

LAPORAN PRATIUM STRUKTUR DATA

MODUL PRAKTIKUM 9 HEAPS

**Dosen Pengampu :
TRI MUKTI LESTARI, M.Kom**

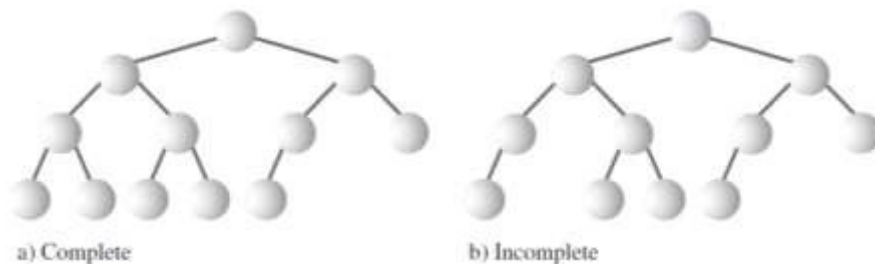


**Oleh :
Firna (220605110017)**

**Kelas J
JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2023**

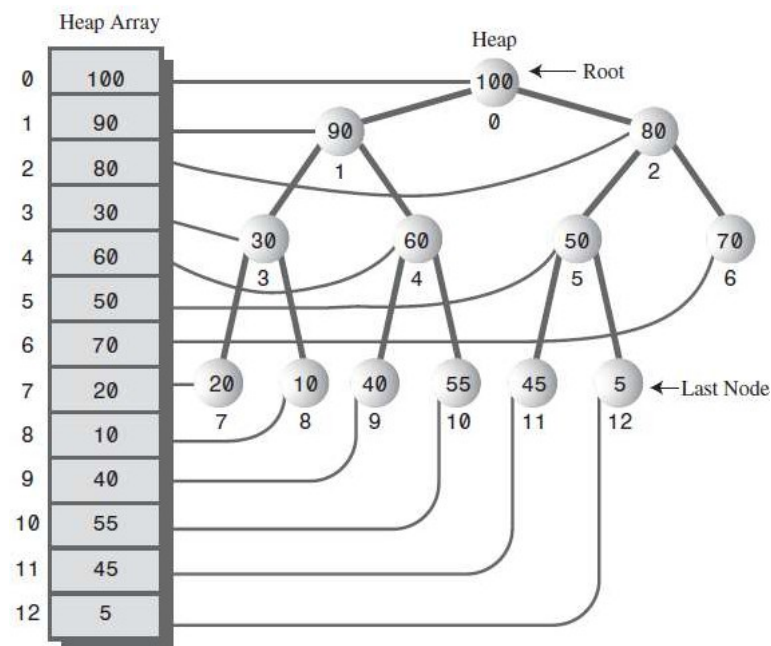
Heap adalah sebuah binary tree yang memiliki karakteristik berikut ini:

1. Binary tree pada heap adalah tree yang complete, yaitu node pada tiap levelnya penuh dari kiri hingga ke kanan, hanya pada level terakhir yang dibolehkan tidak penuh.



Gambar 9.1 Binary tree yang komplit dan tidak komplit

2. Sebuah heap biasanya diimplementasikan sebagai sebuah array daripada menggunakan reference untuk menghubungkan tiap node
3. Setiap node pada sebuah heap memenuhi kondisi/syarat heap, yaitu:
 - Setiap key dari parent node lebih besar atau sama dengan key dari children node tersebut ($\text{Parent}(\text{key}) \geq \text{Children}(\text{key})$) untuk Heap MAX
 - Setiap key dari parent node lebih kecil atau sama dengan key dari children node tersebut ($\text{Parent}(\text{key}) \leq \text{Children}(\text{key})$) untuk Heap MIN



Gambar 9.2 Contoh sebuah heap dan penyimpanan heap pada Array Tampak

pada gambar 9.2, sebuah heap sebagai bentuk binary tree yang complete ketika direpresentasikan pada sebuah array, maka tiap cell array tersebut akan terisi penuh mulai indeks ke-0 hingga N-1.

Struktur data heap dapat digunakan untuk mengimplementasikan ADT priority queues, yaitu bentuk queue yang tersusun berdasarkan prioritas tiap item. Tingkat prioritas tersebut dinyatakan dengan key setiap item. Priority queue yang diimplementasikan menggunakan sebuah Heap MAX, dimana nilai key maksimum berada pada root, disebut dengan descending-priority queues. Sebaliknya, ketika diimplementasikan pada Heap MIN (root berisi nilai key minimal), maka disebut ascending-priority queues.

A. PENDAHULUAN

1. Listing program java untuk Heap

Berikut ini listing program untuk Heap. Tulis dan pelajari listing ini

public class Node {	Awal Class Node
private int data;	deklarasi variabel bertipe int bernama data
public Node(int key) {	konstruktor dari kelas Node berparameter int key
data = key;	inisialisasi nilai dari parameter key ke data
}	penutup konstruktor
public int getKey() {	deklarasi method public int getKey
return data;	mengembalikan nilai dari variabel data
}	penutup method public getKey
public void setKey(int id) {	deklarasi method public void setKey
data = id;	mengatur nilai variabel data sama dengan nilai id
}	Akhir method public setKey
}	Akhir class Node

public class Heap {	Deklarasi awal class Heap
private Node[] heapArray;	Deklarasi Node bernama HeapArray
private int maxSize;	Deklarasi variabel int maxSize
private int currentSize;	Deklarasi variabel int currentSize
public Heap(int size) {	Konstruktor Heap berparameter size
maxSize = size;	Mengatur nilai maxSize = size
currentSize = 0;	Mengatur currentSize = 0
heapArray = new Node[size];	Deklarasi array baru heapArray = new Node[size]
}	Akhir konstruktor
public boolean isEmpty() {	Deklarasi method mengembalikan nilai boolean
return currentSize == 0;	Mengembalikan currentSize sama dengan 0
}	Penutup method public Boolean isEmpty() {

<pre> public boolean insert(int key) { if (currentSize == maxSize) { return false; } Node newNode = new Node(key); heapArray[currentSize] = newNode; trickleUp(currentSize++); return true; } </pre>	<p>Deklarasi method insert dengan parameter key bertipe integer akses publik Jika currentSize sama dengan maxSize Maka mengembalikan nilai false</p> <p>Deklarasi objek node Deklarasi heapArray = newNode Memanggil method trickleUp Mengembalikan nilai true Akhir method insert</p>
<pre> public void trickleUp(int index) { int parent = (index - 1) / 2; Node bottom = heapArray[index]; while (index > 0 && heapArray[parent].getKey() < bottom.getKey()) { heapArray[index] = heapArray[parent]; index = parent; parent = (parent - 1) / 2; } heapArray[index] = bottom; } </pre>	<p>Deklarasi method trickleUp dengan parameter int index Deklarasi var parent Deklarasi objek bottom</p> <p>Awal perulangan while ketika index lebih besar dari 0 dan heaparray[parent] kurang dari nilai bottom</p> <p>Deklarasi Heaparray[index] diisi dengan heap array [parent]</p> <p>Index diisi dengan nilai parent Parent diisi dengan nilai (parent - 1) / 2 Penutup while Heaparray [index] diisi dengan bottom Akhir method trickleUp</p>

