Data Structures

Heaps

Teacher : Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. 殷人昆, 数据结构
3. 金远平, 数据结构
4. http://inside.mines.edu/~dmehta/

Priority Queues

- At any time, an element with arbitrary priority, such as highest or lowest, can be inserted into or removed from the queue
- priority queues is as an unordered linear list
- Heaps are frequently used to implement priority queues

- **Two kinds :**

Min priority queue

```
//最小优先级队列类的定义
template <class E>
class MinPQ
{
  public:
     Virtual bool Insert (E& d) = 0;
     Virtual bool Remove (E& d) = 0;
};
```

Max priority queue

```
//最小优先级队列类的定义
template <class E>
class MaxPQ
{
  public:
     Virtual bool Insert (E& d) = 0;
     Virtual bool Remove (E& d) = 0;
};
```

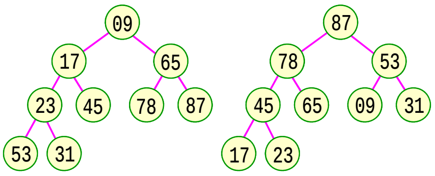
Definition of a Heap

- A **complete** binary tree
- A **min** tree

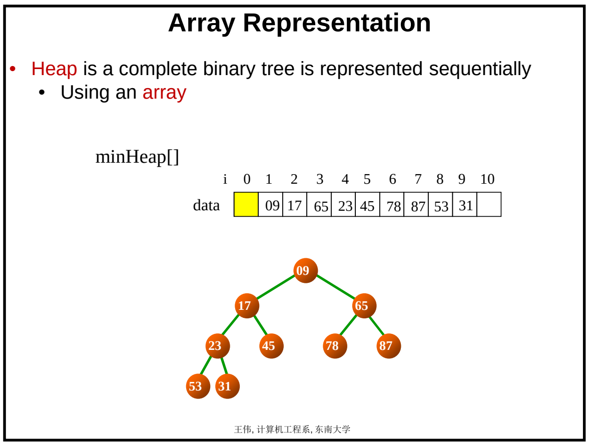
$K_i \le K_{2i+1}$ && $K_i \le K_{2i+2}$

- A **complete** binary tree
- A **max** tree

$K_i \ge K_{2i+1}$ && $K_i \ge K_{2i+2}$



Min heap **Max heap**

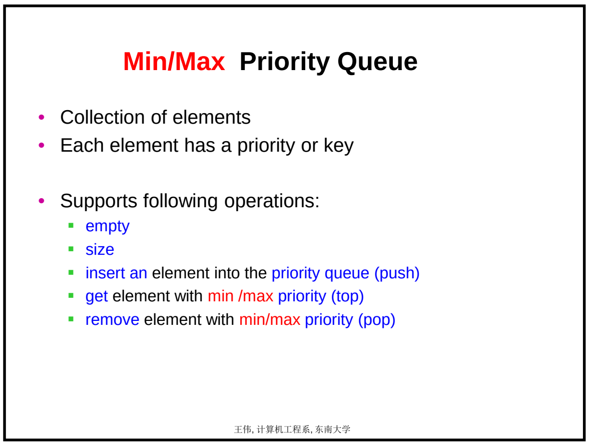


Array Representation

- Heap is a complete binary tree is represented sequentially
  - Using an array

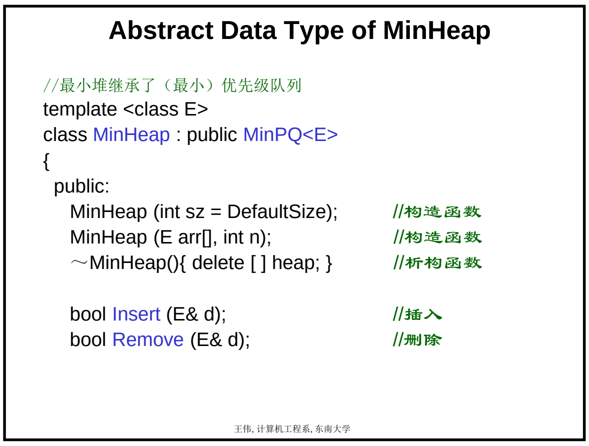
minHeap[]

