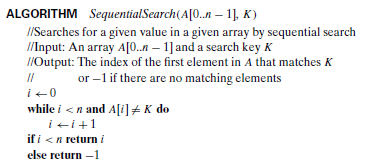
Sandeep Heera

Sequential Search vs Binary Search

**Sequential Search**

Worst case running time of sequential search will be = O(*n*) since in the worst case we’d be looking at every single element in the array. The algorithm for sequential search looks like this:



(p. 47 from Introduction to the Design and Analysis of Algorithms, Sality)

Since we want an actual running time and not a bound, we can perform a more thorough analysis. If each primitive operation takes 1 ns as specified in the problem, then this algorithm, in the worst case where the key *K* is not found, will take 1 ns to set *i*  to 0 initially, 1 ns to check whether *i < n* for each iteration of the while loop which will (this statement will execute *n +* 1 times), 1 ns to index into the array and 1 ns to compare the key *K* to the element at A[*i*] which will occur *n* times (since the first condition evaluates to false for the *n* + 1th iteration when *i = n* and thus the second half of the and statement will not be checked since it would lead to an index out of bounds exception),1 ns per *n* to increment *i* and 1 ns to set *i*, 1 ns to check if *i < n*, and 1 ns to return -1. Using summation notation:

**In the worst case, sequential search would require 5*n* + 4 ns.**

**Binary Search**

In the worst case for binary search, the amount of time it takes for the algorithm to execute is approximately 10 log *n* ns (from binary search analysis sheet on course web site). Below is a table for the required values of *n*:



To find values for which sequential search is faster than binary search, we can graph the two functions and find their point of intersection. The equations graphed were 10 log *n* and 5*n* + 4. The equations were graphed on a TI-Nspire and no points of intersection for *n >* 0 were found. This implies that **the sequential search algorithm is always slower than the binary search algorithm.**