<pre> public Node remove() { Node root = heapArray[0]; heapArray[0] = heapArray[--currentSize]; trickleDown(0); return root; } </pre>	<p>Deklarasi method remove</p> <p>Deklarasi variabel root = heaparray[0]</p> <p>Heaparray[0] diisi dengan heaparray [-- currentSize]</p> <p>Memanggil method trickledown</p> <p>Mengembalikan nilai root</p> <p>Akhir method remove</p>
<pre> public void trickleDown(int index) { int largerChild; Node top = heapArray[index]; while (index < currentSize / 2) { int leftChild = 2 * index + 1; int rightChild = leftChild + 1; if (rightChild < currentSize && heapArray[leftChild].getKey() < heapArray[rightChild].getKey()) { largerChild = rightChild; } else { largerChild = leftChild; } if (top.getKey() >= heapArray[largerChild].getKey()) { break; } heapArray[index] = heapArray[largerChild]; index = largerChild; } heapArray[index] = top; } </pre>	<p>Deklarasi method trickleDown dengan parameter int index</p> <p>Deklarasi largerchild bertipe int</p> <p>Deklarasi variabel top</p> <p>Perulangan while index kurang dari currentsize/2</p> <p>Leftchild diisi dengan atau sama dengan 2*index +1</p> <p>Rightchild diisi dengan atau sama dengan leftchild +1</p> <p>Jika rightchild < currentsize dan Heaparray[leftchild] nilainya kurang dari Heaparray[rightchild]</p> <p>Maka large largerchild diisi rightchild</p> <p>Jika kondisinya tidak sesuai</p> <p>Largechild diisi leftchild</p> <p>Jika nilai top.getKey lebih dari sama dengan Heaparray[largerchild]</p> <p>Perulangan berhenti</p> <p>Deklarasi Heaparray[index] diisi dengan Heaparray[largerchild]</p> <p>Deklarasi index diisi dengan largerchild</p> <p>Deklarasi heapArray yang diisi top</p> <p>Akhir method trickleDown</p>
<pre> public void displayHeap() { System.out.println("Heap Array: "); for (int i = 0; i < currentSize; i++) {if (heapArray[i] != null) { System.out.print(heapArray[i].getKey() + " "); } else { System.out.println("--"); } } System.out.println("");int nBlanks = 32; int itemsPerRow = 1;int column = 0; int j = 0; String dots = "."; System.out.println(dots + dots);while (currentSize > 0) { if (column == 0) { for (int k = 0; k < nBlanks; k++) {System.out.print(' '); } } System.out.print(heapArray[j].getKey());if (++j == currentSize) { break; } if (++column == itemsPerRow) { </pre>	

```

        nBlanks /= 2;
        itemsPerRow *= 2;
        column = 0;
        System.out.println();
    } else {
        for (int k = 0; k < nBlanks * 2 - 2; k++) {
            System.out.print(' ');
        }
    }
    System.out.println("\n" + dots + dots);
}

public void displayArray() {
    for (int j = 0; j < maxSize; j++) {
        System.out.print(heapArray[j].getKey() + " ");
    }
    System.out.println("");
}

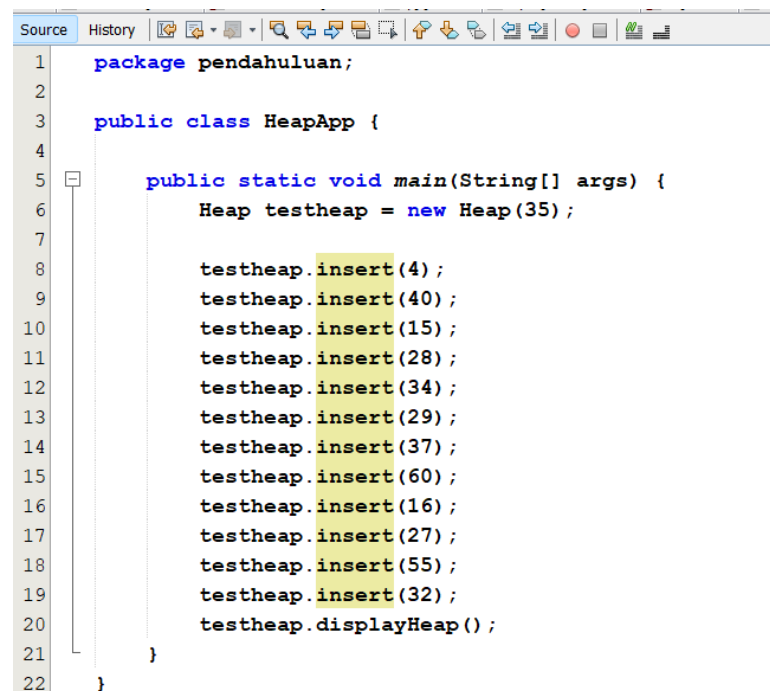
} //akhir class Heap

```

Lengkapi listing tersebut dengan sebuah class HeapApp berisi method main. deklarasikan sebuah heap dengan ukuran 35, lakukan penambahan 12 item, tampilkan heap tersebut. Jalankan program, bagaimana output program yang telah anda lengkapi?

JAWAB:

➤ Sourcecode Class HeapApp



```

1 package pendahuluan;
2
3 public class HeapApp {
4
5     public static void main(String[] args) {
6         Heap testheap = new Heap(35);
7
8         testheap.insert(4);
9         testheap.insert(40);
10        testheap.insert(15);
11        testheap.insert(28);
12        testheap.insert(34);
13        testheap.insert(29);
14        testheap.insert(37);
15        testheap.insert(60);
16        testheap.insert(16);
17        testheap.insert(27);
18        testheap.insert(55);
19        testheap.insert(32);
20        testheap.displayHeap();
21    }
22 }

```

➤ Output:

```

run:
Heap Array:
60 55 37 34 40 32 29 4 16 27 28 15
.....
                        60
                     /   \
                   55     37
                  /  \   /  \
                 34  40 32  29
                / \ / \ / \
               4 16 27 28 15
.....
BUILD SUCCESSFUL (total time: 0 seconds)

```

Dari output program tersebut, tampak bahwa heap yang diimplementasikan adalah

Heap MAX / ~~MIN~~ (*coret salah satu*)

Karena ini merupakan sifat dari heap MAX yang mana nilai terbesar (terbesar) akan berada pada node teratas atau root.

Pada program heap array yang saya tambahkan dalam class HeapApp, nilai maksimumnya adalah 60 maka dapat dilihat nilai 60 yang berada pada root (node teratas). Selanjutnya sifat Heap MAX lainnya, yaitu setiap node anak(simpul anak) memiliki nilai yang lebih kecil dari nilai node induk.

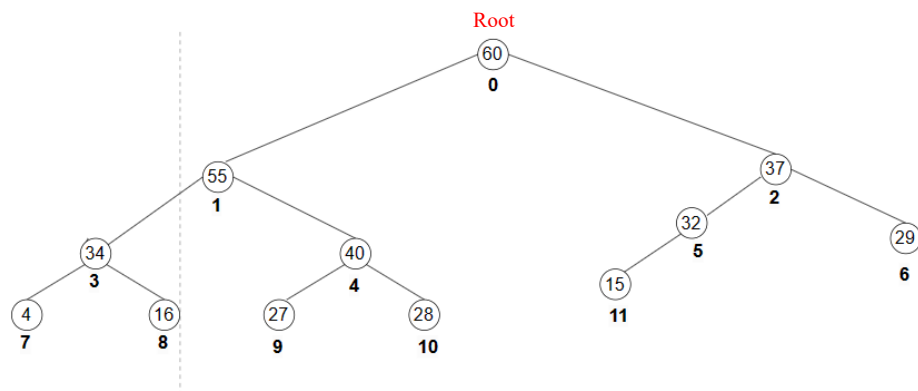
2. Heap Array

Perlu diingat bahwa heap adalah sebuah complete binary tree. Dalam bentuk heap (tree) maka akan jelas terlihat parent dan child suatu node. Sedangkan, jika heap tersebut direpresentasikan pada array (sehingga disebut Heap Array), maka visualisasi struktur tree tidak tampak, sehingga bagaimana mengetahui parent dan child suatu node? Pada heap array, tiap node disimpan pada tiap cell array, oleh karena itu, parent dan children suatu node dapat ditelusuri dari posisi indeks node tersebut.

