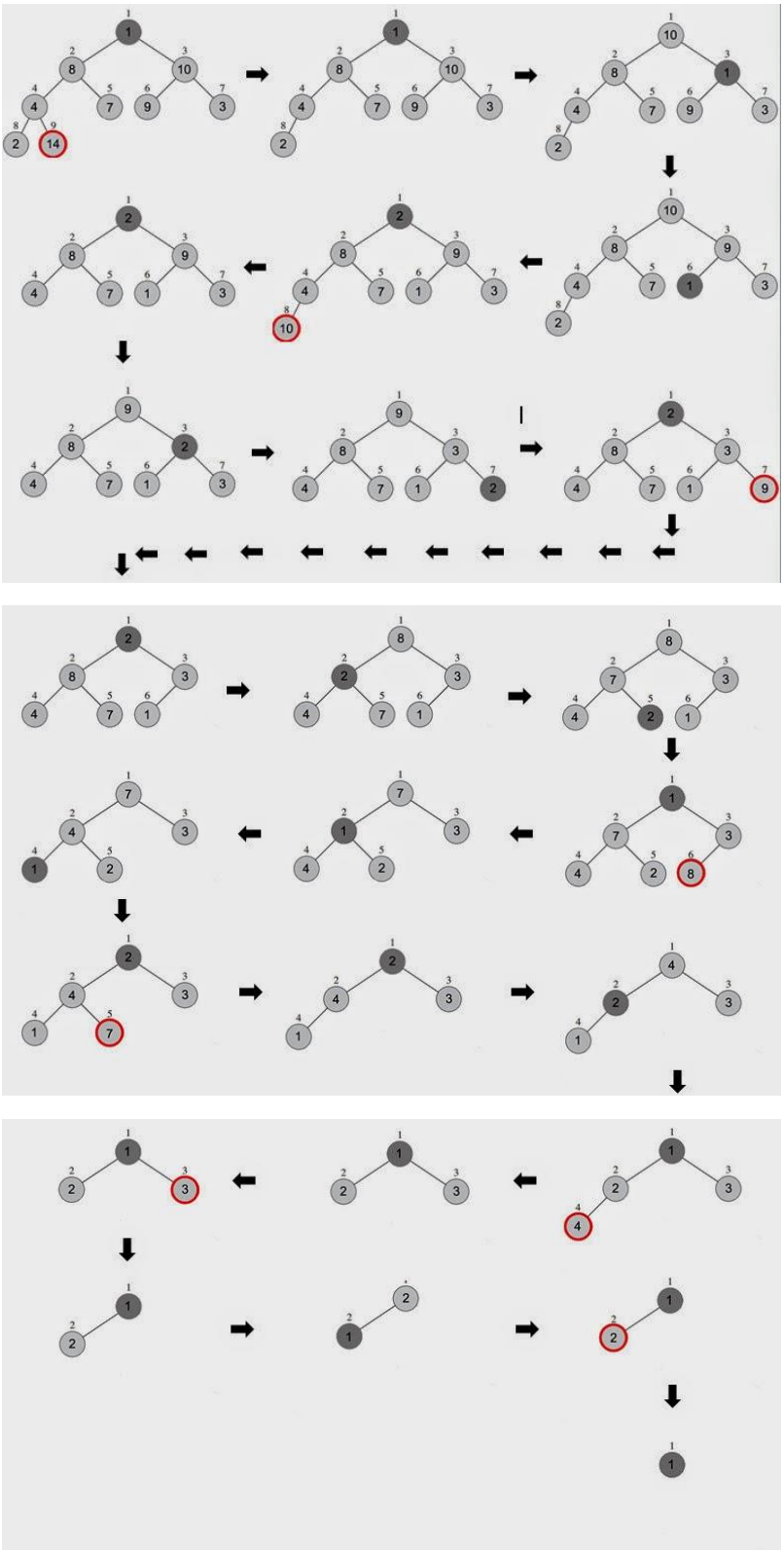
Sampai disini maka proes Build-Max-Heap telah selesai karena masing-masing parent node memiliki nilai yang lebih besar dari child nodenya, yaitu seperti heap tree berikut :



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 :



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 :



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.

Kompleksitas Waktu :

Kompleksitas Waktu untuk Hipfy adalah O(n).

Kompleksitas Waktu untuk createAndBuildHeap() adalah O(log n)

Jadi Kompleksitas waktu total untuk Hipsort adalah O(n log n).