2110211 โครงสร้างข้อมูลเบื้องต้น

Priority Queue

สมชาย ประสิทธิ์จูตระกูล

Priority Queue

- เป็น queue ชนิดพิเศษ (ข้อมูล "ลัดคิว" ได้)
- ข้อมูลมี "priority"
  - ข้อมูลที่มี priority สูงสุด จะลัดคิวไปอยู่ที่หัวคิว

```
Queue q = new PriorityQueue();
q.add(new Integer(23));
q.add(new Integer(2));
System.out.println( q.dequeue() + ", " );
q.add(new Integer(3));
q.add(new Integer(13));
q.add(new Integer(10));
while( !q.isEmpty() ) {
  System.out.println( q.dequeue() + ", " );
}
```

```
23, 13, 10, 3, 2,
```


Priority Queue

- เป็น queue ชนิดพิเศษ (ข้อมูล "ลัดคิว" ได้)
- ข้อมูลมี "priority"
  - ข้อมูลที่มี priority สูงสุด จะลัดคิวไปอยู่ที่หัวคิว

```
public interface PriorityQueue extends Queue {
  /**
   * remove the highest priority element in this queue
   * @return the highest priority element
   */
  public Object dequeue();

  /**
   * get the highest priority element in this queue
   * @return the highest priority element
   */
  public Object peek();
}
```


Priority Queue

```
PriorityQueue q = new ArrayPQ();
q.enqueue(new Integer(19));
q.enqueue(new Integer(2));
System.out.print(q.dequeue() + ", ");
q.enqueue(new Integer(30));
System.out.print(q.dequeue() + ", ");
q.enqueue(new Integer(1));
q.enqueue(new Integer(10));
while (!q.isEmpty()) {
  System.out.print(q.dequeue() + ", ");
}
```

```
2, 19, 1, 10, 30,
```

กำหนดให้ `Integer` มีค่ายิ่งน้อย `priority` ยิ่งมาก

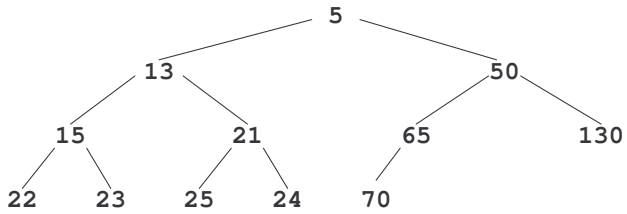
สร้าง Priority Queue แบบง่าย ๆ (แต่ช้า)

```
public class ArrayPQ implements PriorityQueue {
  private ArrayList list = new ArrayList();
  public boolean isEmpty() { return list.isEmpty(); }
  public int size() { return list.size(); }
  public Object peek() { return list.get(minIndex()); }
  public void enqueue(Object e) { list.add(e); }
  public Object dequeue() {
    int min = minIndex();
    Object result = list.get(min);
    list.remove(min);
    return result;
  }
  private int minIndex() {
    if (isEmpty()) throw new NoSuchElementException();
    int min = 0;
    for (int i = 1; i < list.size(); i++) {
      Comparable d = (Comparable) list.get(i);
      if (d.compareTo(list.get(min)) < 0) min = i;
    }
    return min;
  }
}
```

O(n)

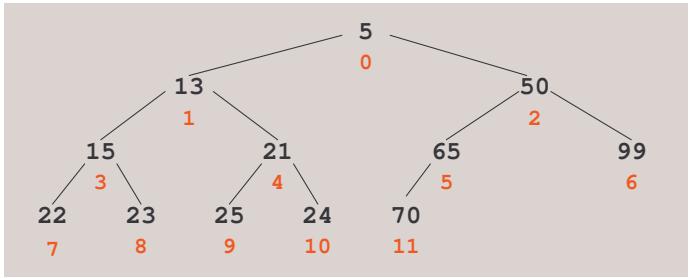
สร้าง Priority Queue ด้วย Binary Heap

- Binary Heap
  - มีโครงสร้างเป็น binary tree แบบได้ดุล
    - node เต็มทุกระดับ
    - ระดับล่างสุด เต็มจากซ้ายไปขวา
  - ข้อมูลของ parent node มีค่าน้อยกว่าของลูก ๆ



