

# 12-A2

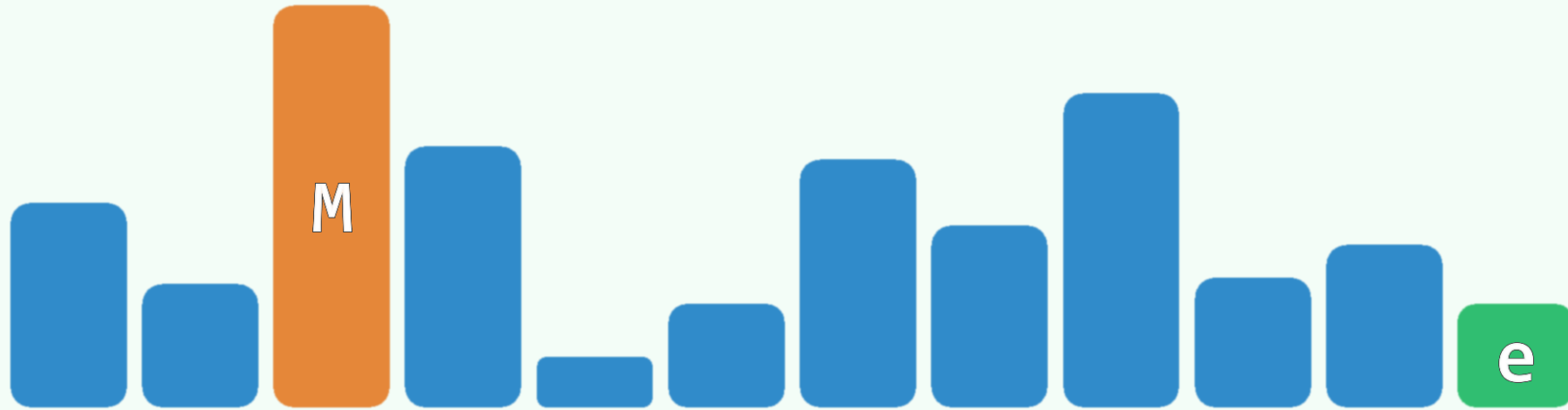
优先级队列

概述：基本实现

邓俊辉

deng@tsinghua.edu.cn

# Vector



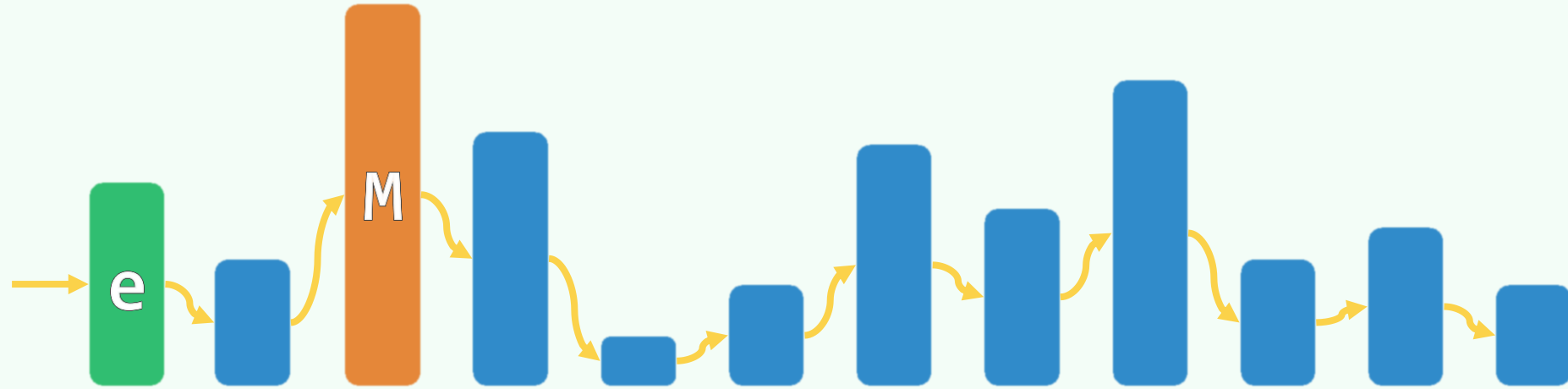
<code>getMax()</code>	<code>delMax()</code>	<code>insert()</code>
<code>traverse()</code> $\Theta(n)$	<code>remove( traverse() )</code> $\Theta(n) + \mathcal{O}(n) = \Theta(n)$	<code>insertAsLast(e)</code> $\mathcal{O}(1)$