Pada program nomor 1, tuliskan kembali tampilan heap dan heap array sesuai data yang telah anda masukkan! Tuliskan indeks tiap item heap array, begitu pula tuliskan indeks item tersebut pada heap. Amati dimana posisi parent dan children suatu node pada heap array! Pada heap Array, beri garis sebagai tanda hubung antara root node dengan left child dan right child dari root. Begitu pula seterusnya, jika right child dari root adalah parent node, hubungkan dengan left child dan right child node tersebut!

JAWAB:➤ **Tampilan Heap Array**

0	1	2	3	4	5	6	7	8	9	10	11
60	55	37	34	40	32	29	4	16	27	28	15

Ket:**Panah Hitam : Left Child****Panah Hijau: Right Child**➤ **Tampilan Heap**

Dari susunan array tersebut diketahui untuk sebuah node pada indeks x , maka:

Indeks Parent	Indeks Children	
x	<i>Left child</i>	$2 * x + 1$
	<i>Right child</i>	Left child + 1 atau $2 * x + 2$
$(x - 1) / 2$	x	

3. Menambahkan sebuah item pada Heap

Langkah untuk menambahkan item pada heap, yaitu:

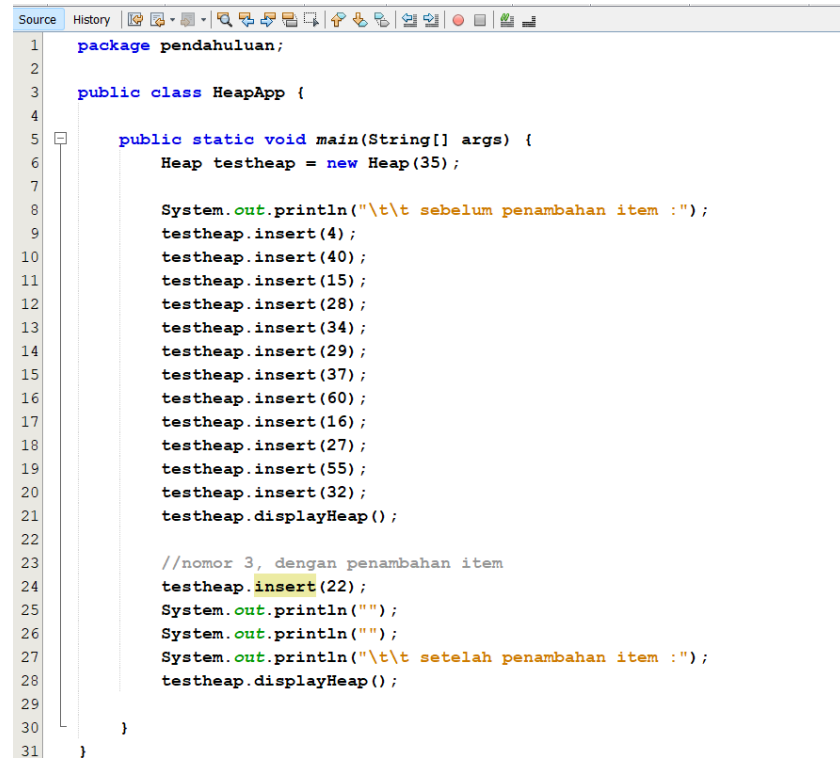
- Tambahkan node baru yang merepresentasikan sebuah item dengan key tertentu pada bagian akhir heap, yaitu sebagai last node
- Lakukan trickle up untuk menempatkan node baru tersebut sehingga berada di posisi yang tepat pada struktur heap

Trickle, bisa juga disebut dengan istilah bubble atau percolate, yaitu proses untuk memindahkan node baik keatas (trickle up) ataupun kebawah (trickle down) pada path node tersebut secara bertahap dengan cara membandingkan node di tiap tahap apakah node tersebut sudah berada pada posisi yang sesuai atau belum. Jika posisi tidak sesuai maka dilakukan pertukaran node tersebut dengan node yang dibandingkan hingga menjadi struktur heap yang sesuai.

Pada program nomor 1, dari 12 item yang telah ada, lakukan penambahan sebuah item. Tampilkan heap sebelum dan setelah penambahan item. Gambarkan langkah-langkah penambahan item tersebut mulai dari penambahan node baru sebagai last node hingga node tersebut berada pada posisi yang tepat sesuai struktur heap! Beri penjelasan pada tiap langkah tersebut!

JAWAB:

➤ SourceCode ClassHeapApp penambahan item:

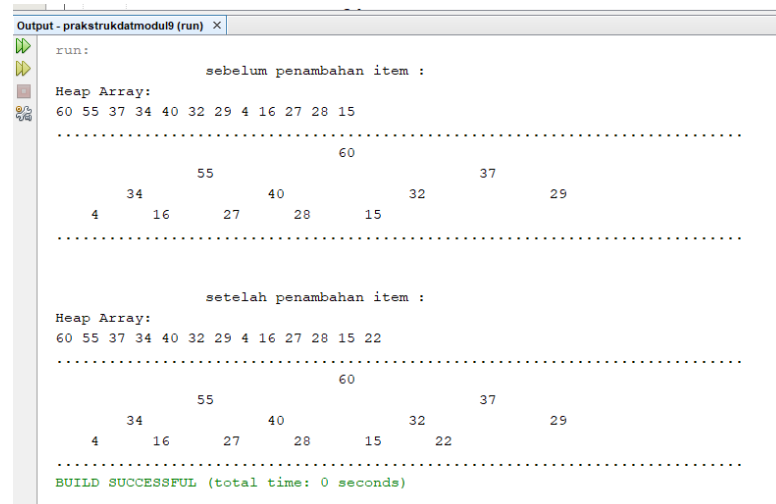


```

1 package pendahuluan;
2
3 public class HeapApp {
4
5     public static void main(String[] args) {
6         Heap testheap = new Heap(35);
7
8         System.out.println("\t\t sebelum penambahan item :");
9         testheap.insert(4);
10        testheap.insert(40);
11        testheap.insert(15);
12        testheap.insert(28);
13        testheap.insert(34);
14        testheap.insert(29);
15        testheap.insert(37);
16        testheap.insert(60);
17        testheap.insert(16);
18        testheap.insert(27);
19        testheap.insert(55);
20        testheap.insert(32);
21        testheap.displayHeap();
22
23        //nomor 3, dengan penambahan item
24        testheap.insert(22);
25        System.out.println("");
26        System.out.println("");
27        System.out.println("\t\t setelah penambahan item :");
28        testheap.displayHeap();
29    }
30 }
31

