Data Structure & Algorithms

Lec 07: Searching & Sorting

Part-01

Fall 2017 - University of Windsor
Dr. Sherif Saad

Agenda

1. What is Searching?
2. Types of Searching
3. Linear Search
4. Binary Search
5. Interpolation Search
6. Searching Implementation

Learning Outcome

By the end of this class, you should be able to

- Explain different search algorithms and their run-time complexity
- Explain the effect of the data structure on the performance of the search algorithm

What is Searching?

Searching is the process of finding an item with specified properties (key) from a collection of items.

The key or the search key could be a simple (primitive) data type (e.g., Integer, Char, etc) or complex data type (e.g. Object, Image, Audio File, etc.)

The feasible region defining the set of all possible solutions is known as the search space (e.g Arrays, DB tables, Graph, etc)

The items in the search space may be sorted based on some properties or unsorted.

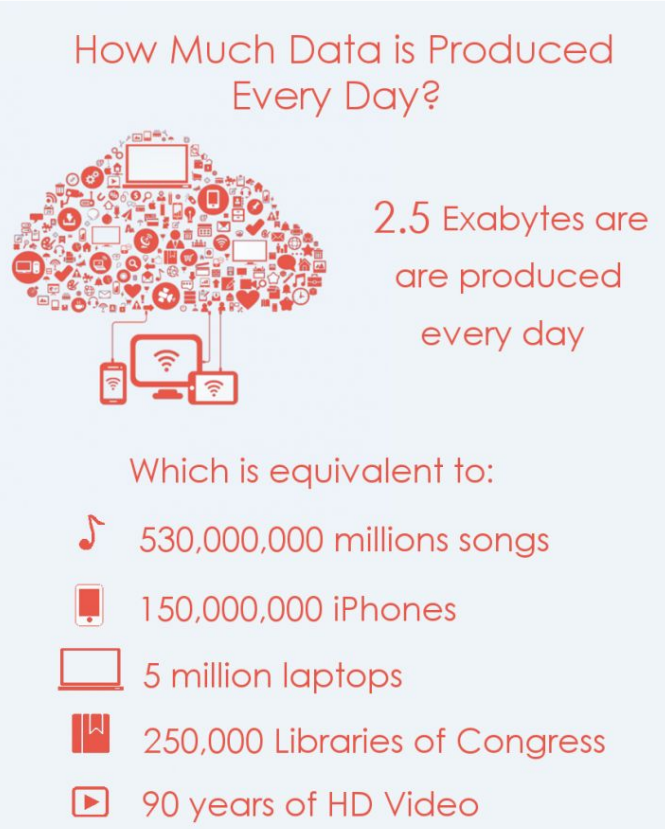
Why Searching?

Computer systems store a lot of data (e.g., Tweets, facebook posts, images) The total amount of data in the world was 4.4 zettabytes in 2013. That is set to rise steeply to 44 zettabytes by 2020.

1 Zettabytes = 1 billion terabytes = 1 trillion gigabytes

To query the stored data and retrieve information from the stored data we need efficient searching algorithms.

There are certain ways to organize the stored data to improve the searching process.



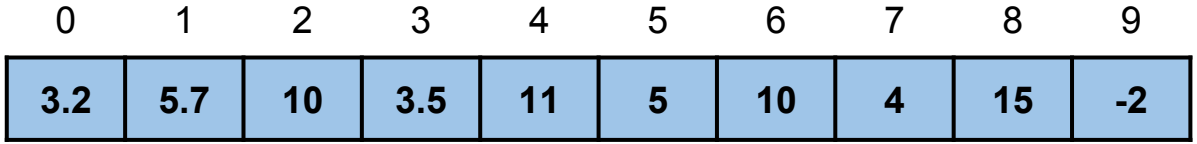
Queue as ADT

1. Linear Search
2. Jump Search
3. Binary Search
4. Interpolation Search
5. Heuristic search
6. Genetic Algorithm

Linear Search

Let us assume that our search space is an array of size n. There are n elements in the array (e.g., n integer numbers)

The order of the elements in the array is not known. This means the elements in the array are not sorted.

