PRAKTIKUM ANALISIS ALGORITMA

Untuk Memenuhi Tugas

Disusun oleh :

Mutia Karimah **140810170002**

TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

2019

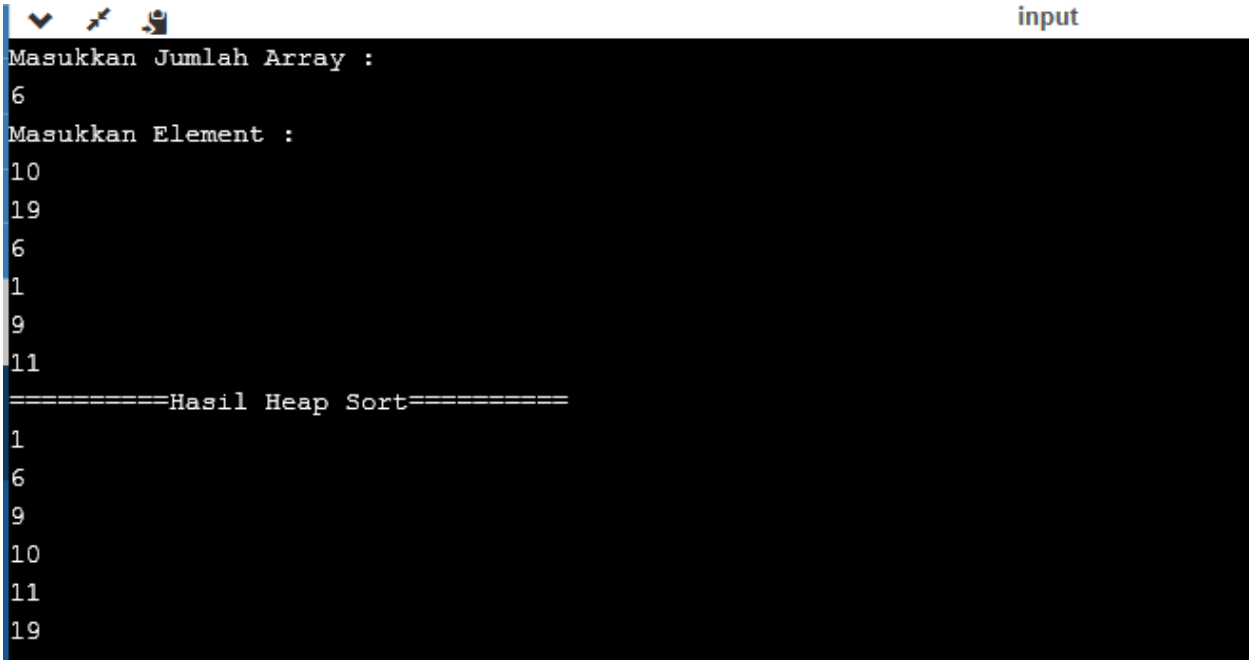
Tugas 1

HeapSort.cpp

```
#include <iostream>

using namespace std;


void MAX_HEAPIFY(int a[], int i, int n)

{

  int l,r,largest,loc;

  l=2*i;

  r=(2*i+1);

  if((l<=n)&&a[l]>a[i])

    largest=l;

  else

    largest=i;

  if((r<=n)&&(a[r]>a[largest]))

    largest=r;

  if(largest!=i)

    {

      loc=a[i];

      a[i]=a[largest];

      a[largest]=loc;

      MAX_HEAPIFY(a, largest,n);

    }
```

```
}
void BUILD_MAX_HEAP(int a[], int n)
{
  for(int k = n/2; k >= 1; k--)
    {
      MAX_HEAPIFY(a, k, n);
    }
}
void HEAPSORT(int a[], int n)
{

  BUILD_MAX_HEAP(a,n);
  int i, temp;
  for (i = n; i >= 2; i--)
    {
      temp = a[i];
      a[i] = a[1];
      a[1] = temp;
      MAX_HEAPIFY(a, 1, i - 1);
    }
}

int main()
```

```
{
  int n;
  cout<<"Masukkan Jumlah Array : "<<endl;
  cin>>n;
  int a[n];
  cout<<"Masukkan Element : "<<endl;
  for (int i = 1; i <= n; i++)
    {
      cin>>a[i];
    }
  HEAPSORT(a, n);
  cout<<"==========Hasil Heap Sort=========="<<endl;
  for (int i = 1; i <= n; i++)
    {
      cout<<a[i]<<endl;
    }
    cout<<endl;
          return 0;
}
```



Kompleksitas waktu dan big-O :

Algoritma pengurutan Heap Sort merupakan salah satu metode pengurutan tercepat setelah Merge Sort dan Quick Sort dengan kompleksitas O(n log n)

Pseudo-code

```
BUILD-HEAP(A)
   heapsize := size(A);
   for i := floor(heapsize/2) downto 1
      do HEAPIFY(A, i);
   end for
END
```

