18-2 Making binary search dynamic
[Solution]

a. To perform search operation for this structure, we can just perform binary search for each of the sorted arrays till we find the data we are searching for.

   As we know, the time of the execution of the binary search algorithm on the array A_i is $O(\lg 2^i) = O(i)$. Therefore, the worst-case running time of the dynamic binary search is

$$\sum_{i=0}^{k-1} O(i) = O(\frac{1}{2}k(k-1)) = O(k^2) = O(\lg^2 n) \quad (\because k = \lceil \lg(n+1) \rceil)$$

b. To insert a new element into this data structure, we just insert a new array A_0 into the arrays. If another A_0 exists, we merge these two A_0 into one A_1. If another A_1 exists, we merge these two A_1 into one A_2. We going on and keep merging till we do not need merge any longer.

   The worst case running time is

$$\sum_{i=1}^{k} 2^k = O(n) \qquad (k = \lceil \lg(n+1) \rceil)$$

   For the amortized time, we can notice the binary representation of n, i.e., $\langle n_{k-1}, n_{k-2}, \cdots, n_0 \rangle$. When we perform a series of insertion, n_0 flips every time, n_1 flips every $2th$ time, …, n_{k-1} flips every $2^k th$ time. A flip indicates a merge operation. So the total running time for m insertions is

$$T \le \sum_{i=0}^{k-1} \left\lfloor \frac{m}{2^i} \right\rfloor 2^{i+1} \le 2mk = mO(\lg n)$$

   So the amortized running time for each operation is $mO(\lg n)/m = O(\lg n)$.

c. To delete an element, say x, in A_i, first we find the smallest array that is not empty. Assume it is A_j. Delete x from A_i. If $i \ne j$, take an element out of A_j and insert it into A_i. Whether $i \ne j$ or not, now the length of A_j is $2^j - 1$. Then we divide array A_j into j arrays, $A_0, A_1, \cdots, A_{j-1}$, which are consistent with this data structure.

5.22 [Solution]

Algorithm:

```
SELECTION(n, k)
  candidate ← Key[1]
  for i ← 2 to n − k + 1
      if candidate > Key[i]
          candidate ← Key[i]
  return candidate
```

Number of comparisons: $n - k$
Lower bound of comparisons: $n - k$