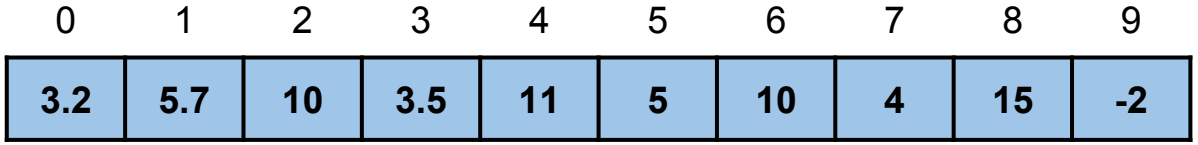


Array of 10 numbers

In-Class Activity

Write an algorithm to check if an array of size n is sorted or not?

Hint: make sure your algorithm is a good algorithm (e.g. Generality)



Linear Search (cont...)

How does it work?

We look at each item in the list in turn, quitting once we find an item that matches the search term or once we've reached the end of the list.

We have to scan the entire array and see if the element is there or not. If the size of the array (search space) is n, then we have to scan n elements.

The number of steps the algorithm needs to perform to decide if the item exist or not is n.

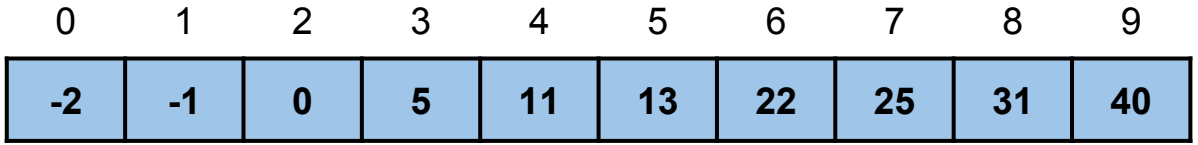
Linear Search (cont…)

```
Index = -1
For each item e in the List:
    If e match the search criteria
        Index = index of e
        return index
return index
```

```
7  int LinearSearch(int key, int data[], int size){
8
9      int index =-1;
10
11     for(int i=0; i<size; i++){
12
13         if(data[i]==key){
14             index = i;
15             return index;
16         }
17     }
18
19     return index;
20 }
```

Linear Search (cont...)

If the data is sorted (if we know that the data is sorted in the search space). Could we improve the linear search?



Linear Search (cont...)

If the data is sorted (if we know that the data is sorted in the search space). Could we improve the linear search?

```
index = -1
For each item e in the List:
    If e match the search criteria
        Index = index of e
        return index
    Else if current item > e
        return index
```

what is the run time in this case?

what if we have a large number of items?

Linear Search (cont...)

What is the performance of the linear search?

1. What is the fewest number of comparisons necessary to find an item? [Best Case]
2. What is the most number of comparisons necessary to find an item? [Worst Case]
3. On average, how many comparisons does it take to find an item in the list?[Average Case]

Linear Search (cont...)

Performance of the Linear Search

- Best Case: when the item is the first item in the list
- Worst Case: when the item is that last or does not exist in the list
- Average Case: the item usually something in the middle of the list (N/2)

The number of comparisons is proportional to the number of items in the array, N. The linear search has a time complexity of N

Binary Search

Let us consider searching for a word in a dictionary or a name in a phone book.

We usually go to an approximate page (e.g., middle page) then if the name we are looking for is on this page then we are done. Otherwise, we see if the current page is before or after the page we are looking for.

If the current page after the page we are looking for, then we repeat the same process. However, we limit the end of our search to the current page otherwise we change the

Binary Search (cont...)

Does it make sense to look for "Sean" by starting at the beginning and looking at each name in turn.

The binary search algorithm divides the search space into two regions or partitions. Then based on search key, it ignores one partition and searches the second one (That is why it is called binary).

How many comparisons we need to find the Adam's phone number?

Adam	: 250 887 1225
Alex	: 250 787 1242
Alice	: 250 677 4221
Andy	: 250 999 2251
Barbara	: 250 887 8811
Becky	: 250 117 1125
John	: 250 856 7612
Mike	: 250 712 6312
Sarah	: 250 471 6315
Sean	: 250 823 3625
Tom	: 250 927 3462

Phone Book

Binary Search (cont...)

The binary search navigates the search space by using the beginning and the end of the list to find the middle of the list.

Mid = (Begin+End)/2.

If the item at the middle of the list is lower of the item, we are searching for we eliminate the part from the beginning to the middle of the list and repeat our search starting from the item next to the middle to the end of the list.

If the item at the middle is bigger we eliminate the part from the middle to the end of the list and repeat our search with new end

Binary Search (cont...)