Binary Heap เก็บข้อมูลในอาเรย์

- ลูก<u>ซ้าย</u>ของ node ที่ index k อยู่ที่ index 2k + 1
- ลูก<u>ขวา</u>ของ node ที่ index k อยู่ที่ index 2k + 2
- พ่อของ node ที่ index k อยู่ที่ index (k – 1) / 2



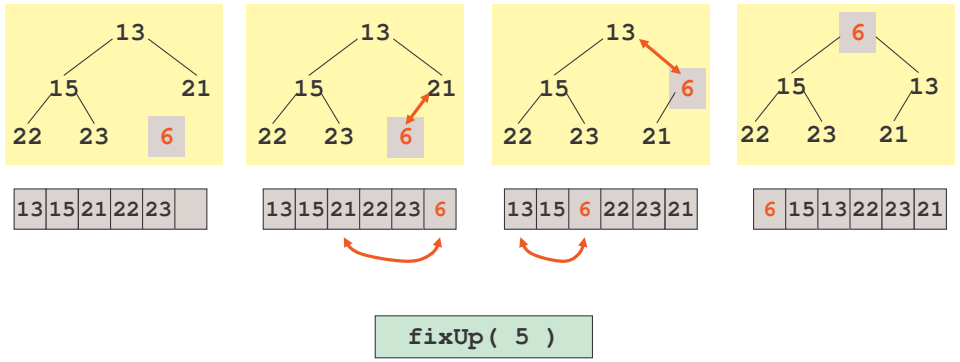
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
size: 12	5	13	50	15	21	65	99	22	23	25	24	70		
elementData														

คลาส BinaryHeap

```
public class BinaryHeap implements PriorityQueue {
  private Object[] elementData;
  private int size;

  public BinaryHeap() {
    elementData = new Object[10];
  }

  public boolean isEmpty() { return size == 0; }

  public int size() { return size; }

  public Object peek() {
    if (isEmpty()) throw new NoSuchElementException();
    return elementData[0];
  }
  ...
}
```

BinaryHeap : enqueue

- enqueue(x)
  - นำ x ไปต่อเป็นใบถัดไป (เพิ่มท้ายในอาเรย์)
  - ดัน x ขึ้นตามทางไปถึงราก จนกว่า x จะไม่น้อยกว่าพ่อ



BinaryHeap : enqueue

```
public class BinaryHeap implements PriorityQueue {
  private Object[] elementData;
  private int size;

  public void enqueue(Object e) {
    ensureCapacity(size+1);
    elementData[size] = e;
    fixUp(size++);
  }
  private void fixUp(int k) {
    Comparable tmp = (Comparable) elementData[k];
    while (k > 0) {
      int p = (k-1)/2;
      if (tmp.compareTo(elementData[p]) >= 0) break;
      elementData[k] = elementData[p];
      k = p;
    }
    elementData[k] = tmp;
  }
  ...
```

Comparable ?

จาวา : Comparable

- ที่ผ่านมาเราใช้ equals เปรียบเทียบออปเจกต์แบบ "เท่าไม่เท่า" a.equals(b)
  - equals เป็นเมท็อดมาตรฐานของคลาส Object
- ถ้าต้องการเปรียบเทียบแบบน้อยกว่า มากกว่า
  - ใช้ Comparable interface กำกับคลาส
  - คลาสที่ implements Comparable ต้องมีเมท็อด public int compareTo(Object that)
    - คืน 0 ถ้า this เท่ากับ that
    - คืนค่าที่น้อยกว่า 0 ถ้า this น้อยกว่า that
    - คืนค่าที่มากกว่า 0 ถ้า this มากกว่า that

