Principles of Computer Science

Binary Search

Today's Lecture

- Searching an Unsorted Array
  - Iterative Sequential Search
  - Recursive Sequential Search
  - Efficiency of Sequential Search
- Searching a Sorted Array
  - Binary Search
  - Why it doesn't work for LinkedLists?
  - Java Class Library: the Method **`binarySearch`**
  - Efficiency of Binary Search
- Choosing a Search Method

Recursive Sequential Search an Unsorted Array

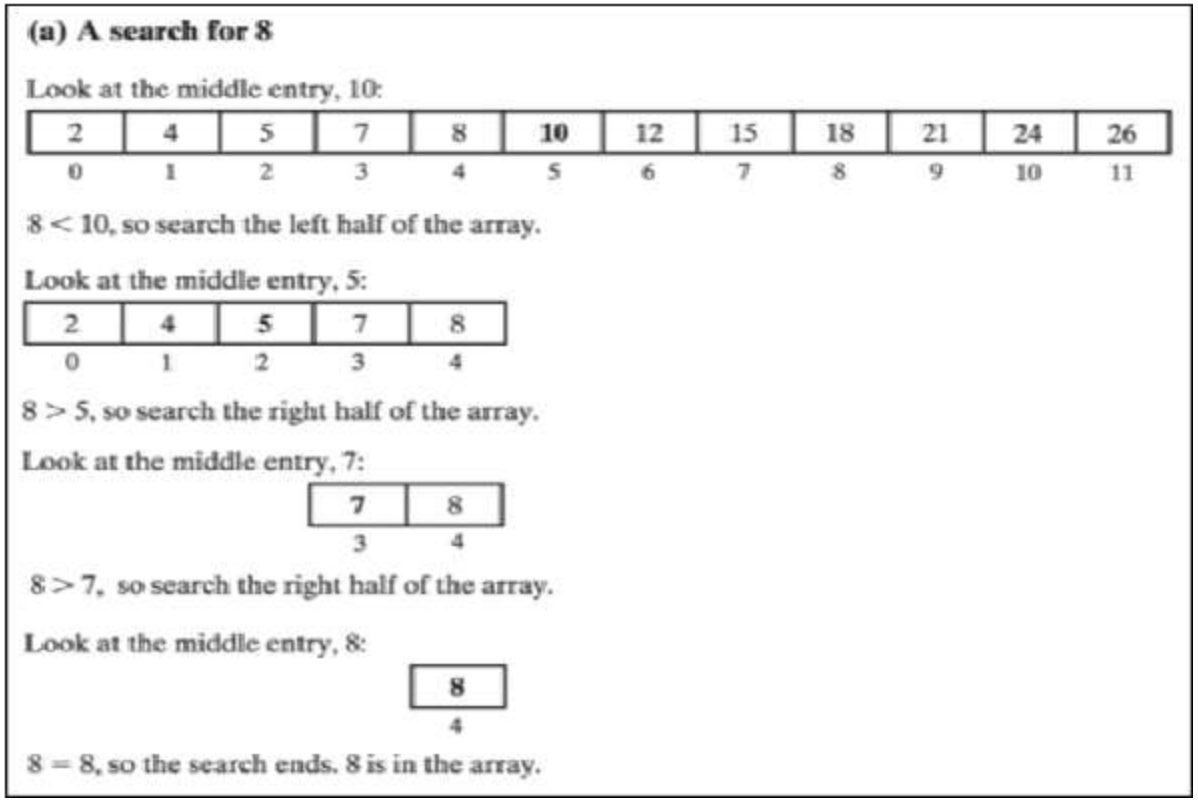
- Pseudocode for a recursive algorithm to search an array.

```
Algorithm to search a [first]
        through a[last] for desiredItem
if (there are no elements to search)
    return false
    else if (desiredItem equals a [first])
        return true
    else
        return the result of
        searching a [first + 1] through a [last]
```

Efficiency of a Sequential Search

- Best case *O(1)*
  - Locate desired item first
- Worst case *O(n)*
  - Must look at all the items
- Average case *O(n)*
  - Must look at half the items
  - O(n/2) is still O(n)

Binary Search of Sorted Array



A recursive binary search of a sorted array that finds its target.

Binary Search of Sorted Array

(b) A search for 16

Look at the middle entry, 10:

2	4	5	7	8	**10**	12	15	18	21	24	26
0	1	2	3	4	5	6	7	8	9	10	11

16 > 10, so search the right half of the array.

Look at the middle entry, 18:

12	15	**18**	21	24	26
6	7	8	9	10	11

16 < 18, so search the left half of the array.

Look at the middle entry, 12:

12	15
6	7

16 > 12, so search the right half of the array.

Look at the middle entry, 15:

15
7

16 > 15, so search the right half of the array.

The next subarray is empty, so the search ends. 16 is not in the array.

A recursive binary search of a sorted array that does not find its target.

Binary Search of Sorted Arrays?

