Linear Search

```
int search(int[] list, int target, int n)
 {
   for (int i=1; i<=n; i++)
     if (target == list[i]) return i;
   return -1;
 }
```

Order of growth for worst case?

Lower Bound on Searching an Unordered List

Prove we can’t do any better than T(n)=n by showing that any algorithm with less than n comparisons will be incorrect for some list L.

Suppose algorithm has n-1 comparisons (or less). Then there exists an element of L which is never compared to the target. So if that is the one that we are searching for the algorithm fails.

Can you find a flaw in that argument? There is one!

Lower Bound on Searching an Unordered List

Prove we can't do any better than T(n)=n by showing that any algorithm with less than n comparisons will be incorrect for some list L.

Try 2: Suppose algorithm does k comparisons of preprocessing L and j comparisons with target.

Preprocessing splits L into m "pieces" which are "connected" by comparisons. What is the minimum number of comparisons for m pieces?

Lower Bound on Searching an Unordered List

Prove we can’t do any better than T(n)=n by showing that any algorithm with less than n comparisons will be incorrect for some list L.

Try 2: Suppose algorithm does k comparisons of preprocessing L and j comparisons with target.

Preprocessing splits L into m “pieces” which are “connected” by comparisons. What is the minimum number of comparisons for comparing the target?

Lower Bound on Searching an Unordered List

Prove we can’t do any better than T(n)=n by showing that any algorithm with less than n comparisons will be incorrect for some list L.

Try 2: Suppose algorithm does k comparisons of preprocessing L and j comparisons with target.

Given the previous two answers, why must j+k be at least n?

Binary Search on sorted list

```
int Bsearch(int[] list, int target, int low, int high){
   if (low == high)
      if  target == list[low]) return low;
      else return -1;
   else {
      mid = floor((low+high)/2);
      if (target > list[mid]
            return Bsearch(list,target,mid+1,high);
      else
            return Bsearch(list,target,low,mid);
   }
 }
```

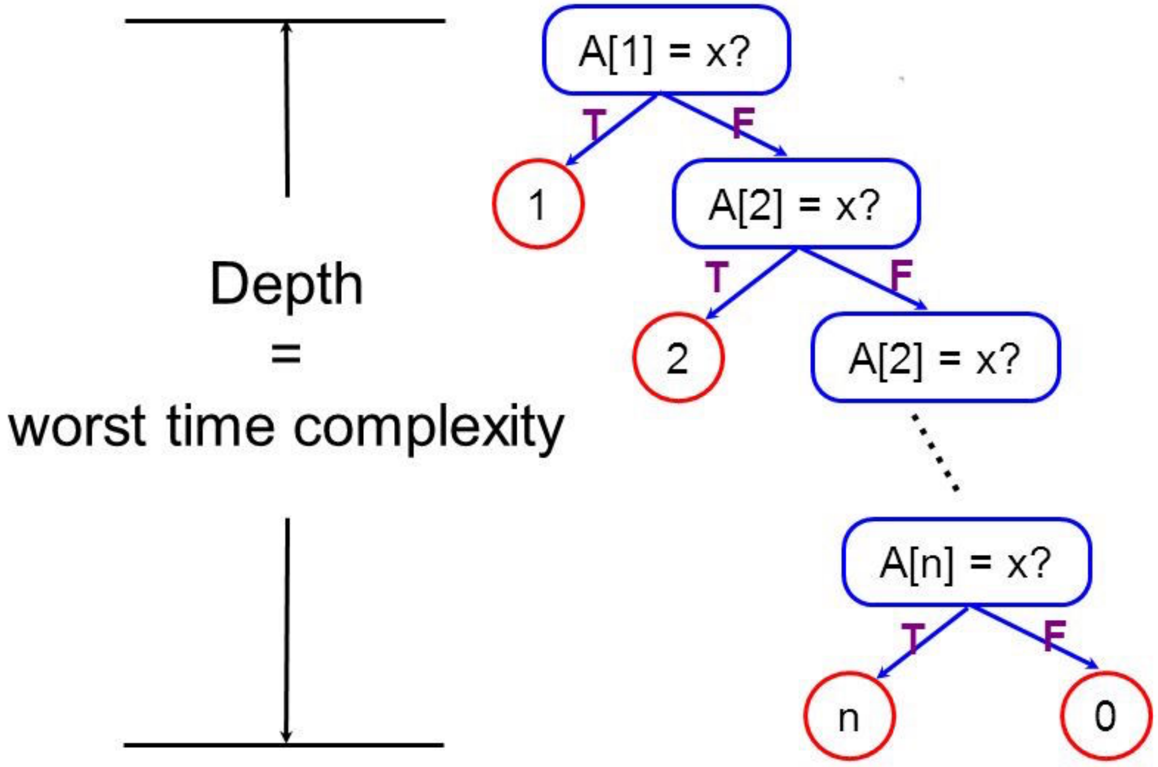
Give the recurrence relation for worst case number of comparison tests.

Binary Search on sorted list

```
T(1) = 1
T(n) = T(n/2) + 1
```

Solve the recurrence to get the number of comparisons we are doing.

Decision Tree for Linear Search



What would a decision tree for Binary Search look like?

Lower Bound on Search

Given ANY algorithm for search, look at the decision tree to determine the worst case.

We need to know facts about trees.

Use induction to show that a binary tree of height h has at most $2^{h+1} - 1$ nodes.

Lower Bound on Search

Given ANY algorithm for search, look at the decision tree to determine the worst case.

A binary tree of height h has at most $2^{h+1} - 1$ nodes.

What is the minimum height of a decision tree with n nodes? Why?

Binary Search is Optimal!

But wait… that is assuming that all we can do is compare values.

Which of the following would change the analysis?

A. Data is not sorted or sortable
B. Data is sorted but in a data structure where it does not cost the same to get at each location
C. Data is static so we know all possible search requests
D. All of the above