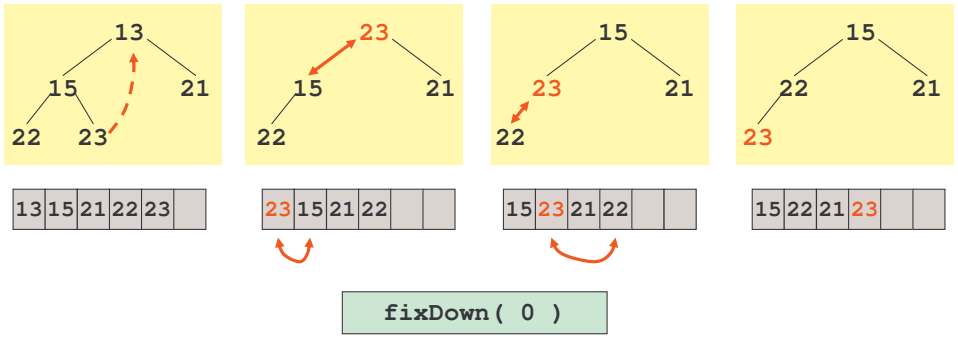
Integer implements Comparable

```
public class Integer implements Comparable {
  private int value;

  public Integer(int v) {
    this.value = v;
  }
  public boolean equals(Object that) {
    if (!(that instanceof Integer)) return false;
    return this.value == ((Integer)that).value;
  }
  public int compareTo(Object that) {
    int v = ((Integer)that).value;
    if (this.value == v) return 0;
    return (this.value < v) ? -1 : 1;
  }
  ...
}
```

หมายเหตุ : `Integer` ของจริงในจาวา ซับซ้อนกว่านี้

BinaryHeap : dequeue

- dequeue( )
  - เก็บรากไว้เป็นคำตอบ
  - ย้ายข้อมูลที่ใบล่างขวาสุด มาเก็บที่ราก
  - ดันข้อมูลที่รากลงมาตามทาง จนกว่าพ่อจะไม่มากกว่าลูก



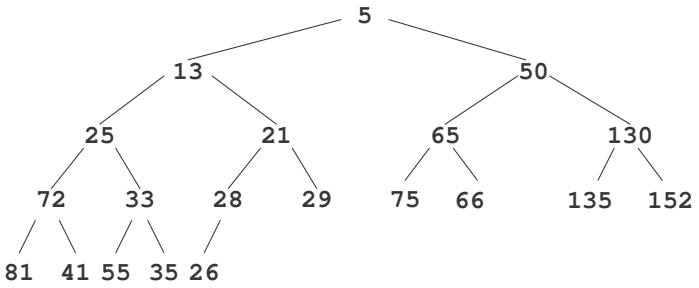
BinaryHeap : dequeue

```
public class BinaryHeap implements PriorityQueue {

  private Object[] elementData;
  private int size;

  public Object dequeue() {
    Object result = peek();
    elementData[0] = elementData[--size];
    elementData[size] = null;
    if (size > 1) fixDown(0); // next slide
    return result;
  }
  ...
```

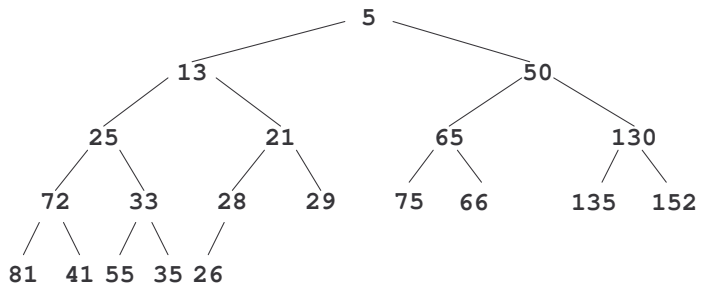

BinaryHeap : fixDown

```
public class BinaryHeap implements PriorityQueue {
  ...
  private void fixDown(int k) {
    int c;
    Comparable tmp = (Comparable) elementData[k];
    while ((c = 2 * k + 1) < size && (c > 0)) {
      if (c < size-1) {
        Comparable left = (Comparable) elementData[c];
        if (left.compareTo(elementData[c+1]) > 0) c++;
      }
      if (tmp.compareTo(elementData[c]) <= 0) break;
      elementData[k] = elementData[c];
      k = c;
    }
    elementData[k] = tmp;
  }
  ...
}
```