- Let int[] array be a **large** sorted array of integers.
  - Example:
    ```
    int array = {1,2,8,9,10,13, …. },
    arrays.length = 1,000,000
    ```

- How do we search for a specific value?
  - `int find(int [] array, int value)`
    returns the index of the value in the array or -1 if the value is not found
  - Example
    - `find(array, 12)` returns -1
    - `find(array,10)` returns 4

Binary Search

- Easy but highly efficient recursive algorithm—human brain *perhaps* performs binary search **all the time.**

Efficiency of a Binary Search

Easy but highly efficient recursive algorithm – human brain performs binary search **all the time.**

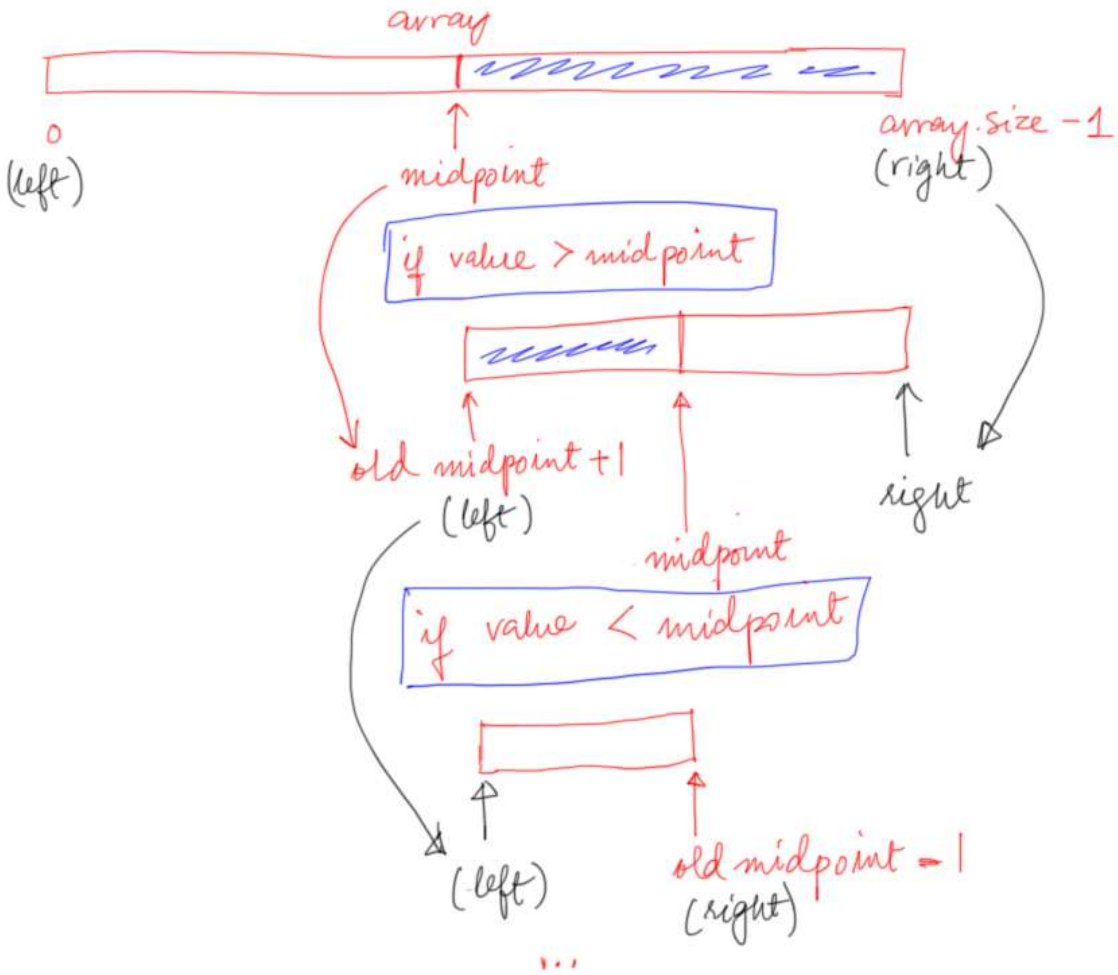
- Best case *O(1)*
  - Locate desired item first
- Worst case *O(log n)*
  - Must look at all the items
- Average case *O(log n)*

Binary Search

```
int binarySearch(int[] array, int value) {
  return findInRange(array, value, 0, array.length-1);
}

int findInRange(int[] array, int value,  int left, int right) {
  if( right < left )
    return -1;
  else {
    int midpoint = (left + right)/2;
    if( array[midpoint] == value )
      return midpoint;
    else if( array[midpoint] < value )
      return findInRange(array, value, midpoint+1, right);
    else
      return findInRange(array, value, left, midpoint-1);
  }
}
```

Binary Search



Time Complexity of Binary Search

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ T(n/2) + 1 & \text{otherwise} \end{cases}$$

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= T(n/4) + 2 \\ &= T(n/8) + 3 \\ &\cdots \\ &= T(n/2^i) + i \end{aligned}$$

Note that the size of the input decreases with every recursive step. The recursion stops when the input equals 1.

$n/2^i = 1 \Rightarrow n = 2^i$ ← number of steps when the input equals 1.

$\Rightarrow i \log(2) = \log(n) \Rightarrow i = \log(n)/\log(2)$

How long does it take to search through sorted array?

- Find a person in Toronto:
  - 4 million people: $\log(4 \times 10^6) = 22$
  - Each iteration takes about 1 microsecond
  - 22 microseconds or
  - 45,000 names per second

- Find a phone number:
  - 100 million phone numbers: $\log(4 \times 10^6) = 26$
  - 26 microsecond, 38,000 numbers per second.

Why We Cannot Do Binary Search on LinkedList?

- In binary search, we need to jump to midpoint of [left, right]:

```
int midpoint = (left + right)/2;
…
array[midpoint] < value …
```

- *This requires random access of arrays.*

- LinkedList does not support random access:
  - These can only iterate forward or backward one element at a time.

What is not great about using Arrays?

- Very difficult to maintain when the data is dynamic:
  - How do you insert an element after the array is already sorted?
  - How do you delete an element after the array is already sorted?
  - Not easy (can be done using *heaps*).

Binary Tree to the Rescue

- Binary trees are extensions to linked list.
- Binary Trees:
  - Support FAST insert and delete
  - Support FAST sort
  - Support FAST search

Reading

- Sec. 13.1