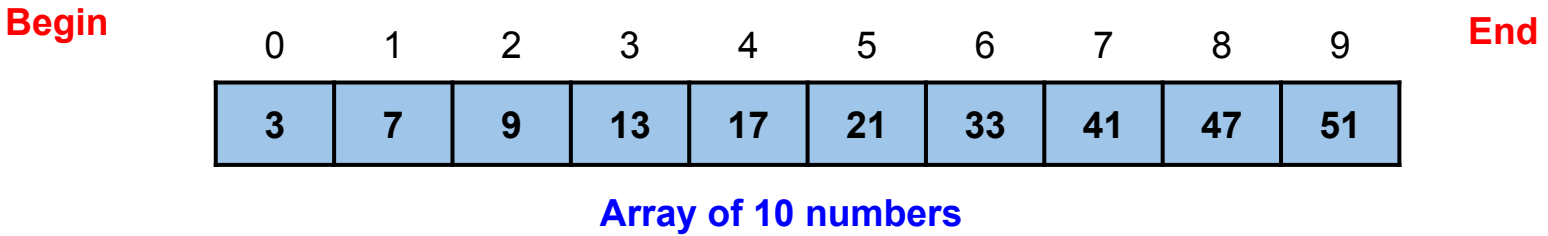
The binary search continue updating (changing) the beginning and the end of the list until it finds the item or the index of the beginning is greater than the index of the end.

Binary search applies an algorithm design paradigm known as ***decrease and conquer*** (note this is not a divide and conquer paradigm).

The run time complexity of of the binary search ***log n***. Since we only considering half of the problem (search space) after each comparison.

Binary Search (cont...)

Given the following array of 10 integers value use binary search to find if the array contains the following values {7, 45)



Begin = 0
End = |Array| -1

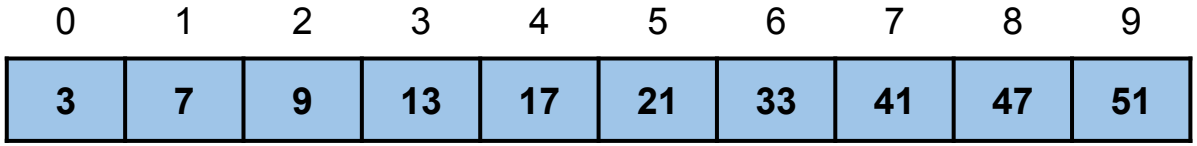
Middle = (Begin+End)/2

Binary Search (cont...)

Find the index of 7 if 7 exist in the array

Begin

End



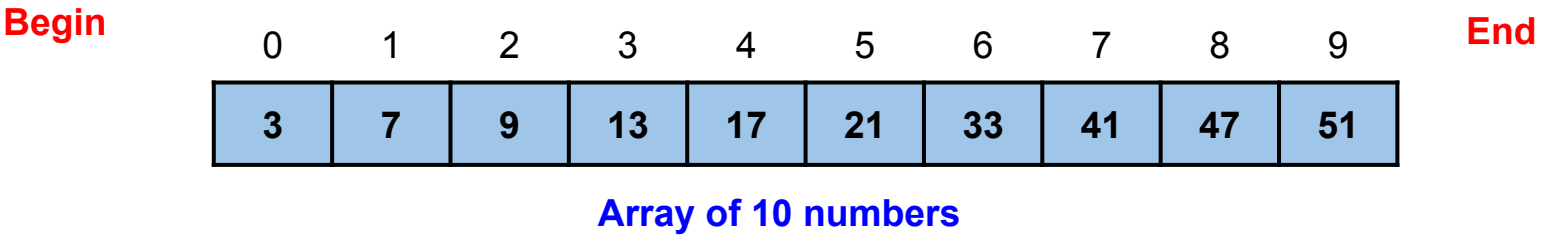
Array of 10 numbers

Begin = 0
End = 9

```
Middle   = (Begin+End)/2
         = (0+9)/2= 4
IF Array[Middle] == Key
   Array[4] == 7
         17 == 7
IF Array[Middle] < Key
    Begin = Middle + 1
Else : End = Middle -1
```

Binary Search (cont...)