```

➤ Output:



```

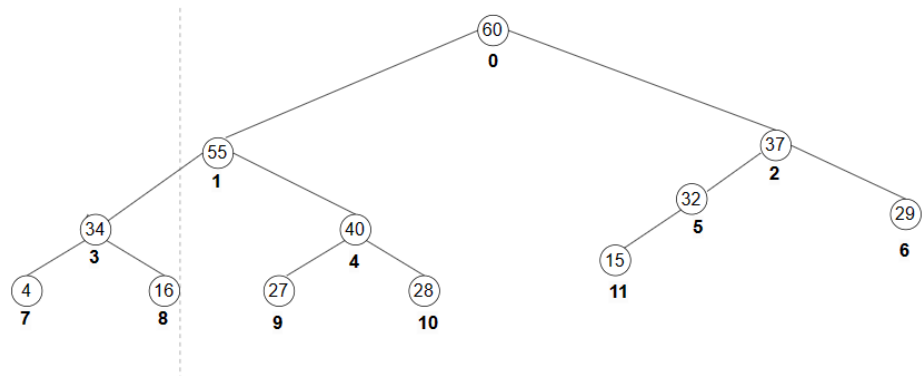
Output - prakstruktatmodul9 (run) X
run:
    sebelum penambahan item :
Heap Array:
60 55 37 34 40 32 29 4 16 27 28 15
.....
          55          37
        34          40          32          29
       4    16    27    28    15
.....

    setelah penambahan item :
Heap Array:
60 55 37 34 40 32 29 4 16 27 28 15 22
.....
          55          37
        34          40          32          29
       4    16    27    28    15    22
.....
BUILD SUCCESSFUL (total time: 0 seconds)

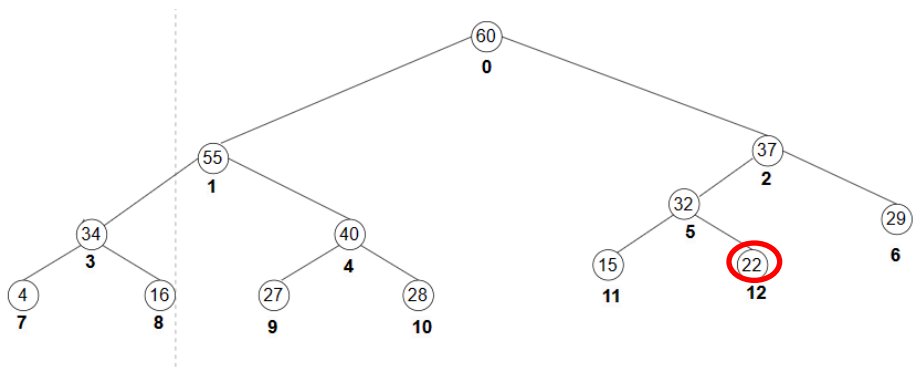
```

Proses pertama :

Pada proses pertama ini banyak node terhitung sebanyak 12.

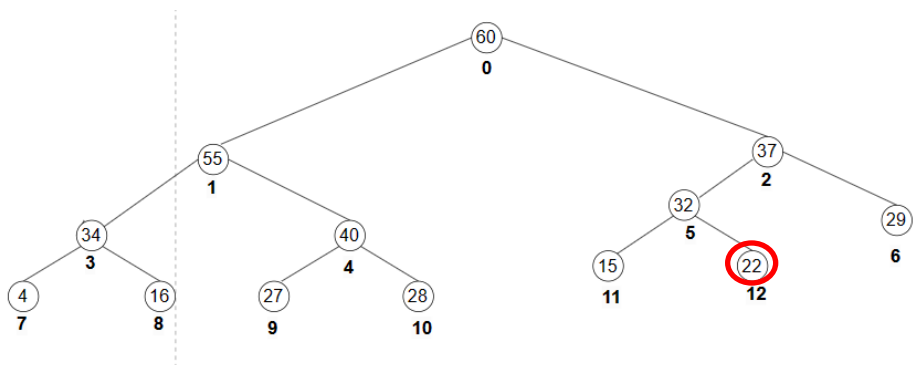
**Proses kedua :**

Dengan menambahkan source code `testheap.insert(22);` di class `HeapApp`. Maka node baru yang nilainya 22, dan menyisipkan node baru (22) tersebut pada posisi terakhir atau menjadi last node dengan index 12. Dan banyak node terhitung menjadi sebanyak 13.

**Proses ketiga:**

Lalu terjadi Trickle up (pengecekan nilai node dengan parent) apabila nilai node baru > dari parent maka posisi ditukar sehingga node baru menjadi parent, namun jika nilai node baru < parent maka posisi node baru tetap.

Pada program saya menambahkan nilai node baru (22) dibandingkan/dicek dengan parent nya (32). Karena $22 < 32$ maka tidak terjadi perubahan apapun dan node baru tetap tidak bertukar posisi. Dimana nilai node baru (22) tetap menjadi parent dan parent (32) tetap menjadi parentnya.



4. Menghapus sebuah item pada Heap

Perlu ingat bahwa removal pada heap berarti menghapus node yang memiliki nilai key maksimum (untuk heap MAX) atau menghapus node yang memiliki nilai key minimum (untuk heap MIN). Sehingga node yang dihapus adalah root node.

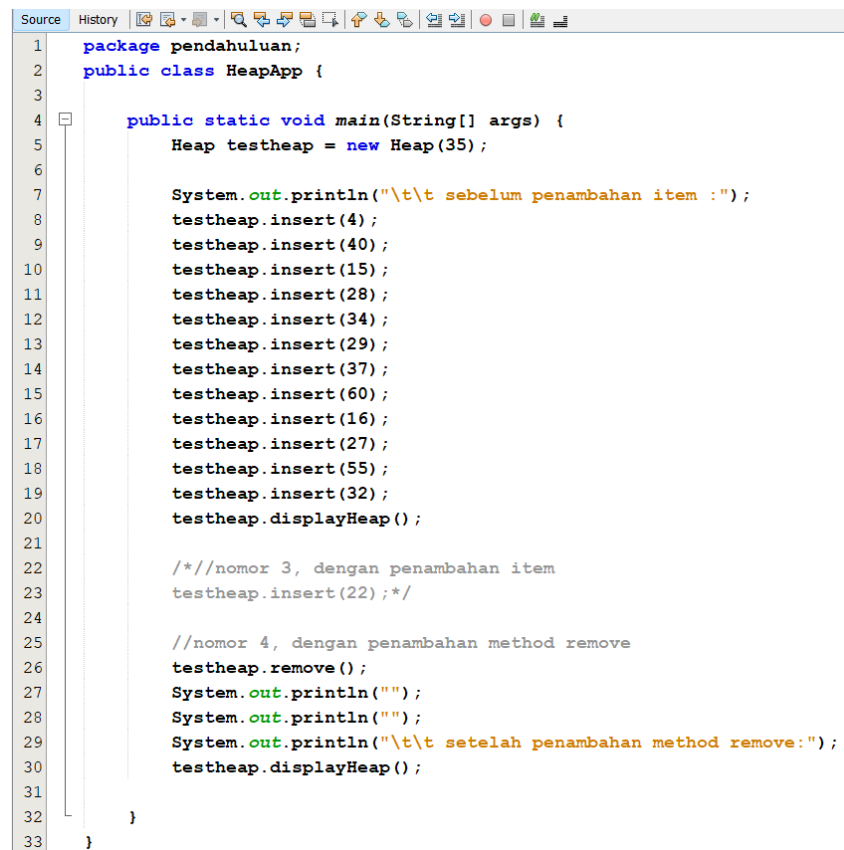
Berikut ini langkah-langkah removal pada heap:

- Hapus root node
- Pindahkan last node pada root
- Lakukan trickle down untuk menempatkan node tersebut pada posisi yang tepat sesuai struktur heap

Pada program nomor 1, panggil method `remove()` pada class `HeapApp`. Tampilkan heap sebelum dan setelah penghapusan. Gambarkan langkah-langkah removal/penghapusan tersebut! Jelaskan tiap tahapnya!