Kompleksitas Waktu

$$T(n) = \sum_{h=0}^{log(n)} \left(\frac{n}{2^{h+1}}\right) * O(h)$$

$$= O\left(n * \sum_{h=0}^{\log(n)} \frac{h}{2^h}\right)$$

$$= O\left(n * \sum_{h=0}^{\infty} \frac{h}{2^h}\right)$$

Big-O notation

$$\sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$$

$$\sum_{n=0}^{\infty} nx^n = \frac{x}{(1-x)^2}$$

$$= O\left(n * \frac{\frac{1}{2}}{1-\frac{1}{2}}\right)$$

$$= O(n*2)$$

$$= O(n)$$

T(n) = O(n) + O(lg n) O(n)

=> T(n) = O(n lg n)

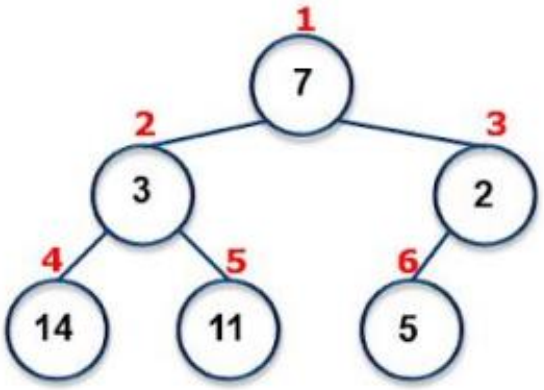
Step by step :

Misal Input yang dimasukkan adalah :

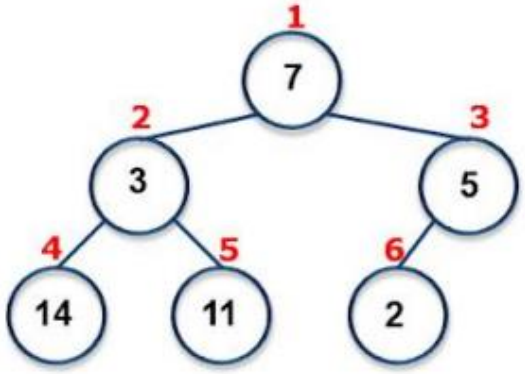
Misal Input yang dimasukkan adalah :

7	3	2	14	11	5
1	2	3	4	5	6

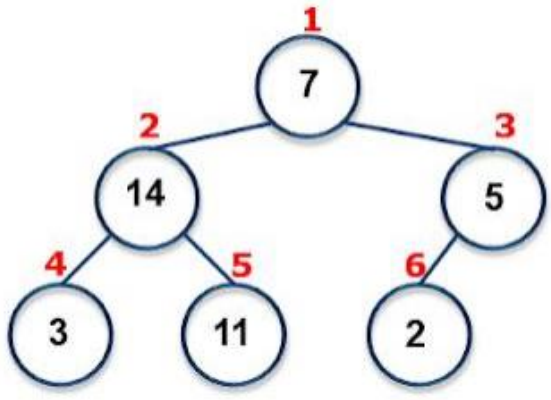
Konversi kedalam bentuk binary tree seperti ini :



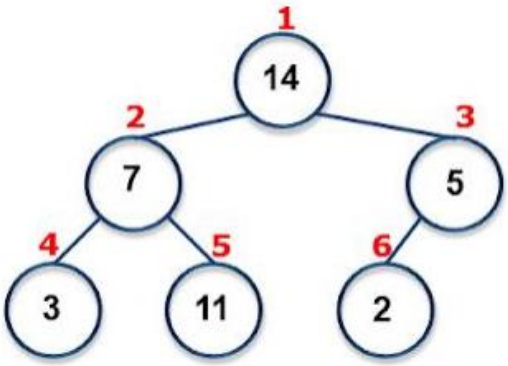
Jika sudah menjadi CBT lalu lakukan proses pengurutan secara max heap dengan cara banyaknya simpul dibagi dua untuk mencari nilai tengah dari sebuah array, sebagai contoh N = 6, Tengah = 6/2 = 3. Lalu lakukan reorganisasi pada simpul atau node ke-3. Dengan cara jika angka yang sekarang dibandingkan dengan angka selanjutnya yang ada di node turunannya itu lebih kecil maka tukar posisi.

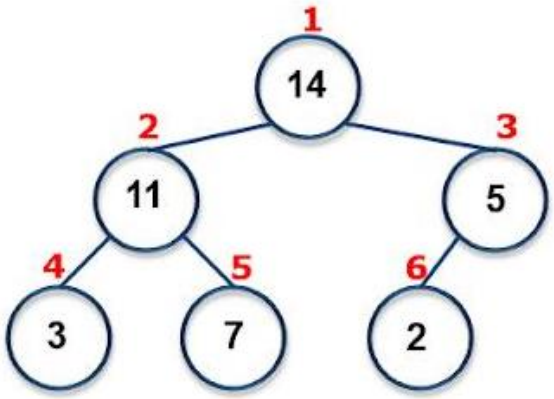


Lalu, lakukan reorganisasi pada simpul ke-2.



