阅读下列说明和伪代码，回答问题 1 至问题3，将解答写在答题纸的对应栏内。

【说明】

堆数据结构定义如下：

对于n个元素的关键字序列{a1 , a2 , ..., an } ，当且仅当满足

其中， 在一个堆中，若堆顶元素为最大元素，则称为大顶堆；若堆顶元素为最小元素， 则称为小顶堆。堆常用完全二叉树表示，下图是一个大顶堆的例子。堆有两个属性：

表示存储空间大小的capacity和表示对元素个数的size。

堆数据结构常用于优先队列中，以维护由一组元素构成的集合。对应于两类堆结构，优先队列也有最大优先队列和最小优先队列，其中最大优先队列采用大顶堆，最小优先队列采用小顶堆。以下考虑最大优先队列。

假设现已建好大顶堆A，且已经实现了调整堆的函数HEAPIFY(A, n, index)。

下面将伪代码中需要完善的三个函数说明如下：

（1）heapMaximum(ARRAY \*A)：返回大顶堆A中的最大元素。

（2）heapExtractMax(Array \*A)：去掉并返回大顶堆A的最大元素，将最后一个元素“提前”到堆顶位置，并将剩余元素调整成大顶堆。

（3）maxHeapInsert（ARRAY \*A,int key）：把元素key插入到大顶堆A的最后位置，再将A调整成大顶堆。

优先队列采用顺序存储方式，其存储结构定义如下：

#define PARENT(i) i/2

typedef struct array{

int \*int\_array;//优先队列的存储空间首地址

int array\_size;//优先队列的长度

int capacity;//优先队列存储空间的容量

}

【C代码】

(1)函数heapMaximum

int heapMaximum(ARRAY \*A){return \_\_\_\_(1)\_\_\_\_}

(2)函数heapExtractMax

int heapExtractMax(Array \*A){

int max;

max=A->int\_array[0];

\_\_\_\_(2)\_\_\_\_

A->array\_size--;

Heapify(A,A->array\_size,0);

return max

}

(3)函数maxHeapInsert

int maxHeapInsert（ARRAY \*A,int key）{

int i,\*p;

if(A->array-size==A->capacity){//存储空间的容量不够时扩充空间

p=(int\*)realloc(A->int\_array,A->capacity\*2\*sizeof(int));

if(!p) return -1;

A->int\_array=p;

A->capacity=2\*A->capacity;

}

A->array\_size++;

i=\_\_\_(3)\_\_\_\_;

while(i>0&&\_\_(4)\_\_\_){

A->int\_array[i]=A->int\_array[PARENT(i)];

i=PARENT(i);

}

\_\_\_(5)\_\_

return 0;

}

【问题 1】

根据以上说明和伪代码，填充伪代码中的空（1）～（5）。

【问题2】

根据以上伪代码，过程heapMaximum、heapExtractMaxmax、HeapInsert

的时间复杂度的紧致上界分别为（6）、（7）和（8）（用O符号表示）。

【问题3】 若将元素10 插入到堆 A =15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1〉中，调用maxHeapInsert过程进行操作，则新插入的元素在堆 A 中第（9）个位置（从1开始）。

答案

问题1：（1）A->int\_array[0]

(2)A->int\_array[0]=A->int\_array[A->array\_size-1]

(3)A->array\_size-1

(4)key>A->int\_array[PARENT(i)]

(5)A->int\_array[i]=key

问题2

（6）O(1) （7）O(log2n) （8）O(log2n)

问题3

（9）3