JAWAB:

➤ **SourceCode ClassHeapApp penambahan method remove:**



```

1 package pendahuluan;
2 public class HeapApp {
3
4     public static void main(String[] args) {
5         Heap testheap = new Heap(35);
6
7         System.out.println("\t\t sebelum penambahan item :");
8         testheap.insert(4);
9         testheap.insert(40);
10        testheap.insert(15);
11        testheap.insert(28);
12        testheap.insert(34);
13        testheap.insert(29);
14        testheap.insert(37);
15        testheap.insert(60);
16        testheap.insert(16);
17        testheap.insert(27);
18        testheap.insert(55);
19        testheap.insert(32);
20        testheap.displayHeap();
21
22        /*//nomor 3, dengan penambahan item
23        testheap.insert(22);*/
24
25        //nomor 4, dengan penambahan method remove
26        testheap.remove();
27        System.out.println("");
28        System.out.println("");
29        System.out.println("\t\t setelah penambahan method remove:");
30        testheap.displayHeap();
31    }
32 }
33

```

➤ **Output:**

```

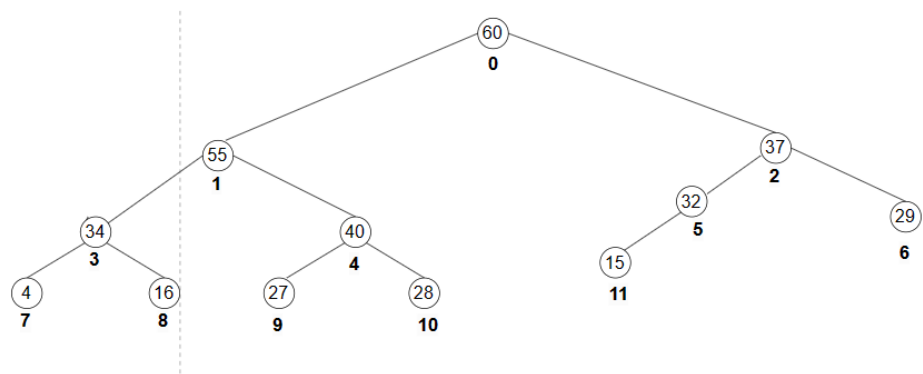
Output - prakstruktmodul9 (run)
run:
    sebelum penambahan item :
Heap Array:
60 55 37 34 40 32 29 4 16 27 28 15
.....
          60
        55      37
       34      40      32      29
      4 16 27 28 15
.....

    setelah penambahan method remove:
Heap Array:
55 40 37 34 28 32 29 4 16 27 15
.....
          55
        40      37
       34      28      32      29
      4 16 27 15
.....
BUILD SUCCESSFUL (total time: 0 seconds)

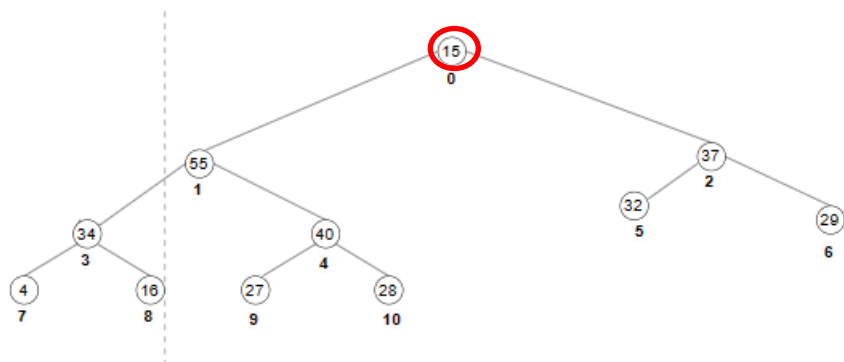
```

Proses pertama:

Pada proses pertama ini banyak node terhitung sebanyak 12.

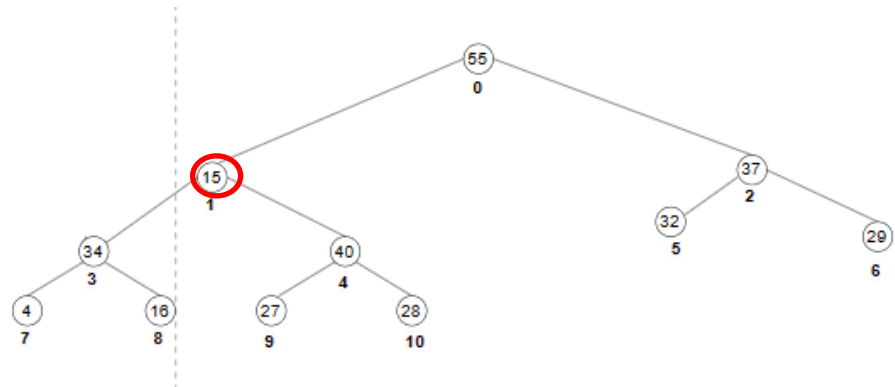
**Proses kedua :**

Pada proses kedua, method remove terjadi yang di hapus adalah 60(yang menjadi rootnya), sehingga tree menjadi seperti dibawah ini, dan banyak node berkurang menjadi sebanyak 11.

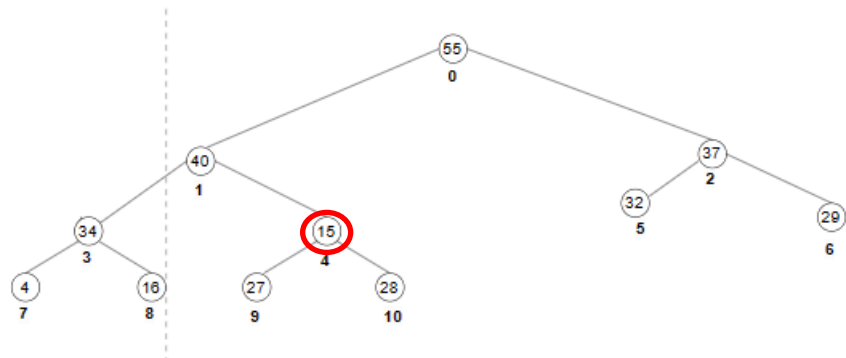


Proses ketiga:

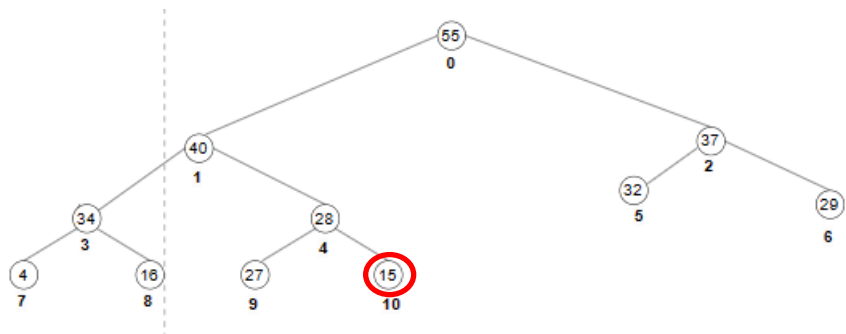
Kemudian terjadi Trickle down, dimana terjadi perbandingan antara root yang baru (15) dengan right child (37) dan left child (55). Karena antara RC dan LC lebih besar LC maka node LC (55) ditukar posisi dengan root (15). Sehingga 55 menjadi root dan 15 menjadi left child nya.

**Proses keempat:**

Pada proses keempat ini, proses trickle down akan terus berlanjut sampai node 15 menempati posisi yang tepat sesuai dengan struktur data heap.

**Proses Kelima:**

Proses kelima (proses terakhir) merupakan heap final. Dengan node 15 menjadi rightchild dari parent 28. Dan yang menjadi Root adalah 55.



5. Merubah key (priority)

Dengan adanya method `trickleUp()` dan `trickleDown()`, maka algoritma untuk merubah suatu key (representasi dari nilai prioritas pada priority queue) sebuah node dapat mudah diimplementasikan. Secara garis besar, berikut ini langkah untuk merubah key suatu node pada heap:

- Ubah nilai key pada node tertentu (ditunjukkan dengan indeks suatu cell pada heap array yang ingin dirubah) dengan nilai yang baru.
- Jika nilai key yang lama kurang dari nilai yang baru, maka lakukan `trickle up`
- Jika tidak, maka lakukan `trickle down`.

Tuliskan sebuah method *change* dengan parameter indeks dan nilai yang baru serta implmentasikan algoritma merubah key pada method tersebut! Panggil method tersebut pada class `HeapApp`, tampilkan heap sebelum perubahan key dan setelah key dirubah.

JAWAB:➤ Pada class `Heap` :

```

125 //nomor 5 mengubah key(method change)
126 public void change(int index, int key) {
127     Node newNode = new Node(key);
128     if (heapArray[index].getKey() < newNode.getKey()) {
129         heapArray[index] = newNode;
130         trickleUp(index++);
131     } else {
132         heapArray[index] = newNode;
133         trickleDown(index++);
134     }
135 }
```

➤ Pada class `HeapApp`:

```
Source History
1 package pendahuluan;
2 public class HeapApp {
3     public static void main(String[] args) {
4         Heap testheap = new Heap(35);
5         System.out.println("\t\t sebelum penambahan mehod change :");
6         testheap.insert(4);
7         testheap.insert(40);
8         testheap.insert(15);
9         testheap.insert(28);
10        testheap.insert(34);
11        testheap.insert(29);
12        testheap.insert(37);
13        testheap.insert(60);
14        testheap.insert(16);
15        testheap.insert(27);
16        testheap.insert(55);
17        testheap.insert(32);
18        testheap.displayHeap();
19
20        /*//nomor 3, dengan penambahan item
21        testheap.insert(22);*/
22
23        /*//nomor 4, dengan penambahan method remove
24        testheap.remove();*/
25
26        /*//nomor 5, dengan penambahan method change
27        testheap.change(4, 9);
28        System.out.println("");
29        System.out.println("");
30        System.out.println("\t\t setelah penambahan method change:");
31        testheap.displayHeap();
32
33    }
34 }
```


➤ **Output:**

```

Output - prakstruktatmodul9 (run)

run:
    sebelum penambahan mehod change :
Heap Array:
60 55 37 34 40 32 29 4 16 27 28 15
.....
          60
        /  \
       55   37
      /  \   \
     34  40  32  29
    /  \   /  \
   4   16 27  28  15
.....

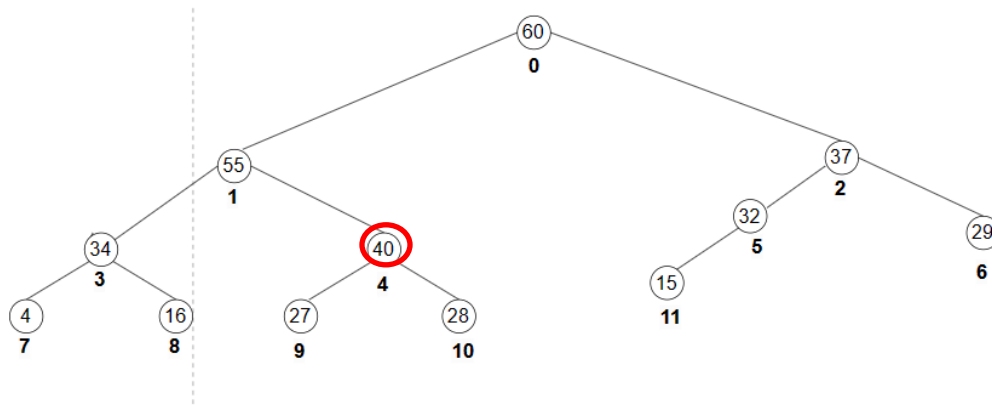
    setelah penambahan method change:
Heap Array:
60 55 37 34 28 32 29 4 16 27 9 15
.....
          60
        /  \
       55   37
      /  \   \
     34  28  32  29
    /  \   /  \
   4   16 27  9  15
.....

BUILD SUCCESSFUL (total time: 0 seconds)

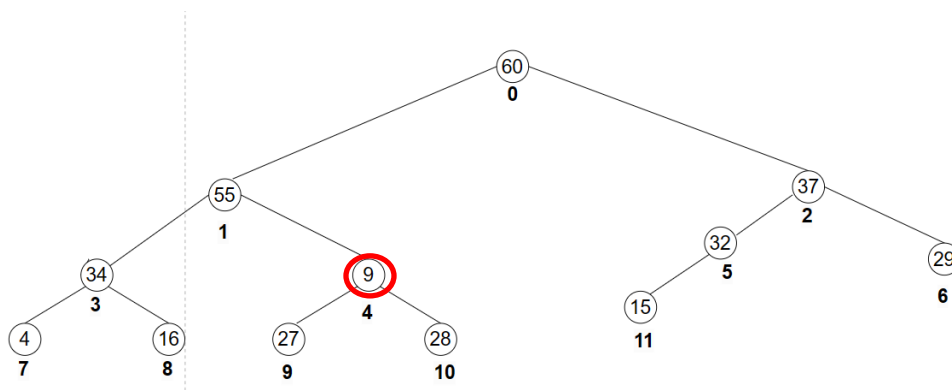
```

Proses Pertama:

Pertama terjadi pengkondisian, jika index yang dimasukkan lebih kecil dari nol atau index yang dimasukkan melebihi ukuran index maka akan false. Kemudian nilai node lama akan digantikan dengan nilai node baru yang dimasukkan setelah pemasukan index.

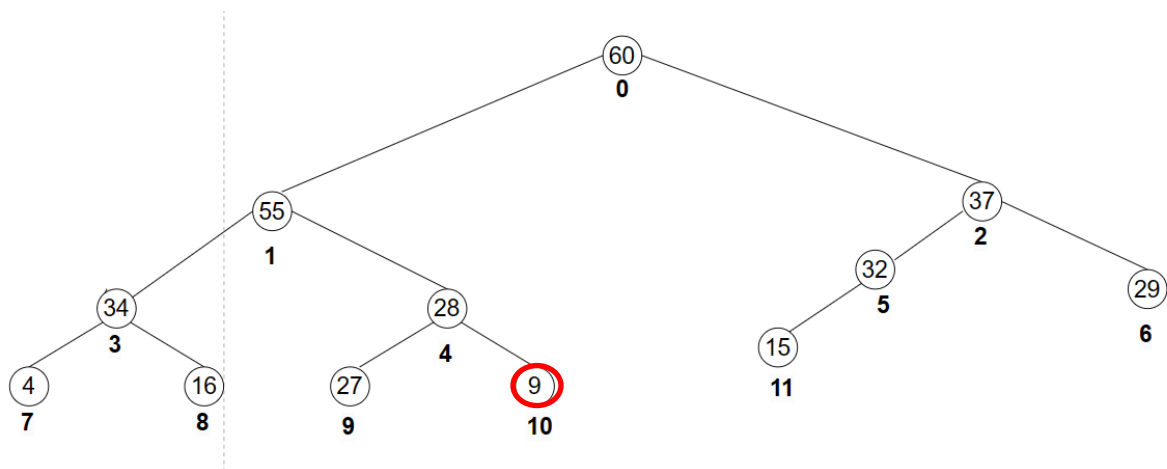
**Proses kedua:**

Proses kedua ini terjadi lagi pengkondisian kedua, yang mana jika nilai node yang lama lebih kecil dari nilai node yang baru maka akan terjadi proses trickleUp, dan jika nilai node yang lama lebih besar dari nilai node baru maka akan terjadi proses trickleDown.



Proses ketiga:

Pada proses ketiga ini sudah terlihat bahwa yang terjadi adalah proses trickle down. Dikarenakan, nilai node yang lama (40) lebih besar dari nilai node baru (9).



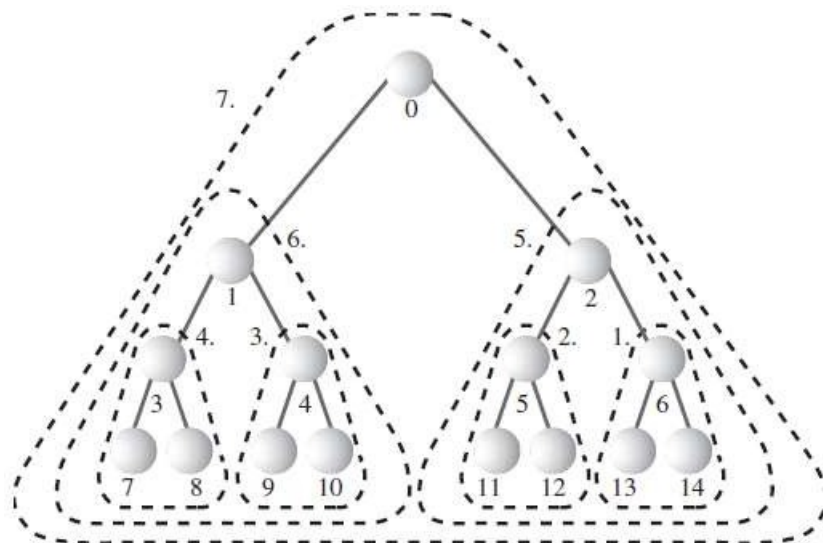
B. PRAKTIKUM

Implementasi lain dari struktur heap adalah digunakan untuk sorting yang dikenal sebagai **Heapsort**.

Buatlah program sorting menggunakan heapsort.

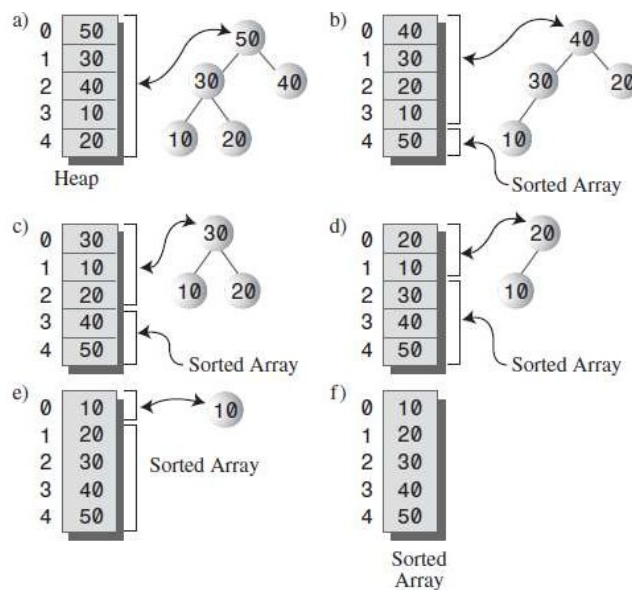
Anda dapat mengimplementasikannya berdasarkan langkah-langkah berikut ini:

- Tentukan array dengan data random
- Ubah posisi indeks tiap item pada array tersebut menjadi susunan struktur heap dengan cara melakukan trickle down item pada indeks ke- $(N/2-1)$ hingga indeks ke-0.



Gambar 9.3 urutan penerapan trickleDown

- Remove heap dan simpan dalam heapArray mulai indeks terakhir



Gambar 9.4 heap removal dan penyimpanan removed node pada heapArray

Langkah tersebut dapat diimplementasikan dalam 3 class, yaitu:

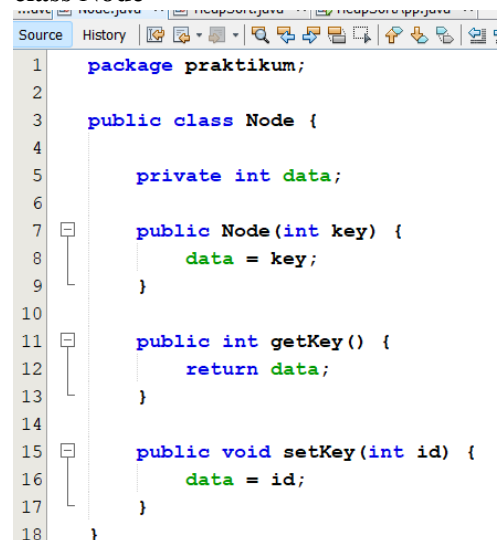
- a. Class Node, sebagai objek dari item
- b. Class Heap, secara umum, class ini sama dengan class Heap pada listing nomor 1 tugas pendahuluan, dengan beberapa penyesuaian sehingga dapat digunakan untuk melakukan pengurutan. Berikut ini method yang diperlukan:
 - Method Heap sebagai constructor dengan parameter size. Di dalamnya dilakukan inisialisasi variable maxSize, currentSize, dan heapArray
 - Method insertAt(indeks, value), berisi code untuk melakukan penambahan item pada heapArray pada indeks sesuai parameter.
 - Method trickleDown(indeks). Sama dengan listing nomor 1 tugas pendahuluan
 - Method displayArray(), untuk menampilkan isi array. Sama dengan listing nomor 1 tugas pendahuluan
 - Method HeapSort(), berisi code untuk trickle down item pada indeks ke- $(N/2-1)$ hingga indeks ke-0. Setelah itu, dilakukan removal secara berulang. Item yang dihapus (node dengan nilai maksimum) disimpan dalam heapArray mulai indeks yang terakhir hingga indeks pertama.
- c. Class HeapSortApp, berisi method main yang didalamnya dilakukan:
 - pemanggilan method insertAt(indeks, value) pada class heap untuk menambahkan item pada HeapArray. Lakukan pemanggilan method ini dalam perulangan dengan batas indeks tertentu. Value yang ditambahkan adalah nilai random.
 - menampilkan array sebelum diurutkan
 - memanggil method HeapSort() pada class Heap untuk menjalankan proses sorting
 - menampilkan array setelah diurutkan

Pengayaan

Modul ini memberikan gambaran implementasi Heap MAX. Sesuaikan listing program pada tugas pendahuluan modul ini sehingga merepresentasikan implementasi dari Heap MIN.

