**Time Analysis of the Binary Search**

**Samuel L. Marateck**

We’ll start by reviewing big Oh notation

1. \(O(cN) = O(N)\) where \(N\) is the number of items considered, e.g., nodes of a tree, elements of an array. The letter \(c\) is a constant. So \(O(100N) = O(N)\).

2. \(O(N + c) = O(N)\). So \(O(N + 5) = O(N)\)

3. \(O(N/c) = O(N)\). So \(O(N/100) = O(N)\).

4. \(O(N^2 + N) = O(N^2)\) since the \(N^2\) term dominates.

5. \(O(c) = O(1)\).

In a linear search of an array in which the elements are sorted, we examine each element in sequential order until we find the element that corresponds to a key. If there are \(N\) elements in the array, on average we would find the key after \(N/2\) steps. By (3) the asymptotic behavior \(O(N/2)\) is \(O(N)\).

Let’s now analyze the binary search for an array in which the elements are sorted. Let’s say there are 16 elements in the array. We want to find the worst possible case in which we have to iterate the maximum number of times. After the first iteration of the loop, there are approximately 8 elements left. After the second iteration of the loop, there are approximately 4 elements left. After the third iteration of the loop, there are approximately 2 elements left. After the fourth iteration of the loop, there is 1 element left, and that is the key. This approximation of course improves as \(N\) approaches \(\infty\).

Now \(8 + 4 + 2 + 1\) is 15, one less than the original number of elements, 16. We know \(\sum_{n=0}^{k}2^n = 2^{k+1} - 1\). When \(k = 3\), this sum is 15. Therefor the number of elements, \(N\), in the array is \(2^{k+1}\). But \(k\) is the maximum number of iterations we must perform to find the key. Since \(N = 2^{k+1}\), let’s take the \(\log_2\) of both sides.

\[
\log_2(2^{k+1}) = \log_2(N) \\
\]

or

\[
k + 1 = \log_2(N) \\
\]

So the time analysis of a binary search is \(O(\log(N))\). This is the worst possible case, in other words the upper limit. Big Oh notation always gives the upper limit.

There are two points you should understand here.

- The sum of the number of elements we consider when we divide the array each time, approaches \(N\), the number of elements in the array, as the number of divisions and \(N\) both approach \(\infty\). Let’s look at a more rigorous proof: We consider \(N/2 + N/4 + N/8 + N/16 + N/32 + \ldots\) which equals \(N(1/2 + 1/4 + 1/8 + 1/16 + \ldots)\); but \(\sum_{n=1}^{\infty}1/2^n = 1\). So the sum equals \(N\) Can you prove this? See the next page.

- The number of times, \(k\) we divide the array is related to \(N\) by approximately \(N = 2^k\).
Let’s compare the binary search with a linear search for an array that has \( N = 2^{31} \) i.e., \( 2,147,483,647 \). Big Oh evaluation for the linear search is \( O(2,147,483,647) \). For the binary search it is \( O(\log_2 2^{31}) \) or 31.

Let’s prove \( \sum_{n=1}^{\infty} 1/2^n = 1 \). Start with

\[
S = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + ..
\]  

(3)

multiply (3) by \( 1/2 \) and get

\[
\frac{1}{2}S = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + ...
\]  

(4)

Subtract (4) from (3) and get

\[
S - \frac{1}{2}S = 1
\]  

(5)

So \( S = 2 \) and by (3) we get

\[
2 = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + ..
\]  

(6)

or

\[
1 = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + ..
\]  

(7)

So \( \sum_{n=1}^{\infty} 1/2^n = 1 \)