12 changeMax

删除最大元素并为新元素寻找位置。

template<class T>

void maxHeap<T>::changeMax(T newElement) {

if (heapSize == 0)

throw queueEmpty();

//删除旧的最大元素

heap[1].~T();

//从根部开始为新元素寻找插入位置

int currentNode = 1,

child = 2;

while (child <= heapSize) {

if (child < heapSize && heap[child] < heap[child + 1])

child++;

if (newElement >= heap[child])

break;

heap[currentNode] = heap[child];

currentNode = child;

child \*= 2;

}

heap[currentNode] = newElement;

}

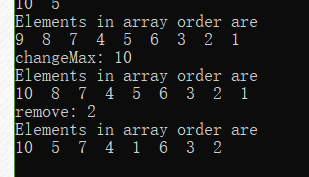
复杂度证明：

child每次循环乘二，在循环x次后，必有2x >= heapSize

也即x=log2(heapSize)

故时间复杂度为O(logn)得证

实际运行正确：



13 remove

删除目标元素并为尾部元素寻找新位置。

template<class T>

int maxHeap<T>::remove(int i) {

if (heapSize == 0)

throw queueEmpty();

int temp = heap[i];

heap[i].~T();

T lastElement = heap[heapSize--];

int child = i\*2;

while (child <= heapSize) {

if (heap[child] < heap[child + 1])

child++;

if (lastElement >= heap[child])

break;

heap[i] = heap[child];

i = child;

child \*= 2;

}

heap[i] = lastElement;

return temp;

}

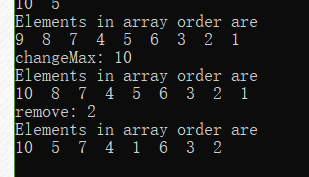
复杂度证明同上题：

child每次循环乘二，在循环x次后，必有2x >= heapSize

也即x=log2(heapSize)

故时间复杂度为O(logn)得证

实际运行正确：

+