特征

和入队顺序无关，总是最大的元素优先出队。
如果说栈是每次输出最先进入的元素，队列是每次输出最后进入的元素，那么优先队列就是每次输出优先级最大的元素。

API

优先队列是一种抽象数据类型，他表示了一组值和对这些值的一些操作。我们不一定要用某种固定的存储和操作方式来实现它，只要满足它的特征那它就是优先队列。但怎样才能高效实现它呢？先列出一份API：

public class MAXPQ	用途
MaxPQ()	创建一个优先队列
MaxPQ(int max)	创建一个初始容量为max的优先队列
MaxPQ(T[] arr)	用arr[]中的元素创建一个优先队列
void insert(T a)	向队列中插入一个元素
T max()	返回最大元素
T delMax()	删除并返回最大元素
boolean isEmpty()	返回队列是否为空
int size()	返回优先队列中的元素个数

实现逻辑

数据结构二叉堆就能很高效的实现这份API。
（堆：当一棵二叉树的某个节点都大于等于它的两个子节点时，它被称为堆有序。）
当有新元素入队时，就把它放入堆的末尾，然后让他自由上浮，直到浮到堆顶或不大于父节点的位置。

```
//添加一个元素
public void insert(T a){
    maxPQ.add(a);
    swim(maxPQ.size()-1);
}
//上浮
private void swim(int idx){
    int fidx = (idx-1)/2;
    if(idx==0||maxPQ.get(fidx).compareTo(maxPQ.get(idx)) >= 0)return;
    Collections.swap(maxPQ,fidx,idx);
    swim(fidx);
}
123456789101112
```

当要删除最大的元素时，就把堆顶元素（堆顶就是最大元素）和堆末尾元素交换位置，然后删除并返回末尾元素，最后还需要将堆顶元素自由下沉到堆低或不小于子节点的位置。

```
```

```
//删除并返回最大元素
public T delmax(){
    if(maxPQ.isEmpty())return null;
    Collections.swap(maxPQ,0,maxPQ.size()-1);
    T max = maxPQ.get(maxPQ.size()-1);
    maxPQ.remove(maxPQ.size()-1);
    if(maxPQ.isEmpty()==false)sink(0);
    return max;
}
//下沉
private void sink(int idx){
    int lastIdx = idx*2+1;
    if(idx<0||idx>=maxPQ.size()||lastIdx>=maxPQ.size())return;
    //如果有右节点且右节点大于左节点

 if(lastIdx<maxPQ.size()-1&&maxPQ.get(lastIdx).compareTo(maxPQ.get(lastIdx+1))
<0)lastIdx++;
    if(maxPQ.get(idx).compareTo(maxPQ.get(lastIdx))>=0)return;       //如果大于所有
子节点就不下沉了
    Collections.swap(maxPQ,idx,lastIdx);
    sink(lastIdx);
}
12345678910111213141516171819
```

完整实现代码

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
//用堆实现优先队列
public class MaxPQUseArray<T extends Comparable>{
    private ArrayList<T> maxPQ;
    MaxPQUseArray(){maxPQ = new ArrayList<>();}
    //创建一个初始容量为max的优先队列
    MaxPQUseArray(int max){maxPQ = new ArrayList<>();maxPQ.ensureCapacity(max);}
    //用数组初始化优先队列
    MaxPQUseArray(T[] arr){
        maxPQ = new ArrayList<>();
        for(T t:arr){
            insert(t);
        }
    }
    //添加一个元素
    public void insert(T a){
        maxPQ.add(a);
        swim(maxPQ.size()-1);
    }
    //返回最大元素
    public T max(){
        return maxPQ.isEmpty()?null:maxPQ.get(0);
    }
    //删除并返回最大元素
    public T delmax(){
        if(maxPQ.isEmpty())return null;
        Collections.swap(maxPQ,0,maxPQ.size()-1);
        T max = maxPQ.get(maxPQ.size()-1);
```

```
        maxPQ.remove(maxPQ.size()-1);
        if(maxPQ.isEmpty()==false)sink(0);
        return max;
    }
    //上浮
    private void swim(int idx){
        int fidx = (idx-1)/2;
        if(idx==0||maxPQ.get(fidx).compareTo(maxPQ.get(idx)) >= 0)return;
        Collections.swap(maxPQ,fidx,idx);
        swim(fidx);
    }
    //下沉
    private void sink(int idx){
        int lastIdx = idx*2+1;
        if(idx<0||idx>=maxPQ.size()||lastIdx>=maxPQ.size())return;
        //如果有右节点且右节点大于左节点

 if(lastIdx<maxPQ.size()-1&&maxPQ.get(lastIdx).compareTo(maxPQ.get(lastIdx+1))
<0)lastIdx++;
        if(maxPQ.get(idx).compareTo(maxPQ.get(lastIdx))>=0)return;       //如果大于
所有子节点就不下沉了
        Collections.swap(maxPQ,idx,lastIdx);
        sink(lastIdx);
    }
    public boolean isEmpty(){
        return maxPQ.isEmpty();
    }
}
12345678910111213141516171819202122232425262728293031323334353637383940414243444
5464748495051525355455565758
```

测试用例

当用一个乱序的数组初始化优先队列，然后再依次输出，输出的元素顺序就是有序的了,这也就是著名的堆排序过程。

```
public static void main(String[] args) {
    Integer[] arr = {2,9,5,1,0,3,6,8,4,7};
    MaxPQUseArray<Integer> maxPQUseArray = new MaxPQUseArray<Integer>(arr);
    while(maxPQUseArray.isEmpty()==false){
        System.out.print(maxPQUseArray.delmax()+" ");
    }
}
1234567
```

输出结果

9 8 7 6 5 4 3 2 1 0

当然在大部分语言的标准库中也有写好的优先队列。

c++API

cppreference.com -> C++ Priority Queues(优先队列)

C++ Priority Queues(优先队列)

C++优先队列类似队列，但是在这个数据结构中的元素按照一定的断言排列有序。

empty()	如果优先队列为空，则返回真
pop()	删除第一个元素
push()	加入一个元素
size()	返回优先队列中拥有的元素的个数
top()	返回优先队列中有最高优先级的元素

测试样例

```
#include<iostream>
#include<queue>
using namespace std;
int main(){
    int arr[] = {2,6,5,4,1,7,9,8,3};
    priority_queue<int> pque(begin(arr),end(arr));
    pque.push(0);
    cout << "size:" << pque.size() << endl;
    while(!pque.empty()){
        int top = pque.top();
        pque.pop();
        cout << top << " ";
    }

    return 0;
}
123456789101112131415161718
```

结果

size:10
9 8 7 6 5 4 3 2 1 0