Find the index of 7 if 7 exist in the array



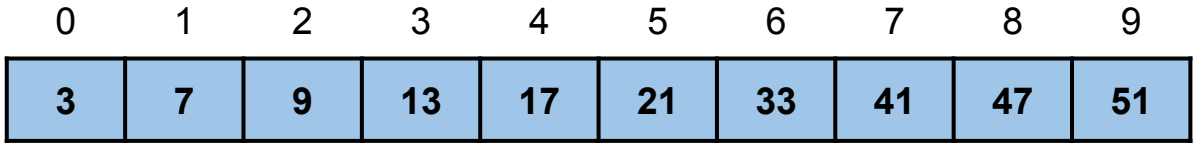
Begin = 0
End = 3

```
Middle   = (Begin+End)/2
         = (0+3)/2= 1
IF Array[Middle] == Key
   Array[1] == 7
         7 == 7
IF Array[Middle] < Key
    Begin = Middle + 1
Else : End = Middle -1
```

Binary Search (cont...)

Find the index of 45 if 45 exist in the array

Begin



End

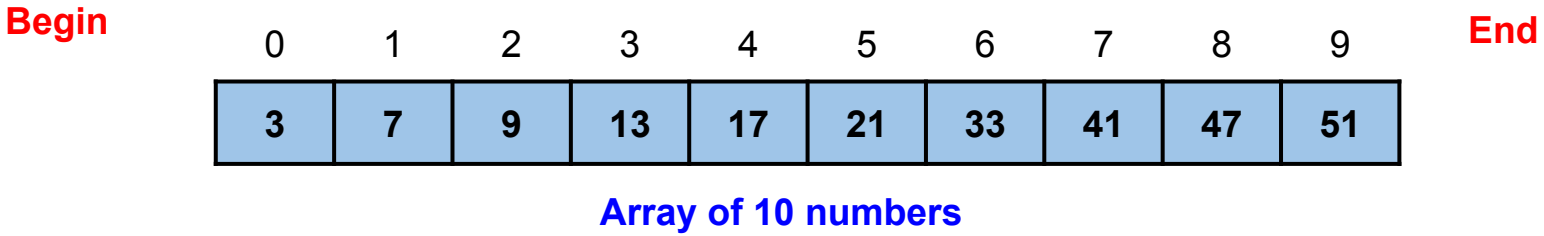
Array of 10 numbers

Begin = 0
End = 9

```
Middle   = (Begin+End)/2
         = (0+9)/2= 4
IF Array[Middle] == Key
   Array[4] == 7
         17 == 45
IF Array[Middle] < Key
    Begin = Middle + 1
Else : End = Middle -1
```

Binary Search (cont...)

Find the index of 45 if 45 exist in the array

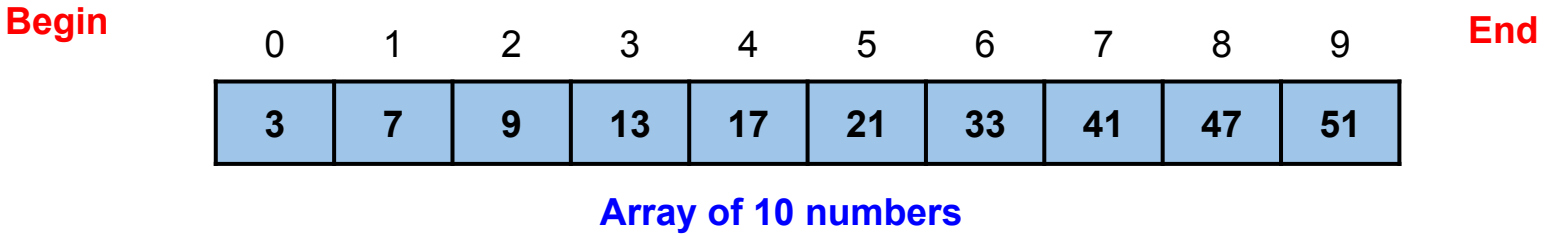


Begin = 5
End = 9

```
Middle   = (Begin+End)/2
         = (5+9)/2= 7
IF Array[Middle] == Key
   Array[7] ==45
         41 == 45
IF Array[Middle] < Key
    Begin = Middle + 1
Else : End = Middle -1
```

Binary Search (cont...)