ทวนอีกครั้ง : enqueue(6)



ทวนอีกครั้ง : dequeue

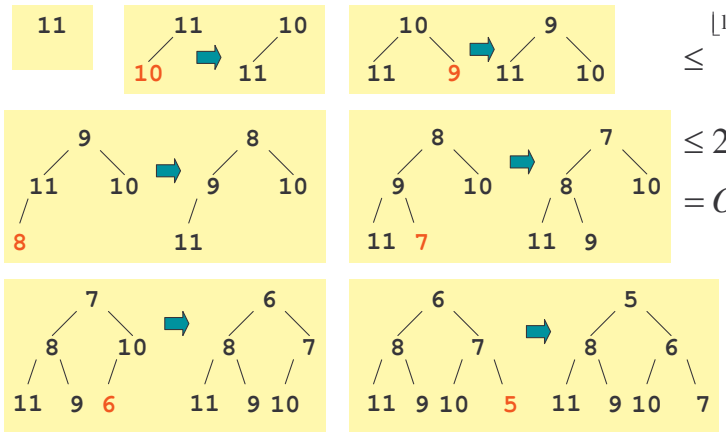


เวลาการทำงาน

- มีข้อมูล n ตัว, balanced binary tree สูง $\lfloor \log_2 n \rfloor$
- peek : O(1)
- enqueue : fixUp = O(h) = O(log n)
- dequeue : fixDown = O(h) = O(log n)

การสร้าง binary heap ให้มีข้อมูล n ตัว

- ใช้การ enqueue ข้อมูลทีละตัว
- จำนวนการดันข้อมูลทั้งหมดใน fixUp เท่ากับ
  - 0 + (1+1) + (2+2+2+2) + ... = 0 + 1x2 + 2x4 + 3x8 + ...



$$\le \sum_{k=0}^{\lfloor \log_2 n \rfloor} k2^k$$

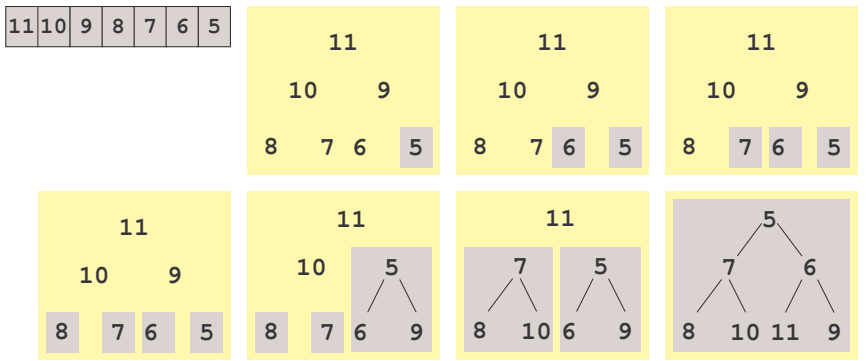
$$\le 2\left(n\log_2 n - n + 1\right)$$

$$= O(n\log n)$$

การสร้าง binary heap ให้มีข้อมูล n ตัว

- อีกวิธี เร็วกว่า for(int k=size-1; k>=0; k--) fixDown(k)
  - มีข้อมูล $n=2^{h+1}-1$ ตัว ต้นไม้สูง h , มี #ดันทั้งหมดใน fixDown
  - $\le 0x2^h + 1x2^{h-1} + 2x2^{h-2} + \ldots + h \times 1$

$$= \sum_{k=0}^{h} k2^{h-k} \le 2^h \sum_{k=0}^{\infty} k2^{-k} = 2n = O(n)$$



การบ้าน : buildHeap

- ในการสร้าง heap จากอาเรย์ เราสามารถเปลี่ยนการ fixDown เริ่มจาก size-1 เป็นเริ่มจาก size/2-1 ก็ได้ ทำไม ?

```
public class BinaryHeap implements PriorityQueue {
  private Object[] elementData;
  private int size;

  public BinaryHeap() {}
  public BinaryHeap(Object [] data) {
    elementData = data.clone();
    size = elementData.length;
    for(int k = size-1; k >= 0; k--) {
      fixDown(k);
    }
  }
  ...
}
```

```
for(int k = size/2-1; k >= 0; k--)
```