# Sorted Vector



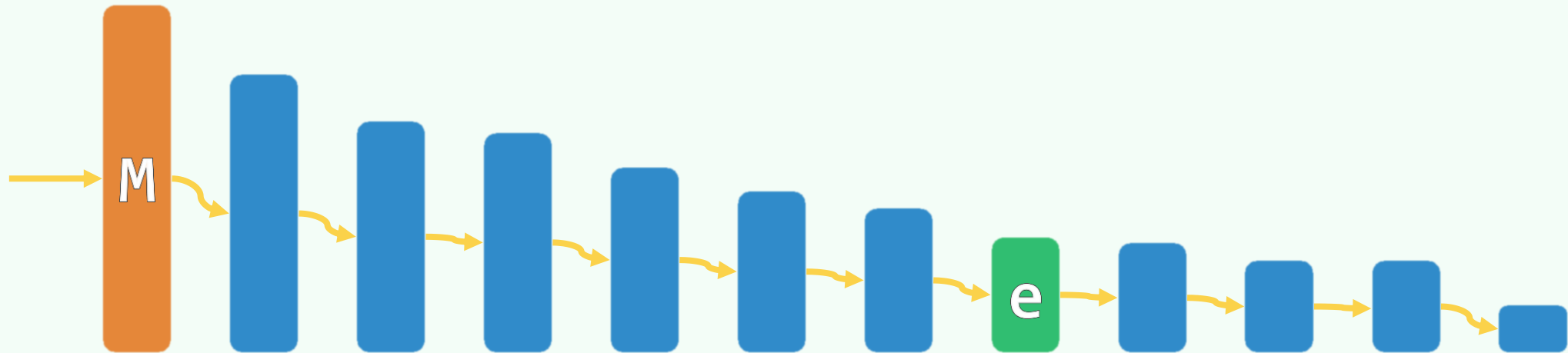
<code>getMax()</code>	<code>delMax()</code>	<code>insert()</code>
<code>[n - 1]</code> $O(1)$	<code>remove(n - 1)</code> $O(1)$	<code>insert( 1 + search(e), e )</code> $O(\log n) + O(n) = O(n)$

# List



getMax()	delMax()	insert()
traverse() $\Theta(n)$	remove( traverse() ) $\Theta(n) + \mathcal{O}(1) = \Theta(n)$	insertAsFirst(e) $\mathcal{O}(1)$

## Sorted List



<code>getMax()</code>	<code>delMax()</code>	<code>insert()</code>
<code>first()</code> $O(1)$	<code>remove( first() )</code> $O(1)$	<code>insertA( search(e), e )</code> $O(n) + O(1) = O(n)$

# BBST

❖ AVL、Splay、Red-black：三个接口均只需 $\mathcal{O}(\log n)$ 时间

但是，BBST的功能远远超出了PQ的需求...

$$\text{❖ PQ} = \boxed{1 \times \text{insert}()} + \boxed{0.5 \times \text{search}()} + \boxed{0.5 \times \text{remove}()}$$

❖ 若只需查找极值元，则不必维护所有元素之间的全序关系，偏序足矣

❖ 因此有理由相信，存在某种更为简单、维护成本更低的实现方式

使得各功能接口的时间复杂度依然为 $\mathcal{O}(\log n)$ ，而且实际效率更高

❖ 当然，就最坏情况而言，这类实现方式已属最优——为什么？

# 统一测试

```
❖ template <typename PQ, typename T> void testHeap( int n ) {  
    T* A = new T[ 2 * n / 3 ]; //创建容量为 $2n/3$ 的数组，并  
    for ( int i = 0; i < 2 * n / 3; i++ ) A[i] = dice( (T) 3 * n ); //随机化  
    PQ heap( A + n / 6, n / 3 ); delete [] A; //Robert Floyd  
    while ( heap.size() < n ) //随机测试  
        if ( dice( 100 ) < 70 ) heap.insert( dice( (T) 3 * n ) ); //70%概率插入  
        else if ( ! heap.empty() ) heap.delMax(); //30%概率删除  
    while ( ! heap.empty() ) heap.delMax(); //清空  
}
```