i	0	1	2	3	4	5	6	7	8	9	10
data	🟨	09	17	65	23	45	78	87	53	31	



Min/Max Priority Queue

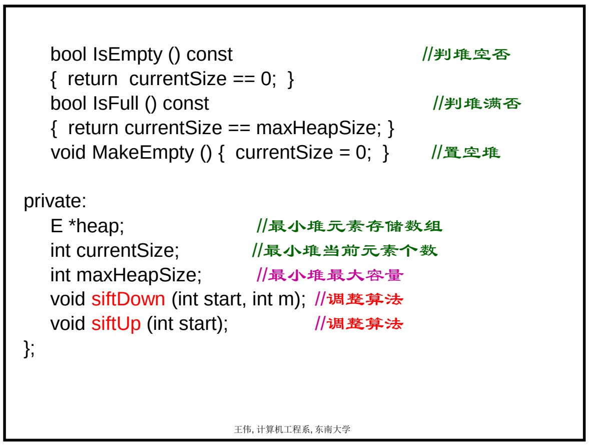
- Collection of elements
- Each element has a priority or key
- Supports following operations:
  - empty
  - size
  - insert an element into the priority queue (push)
  - get element with min /max priority (top)
  - remove element with min/max priority (pop)



Abstract Data Type of MinHeap

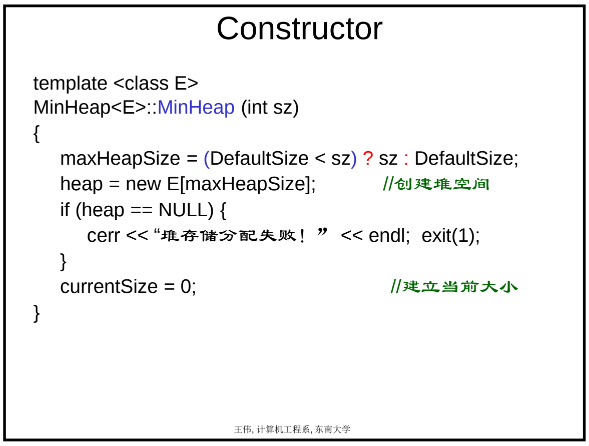
```
//最小堆继承了（最小）优先级队列
template <class E>
class MinHeap : public MinPQ<E>
{
  public:
     MinHeap (int sz = DefaultSize);        //构造函数
     MinHeap (E arr[], int n);              //构造函数
     ～MinHeap(){ delete [ ] heap; }        //析构函数

     bool Insert (E& d);                    //插入
     bool Remove (E& d);                    //删除
```



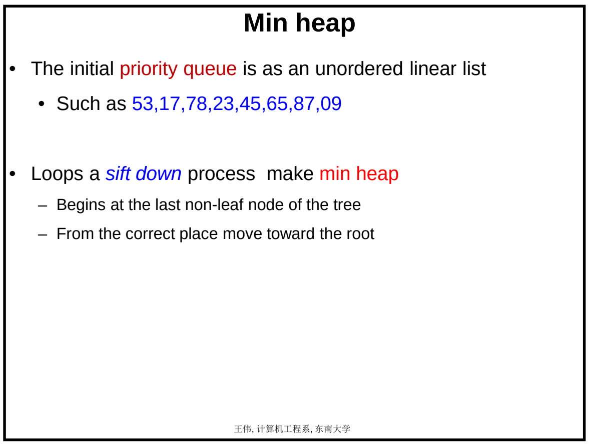
```
    bool IsEmpty () const                         //判堆空否
    {  return  currentSize == 0;  }
    bool IsFull () const                           //判堆满否
    {  return currentSize == maxHeapSize; }
    void MakeEmpty () {  currentSize = 0;  }       //置空堆

private:
    E *heap;                    //最小堆元素存储数组
    int currentSize;           //最小堆当前元素个数
    int maxHeapSize;            //最小堆最大容量
    void siftDown (int start, int m);  //调整算法
    void siftUp (int start);           //调整算法
};
```