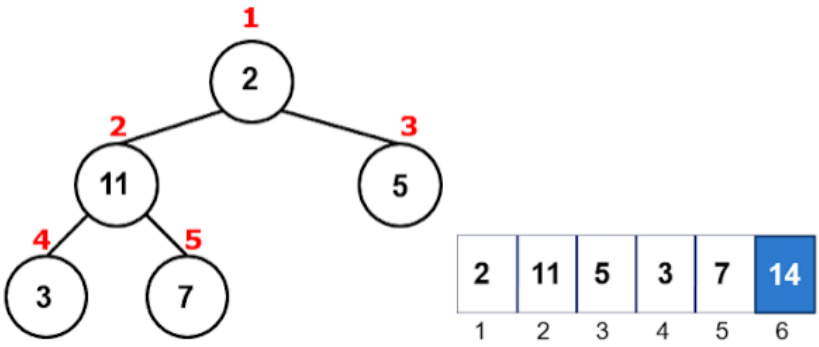
Lalu lakukan juga pada simpul ke-1





Terjadi perulangan karena angka 7 itu lebih kecil dari 14 dan 11, tetapi tidak lebih kecil dari 3. Maka dari itu dipindah posisikan sebanyak dua kali. Dan hasilnya adalah sebagai berikut :

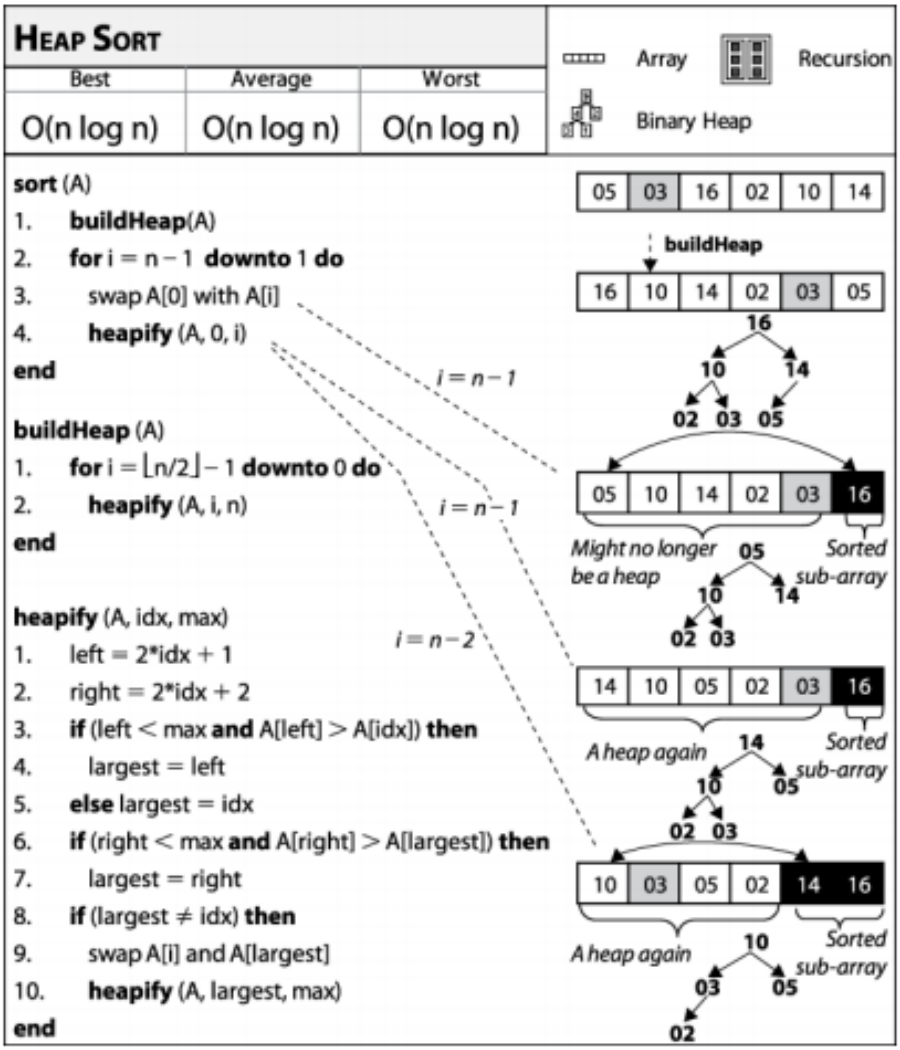
Hapus atau "Pecat" root dan tukarkan dengan simpul pada posisi terakhir. Banyaknya simpul dikurangi 1. Jika n lebih dari 1, maka lakukan reorganisasi heap. Lakukan langkah ke-2 hingga ke-5 sampai n = 0.



Karena datanya Binary Tree tidak dalam keadaan Max Heap, maka harus dilakukan lagi pembentukan heap agar menjadi max Heap. Lakukan terus hingga n = 0. Sehingga hasilnya dapat dilihat sebagai berikut.

Contoh soal dan running time :

T(n) = O(n) + O(lg n) O(n) => T(n) = O(n lg n)



Quick sort: Time taken for execution: 0.005288
Heap sort: Time taken for execution: 0.234245