ตัวอย่างการใช้ Binary Heap : Heap Sort

```
public class BinaryHeap implements PriorityQueue {
  private Object[] elementData;
  private int size;

  public static void heapSort(Object[] data) {
    BinaryHeap h = new BinaryHeap();
    h.elementData = data;
    h.size = data.length;
    for(int k=size/2-1; k>=0; k--) fixDown(k);
    for(int k=size-1; k>0; k--) {
      elementData[k] = h.dequeue();
    }
  }
  ...
}
```

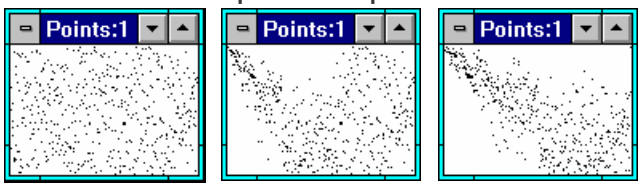
O(n)

O(n log n)

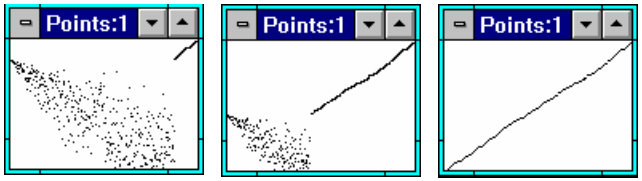
เรียงจากมากไปน้อย

Heap Sort

- ที่ผ่านมา : พ่อมีค่าน้อยกว่าลูก ๆ : min-Heap
- ให้พ่อมีค่ามากกว่าลูก ๆ : max-Heap
  - ต้องใช้ max-heap ให้ heap sort เรียงจากน้อยไปมาก

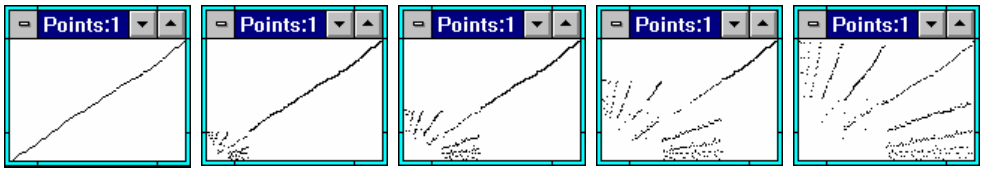


กำลังสร้าง max-heap

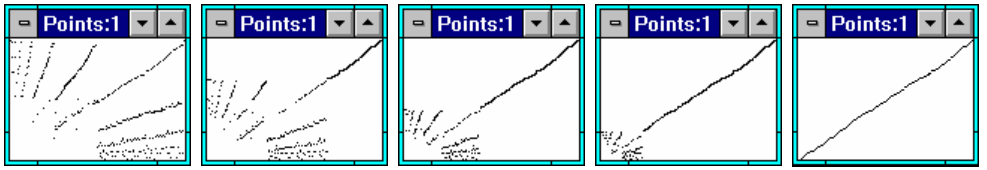


กำลัง dequeue

Heap Sort : เรียงข้อมูลที่เรียงอยู่แล้ว



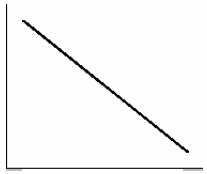
กำลังสร้าง max-heap



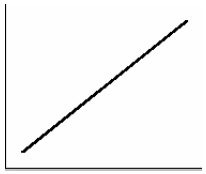
กำลัง dequeue

Heap Sort Animations

random



inverse



sorted