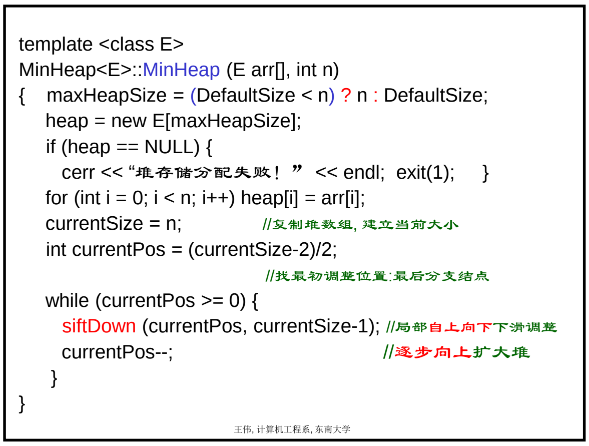
Constructor

```
template <class E>
MinHeap<E>::MinHeap (int sz)
{
    maxHeapSize = (DefaultSize < sz) ? sz : DefaultSize;
    heap = new E[maxHeapSize];          //创建堆空间
    if (heap == NULL) {
        cerr << “堆存储分配失败！” << endl;  exit(1);
    }
    currentSize = 0;                      //建立当前大小
}
```



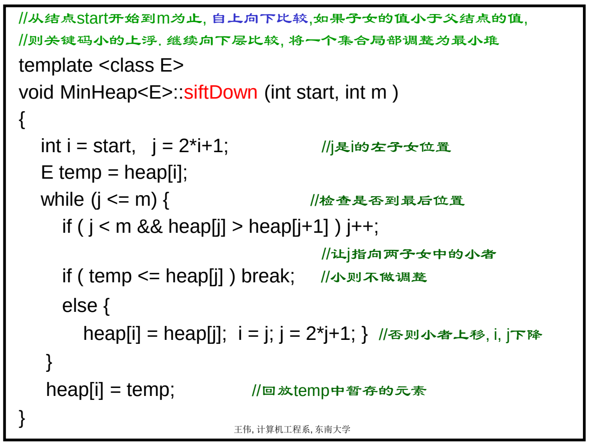
Min heap

- The initial priority queue is as an unordered linear list
  - Such as 53,17,78,23,45,65,87,09
- Loops a *sift down* process make min heap
  - Begins at the last non-leaf node of the tree
  - From the correct place move toward the root



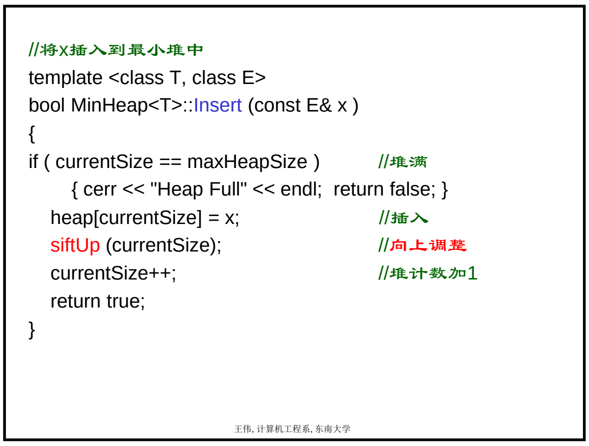
```
template <class E>
MinHeap<E>::MinHeap (E arr[], int n)
{    maxHeapSize = (DefaultSize < n) ? n : DefaultSize;
     heap = new E[maxHeapSize];
     if (heap == NULL) {
        cerr << “堆存储分配失败！ ” << endl;  exit(1);     }
     for (int i = 0; i < n; i++) heap[i] = arr[i];
     currentSize = n;               //复制堆数组, 建立当前大小
     int currentPos = (currentSize-2)/2;
                                    //找最初调整位置:最后分支结点
     while (currentPos >= 0) {
        siftDown (currentPos, currentSize-1); //局部自上向下下滑调整
        currentPos--;                         //逐步向上扩大堆
      }
}
```

