DSA_11

姓名：邵宁录　　　学号：2018202195

17-2

a.

1. 算法设计：

由于单个列表内是有序的，但列表之间不存在特定的大小关系。

那么 **SEARCH** 操作只能遍历的二分查找每个列表。

2. 最坏运行时间：

首先，对于一个长度为 n 的列表，二分查找的最坏运行时间为 $\mathrm{O}(\lg n)$ 。

由于每个列表的长度为 2^i ，所以对每个列表来说，最坏运行时间为 $\mathrm{O}(i)$ 。

又最多会有 $\lceil\lg(n+1)\rceil$ 个列表，即 $\mathrm{O}(\lg n)$ 个列表。

那么 $i=1\ to\ \mathrm{O}(\lg n)$ ，相加得 $\mathrm{O}((\lg n)^2)$ ，即最坏情况时间复杂度为 $\mathrm{O}((\lg n)^2)$

b.

1. 算法设计

由于每个列表的长度都为 2^i ，那么在加入新元素的时候，不妨记这个元素为 x：

1. 若 A_0 为空，则加入 A_0
2. 若 A_0 不为空，那么将 x 与 A_0 中的元素合并一起加入 A_1
3. 若 A_1 不为空，那么将 x 与 A_0, A_1 中的元素合并一起加入 A_2
4. 以此类推

2. 最坏运行时间：

首先需要明确最坏情况下的元素插入是什么样的：列表 $A_0, A_1, \ldots, A_{k-1}$ 都是满的，这时加入了新元素 x 。

那么此时需要新增一个列表 A_k ，并将全部的元素合并到 A_k 中去。

我们已经知道合并两个有序列表的时间复杂度为 $\mathrm{O}(m+n)$ 。那么对于上述的最坏情况，合并所需要的时间复杂度为 $\mathrm{O}(1+2+4+\cdots+2^{k-1})$ ，即 $\mathrm{O}(2^k)$ 。

其中， $k=\lceil\lg(n+1)\rceil$ ，那么上式又可以写为 $\mathrm{O}(n)$ ，即最坏情况时间复杂度为 $\mathrm{O}(n)$

3. 摊还分析：

其每步的摊还代价为 $\mathrm{O}(\lg n)$ 。

下面用核算法进行分析：

设对一个由n个操作组成的操作序列，其每步的操作为摊还代价为 k ，其中 $k=\lceil\lg(n+1)\rceil$ 。

我们知道，最多会有 $A_0, A_1, \ldots, A_{k-1}$ 这 k 个列表。

那么对于一个单独的元素 x ，它最多会从第 0 个列表 A_0 被移动到 第 $k-1$ 个列表 A_k-1 。由于每个列表内部都是有序的，所以这样的移动总共最多会支付 k 点实际代价。

所以对这么一个单独的元素，插入时的已经支付了它以后所有操作的代价，所以总的代价和大于0，因此满足核算法的条件。

综上，其每步的摊还代价为 k 。即 其每步的摊还代价为 $\mathrm{O}(\lg n)$ 。

c.

1. 算法设计：

首先我们讨论找到被删除数所需花的时间代价：

1. 如果 **DELETE** 操作给定的参数是被删除数的 **index** ，那么数组可以在 $\mathrm{O}(1)$ 的时间内找到该数。
2. 如果 **DELETE** 操作给定的参数是被删除数的 **value** ，那么可以在 $\mathrm{O}(\lg n)$ 的时间内找到该数。

下面讨论找到该数并删除后，如何重新维护该多个有序数组：

1. 记 n 的二进制表示为 $<n_{k-1}, n_{k-2}, \ldots, n_0>$ 。设 n_m 为下标最小的那个值为1的 n_i 。
2. 将被删除数和 A_m 中的第一个数互换（可以是任意一个数，方便起见取第一个数）。
3. 此时我们可以看到，需要重新构建的列表只有 A_m （因为 A_m 中的一个数被删掉了，列表的长度为 2^m-1。 又由于它是有序的，所以我们只需对其做分割操作，将里面的数放入 $A_0, A_1, \ldots, A_{m-1}$ 中即可。
4. 删除操作至此结束。

我们可以看到，第 1 步的时间复杂度为 $\mathrm{O}(\lg n)$ ，因为总共只有这个数量的列表。第 2 步的时间复杂度为 $\mathrm{O}(1)$ 。第 3 步的时间复杂度为 $\mathrm{O}(m)$ ，因为序列是有序的，只需做一些分割操作。

所以综上所述，整个 **DELETE** 操作的时间复杂度最多为 $\mathrm{O}(\lg n)$ 。