JAWAB:

➤ class Node



```

1  package praktikum;
2
3  public class Node {
4
5      private int data;
6
7      public Node(int key) {
8          data = key;
9      }
10
11     public int getKey() {
12         return data;
13     }
14
15     public void setKey(int id) {
16         data = id;
17     }
18 }
  
```

➤ class HeapSort

```

1  package praktikum;
2
3  public class HeapSort {
4
5      private Node[] heapArray;
6      private int maxSize;
7      private int currentSize;
8
9      public HeapSort(int size) {
10         maxSize = size;
11         currentSize = 0;
12         heapArray = new Node[size];
13     }
14
15     public void insertAt(int index, int value) {
16         Node newNode = new Node(value);
17         heapArray[index] = newNode;
18         trickleUp(currentSize++);
19     }
20
21     public void trickleUp(int index) {
22         int parent = (index - 1) / 2;
23         Node bottom = heapArray[index];
24
25         while (index > 0
26             && heapArray[parent].getKey()
27             < bottom.getKey()) {
28             heapArray[index] = heapArray[parent];
29             index = parent;
30             parent = (parent - 1) / 2;
31         }
32         heapArray[index] = bottom;
33     }
34
35     public Node remove() {
36         Node root = heapArray[0];
37         heapArray[0] = heapArray[--currentSize];
38         trickleDown(0);
39         return root;
40     }
41

```

```

41
42 public void trickleDown(int index) {
43     int largerChild;
44     Node top = heapArray[index];
45     while (index < currentSize / 2) {
46         int leftChild = 2 * index + 1;
47         int rightChild = leftChild + 1;
48         if (rightChild < currentSize
49             && heapArray[leftChild].getKey()
50             < heapArray[rightChild].getKey()) {
51             largerChild = rightChild;
52         } else {
53             largerChild = leftChild;
54         }
55         if (top.getKey() >= heapArray[largerChild].getKey()) {
56             break;
57         }
58         heapArray[index] = heapArray[largerChild];
59         index = largerChild;
60     }
61     heapArray[index] = top;
62 }
63
64 public void displayHeap() {
65     System.out.println("Heap Array: ");
66     for (int i = 0; i < currentSize; i++) {
67         if (heapArray[i] != null) {
68             System.out.print(heapArray[i].getKey() + " ");
69         } else {
70             System.out.println("--");
71         }
72     }
73
74     System.out.println("");
75     int nBlanks = 32;
76     int itemsPerRow = 1;
77     int column = 0;
78     int j = 0;
79     String dots = ".....";
80     System.out.println(dots + dots);
81     while (currentSize > 0) {
82         if (column == 0) {
83             for (int k = 0; k < nBlanks; k++) {
84                 System.out.print(' ');
85             }
86             System.out.print(heapArray[j].getKey());
87             if (++j == currentSize) {
88                 break;
89             }
90             if (++column == itemsPerRow) {
91                 nBlanks /= 2;
92                 itemsPerRow *= 2;
93                 column = 0;
94                 System.out.println();
95             } else {
96                 for (int k = 0; k < nBlanks * 2 - 2; k++) {
97                     System.out.print(' ');
98                 }
99             }
100         }
101         System.out.println("\n" + dots + dots);
102     }
103 }

```

```

103
104     public void displayArray() {
105         for (int j = 0; j < maxSize; j++) {
106             System.out.print(heapArray[j].getKey() + " ");
107         }
108         System.out.println("");
109     }
110
111     public void HeapSort() {
112         for (int i = maxSize - 1; i >= 0; i--) {
113             for (int j = maxSize / 2 - 1; j >= 0; j--) {
114                 trickleDown(j);
115             }
116             Node last = this.remove();
117             heapArray[i] = last;
118         }
119     }
120 }

```

➤ class HeapSortApp

```

Source History
1 package praktikum;
2
3 import java.util.Scanner;
4
5 public class HeapSortApp {
6
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9         System.out.println("");
10        System.out.println("Tentukan batas index: ");
11        int size = sc.nextInt();
12        HeapSort heap = new HeapSort(size);
13
14        for (int i = 0; i < size; i++) {
15            int random = (int) (Math.random() * 100);
16            heap.insertAt(i, random);
17        }
18
19        heap.displayHeap();
20        System.out.println("");
21        System.out.println("");
22        System.out.print("Array sebelum di Sort: ");
23        heap.displayArray();
24        heap.HeapSort();
25        System.out.println("");
26        System.out.println("");
27        System.out.print("Array setelah di Sort: ");
28        heap.displayArray();
29    }
30 }

```

➤ Output:

```
: Output - prakstrukdatmodul9 (run)
run:
Tentukan batas index:
11
Heap Array:
94 83 78 40 75 5 73 16 28 66 8
.....
          94
        83      78
      40    75    5    73
    16  28  66  8
    .....

Array sebelum di Sort: 94 83 78 40 75 5 73 16 28 66 8

Array setelah di Sort: 5 8 16 28 40 66 73 75 78 83 94
BUILD SUCCESSFUL (total time: 5 seconds)
```


C. KESIMPULAN

Kesimpulan yang diperoleh dari pembahasan praktikum kali ini adalah:

1. Tentang struktur data Heap

Jawab:

Struktur data Heap adalah sebuah binary tree yang memiliki karakteristik berikut ini:

- Binary tree pada heap adalah tree yang complete, yaitu node pada tiap levelnya penuh dari kiri hingga ke kanan, hanya pada level terakhir yang dibolehkan tidak penuh.
- Sebuah heap biasanya diimplementasikan sebagai sebuah array daripada menggunakan reference untuk menghubungkan tiap node
- Dan setiap node pada sebuah heap memenuhi kondisi/syarat heap, yaitu:
 - a. Setiap key dari parent node lebih besar atau sama dengan key dari children node tersebut ($\text{Parent}(\text{key}) \geq \text{Children}(\text{key})$) untuk Heap MAX
 - b. Setiap key dari parent node lebih kecil atau sama dengan key dari children node tersebut ($\text{Parent}(\text{key}) \leq \text{Children}(\text{key})$) untuk Heap MIN

2. Tentang manipulasi data pada heap: insert item, removal, change priority

Jawab:

- **Insert Item:** Proses memasukkan atau menambahkan data pada Heap Array, yang langkah awalnya adalah menempatkan data baru pada index terakhir yang kosong dalam array heap. Setelah itu, terjadi langkah selanjutnya yang disebut dengan trickle-up. Proses trickle-up memastikan bahwa data yang baru dimasukkan menempati posisi yang benar dalam struktur data heap. Pada langkah ini, nilai data baru dibandingkan dengan nilai induknya. Jika nilai induknya kecil (heap MAX) atau besar (heap MIN), terjadi pertukaran posisi antara data baru dan nilai induknya.
- **Removal:** Pada penghapusan node root dalam heap, nilai root diperbarui dengan nilai dari last node dari root awal, dan kemudian akan terjadi trickledown yang memastikan node baru tersebut menempati posisi yang sesuai dalam struktur heap. Trickledown ini melibatkan perbandingan nilai dengan children-nya, dengan pertukaran posisi jika diperlukan. Proses ini terus diulangi hingga kondisi heap terpenuhi setelah penghapusan node root.
- **Change priority:** proses mengganti nilai key pada heap melalui parameter index dan value, dengan menggunakan trickleup (apabila nilai node yang lama lebih kecil dari nilai node yang baru) dan trickledown (apabila nilai node yang lama lebih besar dari nilai node yang baru) sehingga key menempati posisi yang tepat sesuai dengan struktur data heap.

3. Tentang Heapsort

Jawab:

Heapsort adalah metode sorting angka pada sebuah array dengan cara menggunakan binary tree yaitu memvisualisasikan array menjadi sebuah binary tree, kemudian masing – masing nilainya di urutkan.

BERIKUT LINK DRIVE :

https://drive.google.com/drive/folders/14og7bQSaYtnwPmoFyGinoz90DdeyohpJ?usp=drive_link