王伟, 计算机工程系, 东南大学



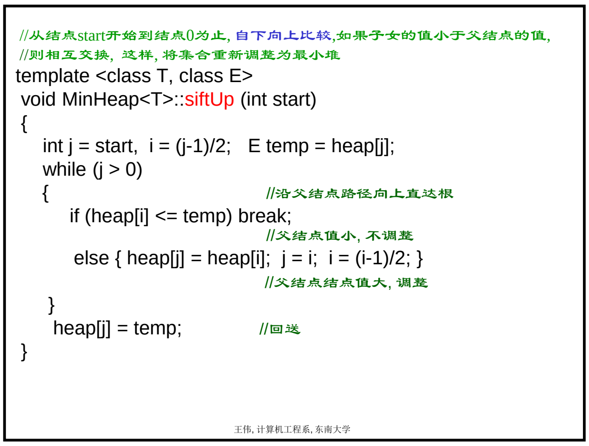
```
//从结点start开始到m为止, 自上向下比较,如果子女的值小于父结点的值,
//则关键码小的上浮. 继续向下层比较, 将一个集合局部调整为最小堆
template <class E>
void MinHeap<E>::siftDown (int start, int m )
{
    int i = start,   j = 2*i+1;              //j是i的左子女位置
    E temp = heap[i];
    while (j <= m) {                         //检查是否到最后位置
       if ( j < m && heap[j] > heap[j+1] ) j++;
                                             //让j指向两子女中的小者
       if ( temp <= heap[j] ) break;         //小则不做调整
       else {
          heap[i] = heap[j];  i = j; j = 2*j+1; }  //否则小者上移, i, j下降
     }
     heap[i] = temp;              //回放temp中暂存的元素
}
```

王伟, 计算机工程系, 东南大学

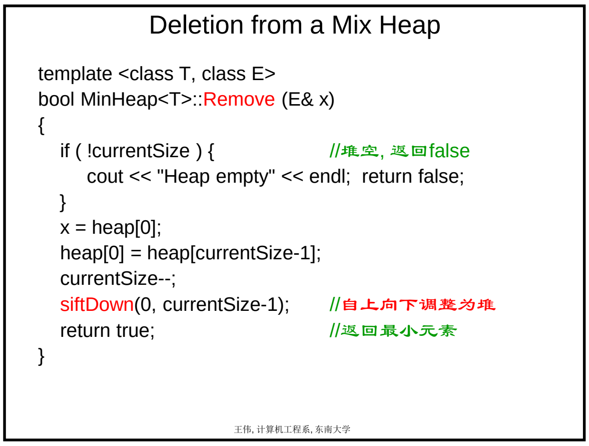


```
//将x插入到最小堆中
template <class T, class E>
bool MinHeap<T>::Insert (const E& x )
{
if ( currentSize == maxHeapSize )          //堆满
      { cerr << "Heap Full" << endl;  return false; }
   heap[currentSize] = x;                  //插入
   siftUp (currentSize);                   //向上调整
   currentSize++;                          //堆计数加1
   return true;
}
```

王伟, 计算机工程系, 东南大学



```
//从结点start开始到结点0为止, 自下向上比较,如果子女的值小于父结点的值,
//则相互交换, 这样, 将集合重新调整为最小堆
template <class T, class E>
 void MinHeap<T>::siftUp (int start)
 {
     int j = start,  i = (j-1)/2;   E temp = heap[j];
     while (j > 0)
     {                                  //沿父结点路径向上直达根
         if (heap[i] <= temp) break;
                                        //父结点值小, 不调整
          else { heap[j] = heap[i];  j = i;  i = (i-1)/2; }
                                        //父结点结点值大, 调整
      }
      heap[j] = temp;                  //回送
 }
```



Deletion from a Mix Heap

```
template <class T, class E>
bool MinHeap<T>::Remove (E& x)
{
    if ( !currentSize ) {              //堆空, 返回false
        cout << "Heap empty" << endl;  return false;
    }
    x = heap[0];
    heap[0] = heap[currentSize-1];
    currentSize--;
    siftDown(0, currentSize-1);        //自上向下调整为堆
    return true;                       //返回最小元素
}
```