Find the index of 45 if 45 exist in the array

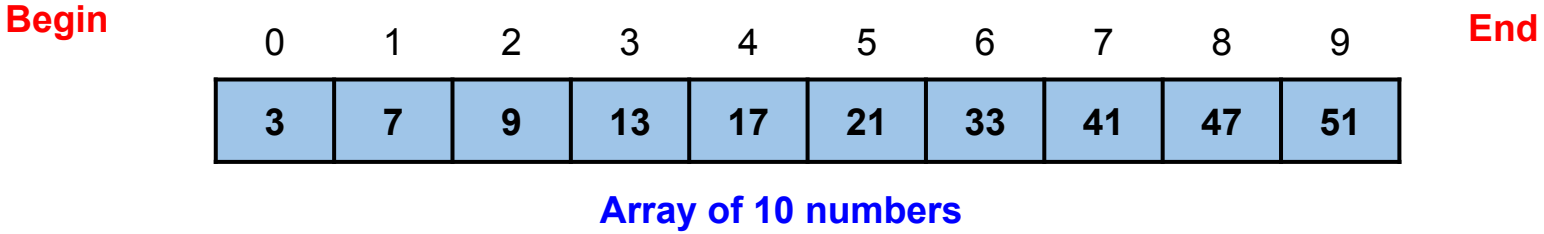


Begin = 8
End = 9

```
Middle   = (Begin+End)/2
         = (8+9)/2= 8
IF Array[Middle] == Key
   Array[8] ==45
         47 == 45
IF Array[Middle] < Key
    Begin = Middle + 1
Else : End = Middle -1
```

Binary Search (cont...)

Find the index of 45 if 45 exist in the array



Begin = 8
End = 7

```
Middle   = (Begin+End)/2
         = (8+9)/2= 8
IF Array[Middle] == Key
   Array[8] ==45
         47 == 45
IF Array[Middle] < Key
    Begin = Middle + 1
Else : End = Middle -1
```

Binary Search (cont...)

```
Binary Search Algorithm
Let Begin = 1, End = N, Mid = N/2
While (item not found and Begin < End)
    compare search term to item at mid If match
        return index
    Else If search term is less than item at mid,
        End = Mid-1
    Else
        Begin = Mid+1
    Mid = (Begin+End)/2
```

Binary Search (cont...)

```
22 int BinarySearch(int key, int data[], int size){
23
24     int begin =0;
25     int end = size-1;
26     int mid;
27
28     while(begin<=end){
29         mid = (begin+end)/2;
30
31         if(data[mid]==key)
32             return mid;
33         if(data[mid] < key)
34             begin = mid + 1;
35         else end = mid -1;
36     }
37     return -1;
38 }
```

Binary Search (cont...)

What is the performance of the binary search?

1. What is the fewest number of comparisons necessary to find an item? [Best Case]
2. What is the most number of comparisons necessary to find an item? [Worst Case]
3. On average, how many comparisons does it take to find an item in the list?[Average Case]

Interpolation Search

The interpolation search try to follow the way we human search for a name in a phone book or a word in dictionary.

If we are looking for Alex's number we will not start from the middle.

In mathematics, **interpolation** is an estimation of a value within two known values in a sequence of values.

Adam	: 250 887 1225
Alex	: 250 787 1242
Alice	: 250 677 4221
Andy	: 250 999 2251
Barbara	: 250 887 8811
Becky	: 250 117 1125
John	: 250 856 7612
Mike	: 250 712 6312
Sarah	: 250 471 6315
Sean	: 250 823 3625
Tom	: 250 927 3462

Phone Book

Interpolation

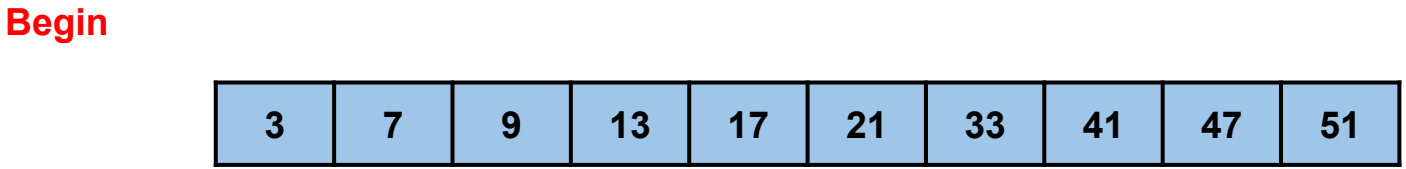
In case of searching rather than always going to the middle as in binary search, we want to estimate the location of the checkpoint based on the key and the actual values in the list.

This means we will not always seek the middle of the list every time we perform a comparison.

Interpolation Search (cont...)

Binary Search always goes to middle of the list. Interpolation search may go to different locations according the value of key being searched.

Loc = Begin + ((End-Begin) / (List[End]-List[Begin])) * (key - List[Begin]))




```
Begin  = 0
End=size -1
If Key = 7
     Loc = 0 + ( (9-0) / (51-3) ) * (7 - 3) = 1
If Key = 45
          Loc = 0 + ( (9-0) / (51-3) ) * (45 -3) = 7
```

Interpolation Search (cont...)

```
Binary Search Algorithm
Let Begin = 1, End = N, Loc;
While (item not found and Begin < End)
    Loc = Begin + ( (End-Begin) / (List[End]-List[Begin]) ) * (key - List[Begin]))
    compare search term to item at mid If match
        return index
    Else If search term is less than item at mid,
        End = Loc-1
    Else
        Begin = Loc+1
```

Comparing Basic Search Algorithms

Algorithm	***Worst Case***	***Average Case***
Linear Search	N	N/2
Jump Search (Linear Search with ordered list and jumping ahead by fixed steps	N	√N
Binary Search	Log N	Log N
Interpolation Search	N	Log (Log n)

Linear Search Vs Binary Search

Assuming that a single comparison cost 1 ms

Search Space Size (number of elements)	Linear Search Performance	Binary Search Performance
1000	1 sec	10 ms
10,000	10 sec	13 ms
100,000	1.67 minutes	16 ms
1000,000	16.66 minutes	20 ms
10,000,000	2.76 hours	23 ms

Searching Implementation

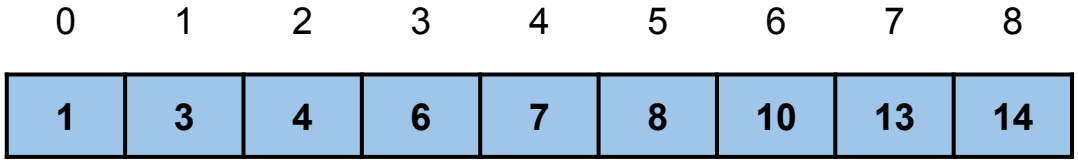
Searching Linked List using binary search is inefficient because with a linked list we do not have the direct access. The only way to traverse the list is by following the pointers? This means the runtime is O(N)

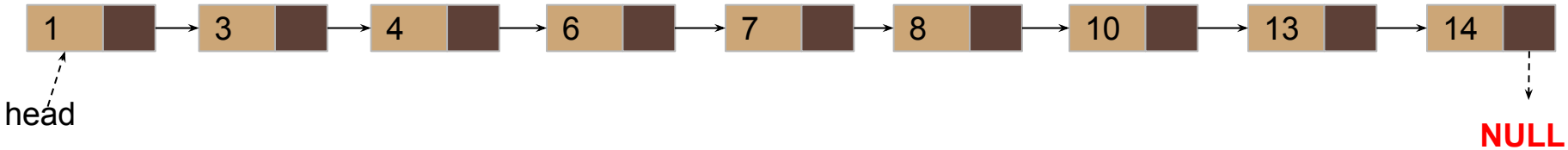
What about Jump search could we still get √N when storing the data in a linked list instead of array?

Searching Implementation

How does the binary search work?

Could we come up with a dynamic data structure that enable use to search in O(log n)?

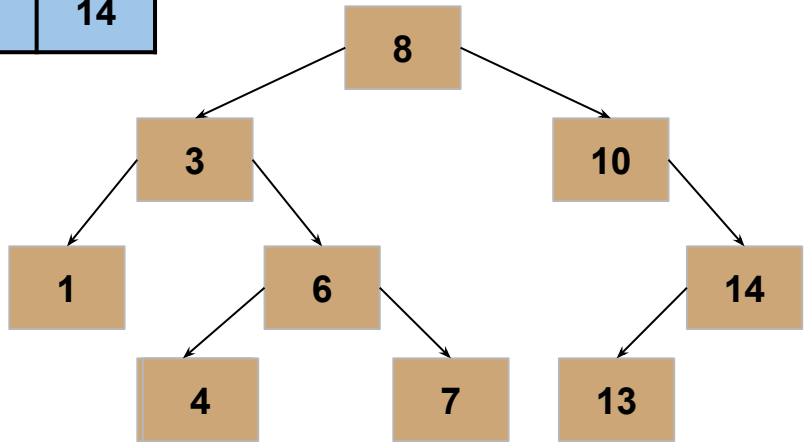




Searching Implementation

We need to store our data in a non linear dynamic data structure to perform search in log n

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14



Self-assessment

Explain why binary search with a dynamic linked list is inefficient?
Given singly linked list explain how we could change it to support Jump Search.