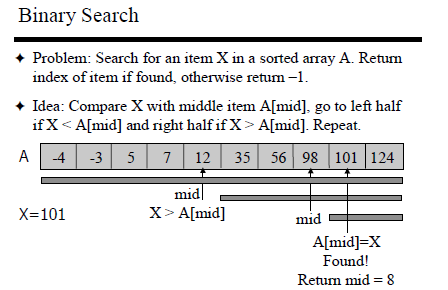
# Binary Search



Given a sorted array arr[] of n elements, write a function to search a given element x in arr[]. A simple approach is to do **linear search.** The time complexity of above algorithm is O(n). Another approach to perform the same task is using Binary Search.

**Binary Search:** Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Algorithm Binary\_search(A, low, high , key)

{

While ( low < = high) // when l>h, loop will stop and element is not found

{

mid= floor ( low + high /2 )

If (key==A[mid])

return mid;

else If ( key < A[mid])

high= mid -1

else

low= mid +1

}

return -1;

}

**Analyzing Binary Search**

**Best case - Ꝋ(1) comparisons**

In the best case, the item X is the middle in the array A. A constant number of comparisons (actually just 1) are required.

**Worst case - Ꝋ (log n) comparisons**

In the worst case, the item X does not exist in the array A at all. Through each recursion or iteration of Binary Search, the size of the admissible range is halved. In other words, the search space in Binary search is divided in half during each iteration until the target element is found or the search space is exhausted.

**Worst Case Analysis: Method1**

First time through loop, loss half of array (2 comps)

Second time, half remainder (1/4 original) 2 comps

Third time, half remainder (1/8 original) 2 comps

Original size of the array is N and it is halved in each iteration.

Loop Iteration Remaining Elements

1 N/2

2 N/4

3 N/8

4 N/16

…

?? 1

How long to get to 1?

Look at the problem in reverse, how long (or how many times) to double the number 1 until we get N?

N=2X and solve for X

Taking log on both sides

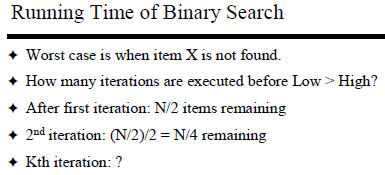
Log2N= Log2(2X)

X= log2N

There are 1 comparison in each iteration of while loop, plus one comparison at the end (on which while loop condition gets false)—binary search takes

1log2N +1 or Ꝋ(log2N) time in worst case.

**Analysis of Binary Search: Method2**



Loop Iteration Remaining Elements

1 N/2

2 N/4

3 N/8

4 N/16

…

Kth N/2k

**K**th iteration: N/2k remaining

But After k iterations, the length of array becomes 1

Therefore, length of array is N/2k=1

Worst case: Last iteration occurs when N/2k = 1

* 2k=N taking log on both sides

=> log2 2k  = log2 N => Number of iterations is K ≤ log2 N

Worst case running time= Ꝋ (log N)

