Average case analysis of binary search

1. A rudimentary (and incorrect) analysis of the average case

Given a sorted array of N elements, it is tempting to say that in average each element would takes (1+logN)/2 to be found successfully. However, this formula does not take into account the fact that each element in the array requires different number of iterations in the binary search before it is found.

Take the following array of 15 elements (Figure 1) as an example:

Index →	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Values →	5	15	25	35	45	55	65	75	85	95	105	115	125	135	145

Figure 1. A sorted array of 15 integers

As shown in the binarySearch( ) method definition (see http://users.cis.fiu.edu/~weiss/dsj4/code/BinarySearch.java), in each iteration the value of *mid* is updated (mid = (low + high) / 2) . The element at position 7, for example, always takes a single iteration to be found. On the other hand, data at position 14 takes 4 iterations in the binary search before being found. Figure 2 shows a binary tree that illustrates the four cases of successful binary search, each of which takes a different number of iterations. For example, elements at positions 1, 5, 9, and 13 each takes three iterations before being found, while elements at positions 3 and 11 each takes only two iterations to be found.

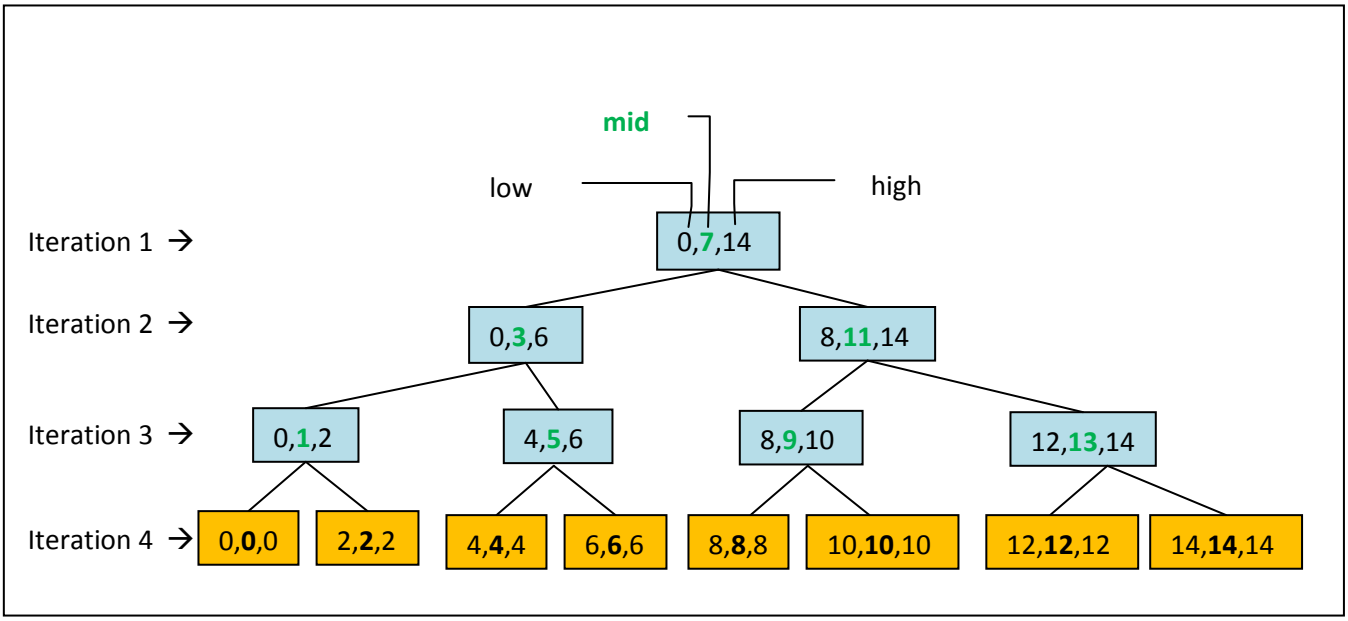

Figure 2. A binary tree showing the number of iterations for each element in an array of 15 to be found via binary search

Table 1 summarizes the array elements and the respective number of iterations for them to be found. The rightmost column shows the percentage of elements for each case. For instance, about 50% (8 out of 15) of the nodes take 4 iterations in the binary search before being found.

Number of iterations	Array elements	Percentage of nodes
1	A[7]	~6.25%
2	A[3], A[11]	~12.5%
3	A[1], A[5], A[9], A[13]	~25%
4	A[0], A[2], A[4], A[6], A[8], A[10], A[12], A[14]	~ 50%

Table 1. Array elements are divided into four cases, each with different number of iterations.

2. The correct analysis

To simplify the calculation, let N be equal to $2^k - 1$ (i.e., k ~= logN). The correct formula to calculate the average number of iterations for successful find is shown below.

$$\sum_{i=1}^{logN}(number\ of\ iterations\ in\ case\ i) * (number\ of\ nodes\ in\ case\ i) \ / \ N$$

$$= \sum_{i=1}^{logN} i * \frac{N}{2^i} \ / \ N$$

$$= (1*\frac{N}{2^{logN}} + \ldots + (logN-1)* \frac{N}{2^2} + logN*\frac{N}{2}) \ / \ N \qquad \textbf{<Sequence 1>}$$

Take the array of 15 elements as an example, the average cost is shown below:

$$\sum_{i=1}^{4}(number\ of\ iterations\ in\ case\ i) * (number\ of\ nodes\ in\ case\ i) \ / \ 15$$

$$= (4*8 + 3*4 + 2*2 + 1*1) \ / \ 15$$

$$\sim= 3.26 \text{ (or } \sim logN)$$

3. The conclusion

The average cost of a successful search is about the same as the worst case where an item is not found in the array, both being roughly equal to logN.

So, the average and the worst case cost of binary search, in big-O notation, is **O(logN)**.

Exercises:

1. Take an array of 31 elements. Generate a binary tree and a summary table similar to those in Figure 2 and Table 1.
2. Calculate the average cost of successful binary search in a sorted array of 31 elements.
3. Given an array of N elements, prove that calculation of Sequence 1 shown above is indeed O(logN).

Programming projects: