Binary Search Review

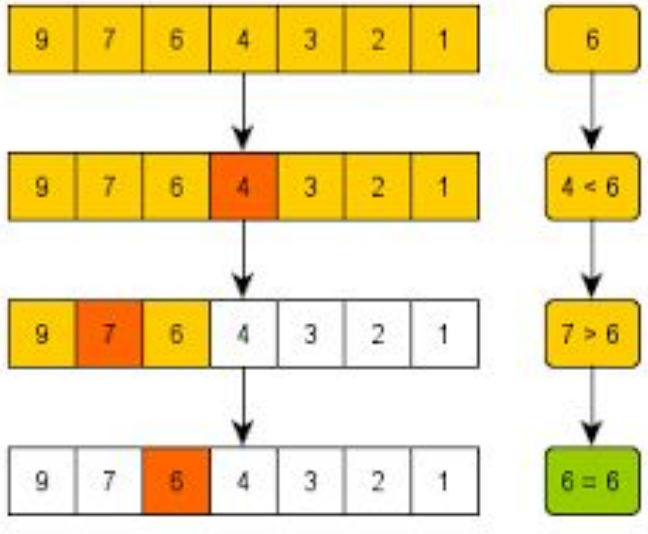
CS 112 Recitation (5)

Agenda

1. Quick warm-up
2. Binary Search Decision Trees
3. Coding Problem
4. Exam questions??

Binary Search - Warmup

1. What must the array (or other data structure) be to run binary search correctly?
2. What is the runtime of binary search?



Q1 - Decision Tree (Part 1)

-Draw the decision tree for binary search for an array of 8 elements. Include failure nodes, and mark comparisons on the nodes and branches.

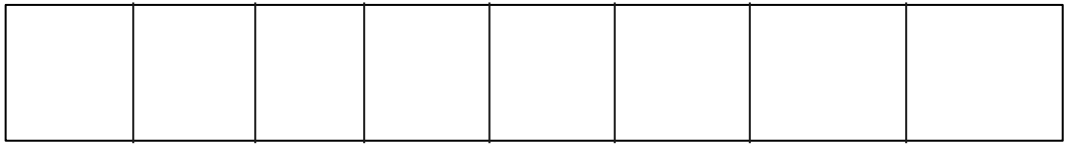
1. Which index has the least number of comparisons for a successful search? How many comparisons?
2. Which index or indexes have the maximum number of comparisons for a successful search? How many comparisons?
3. What is the least number of comparisons for unsuccessful search? How many comparisons?
4. What is the maximum number of comparisons for unsuccessful search? How many comparisons?
5. What is the average number of comparisons for successful searches?
6. What is the average number of comparisons for unsuccessful search if we don't know how many values are being searched.
7. What is the average number of comparisons for unsuccessful search. Assume the search values include 5 numbers that:
    a. 3 numbers that are smaller than the value at index 0.
    b. 2 numbers that are greater than the value at index 3 but is smaller than the value at index 5.

Q1 - Decision Tree (Part 2, Implementation)

```
public static int indexOf (int [] a, int key) {
    int lo = 0;
    int hi = a.lenghth - 1;
    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (key == a[mid]) return mid;
        if (key < a[mid]) hi = mid - 1;
        else lo = mid + 1;
    }
    return -1;  // key is not present in array a
}
```

Step 1: Draw the decision tree

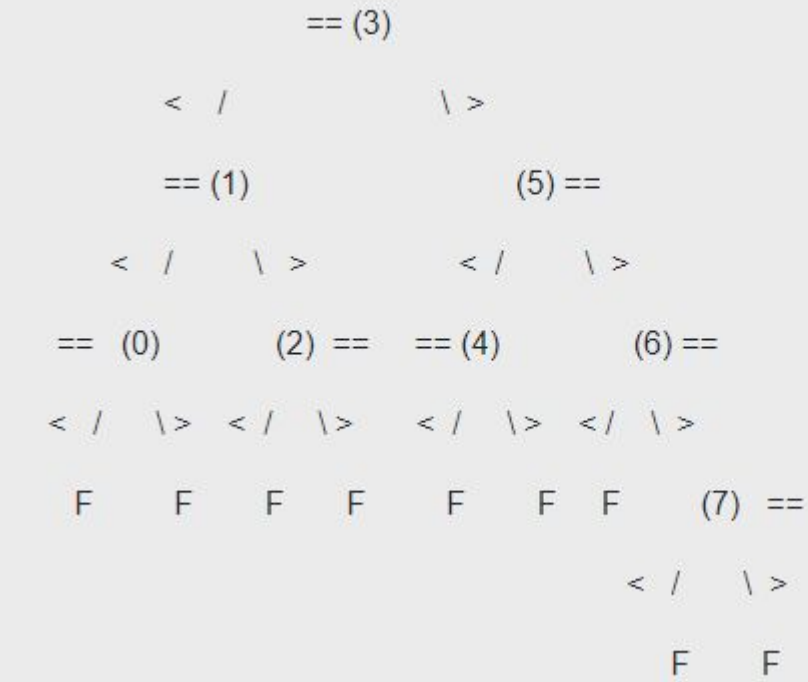
```
public static int indexOf (int [] a, int key) {
   int lo = 0;
   int hi = a.lenghth - 1;
   while (lo <= hi) {
      int mid = lo + (hi - lo) / 2;
      if (key == a[mid]) return mid;
      if (key < a[mid]) hi = mid - 1;
      else lo = mid + 1;
   }
   return -1;  // key is not present in array a
}
```



Q1 - Decision Tree Solution (Part 1)

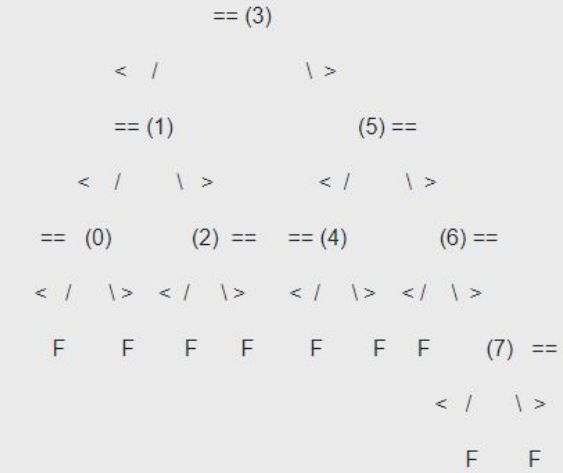
.

Solution



Q1 continued

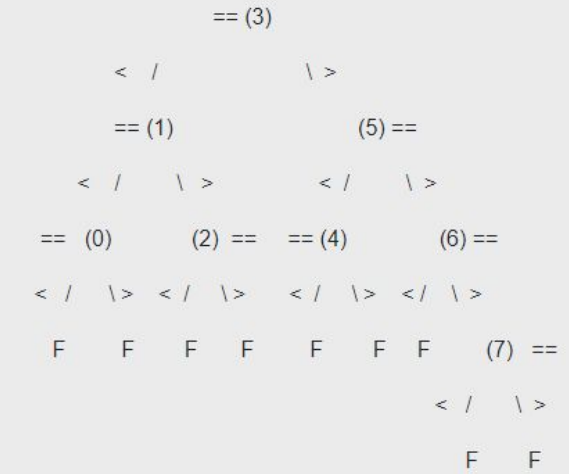
Solution



Which index has the least number of comparisons for a **successful** search? How many comparisons?

Q1 continued

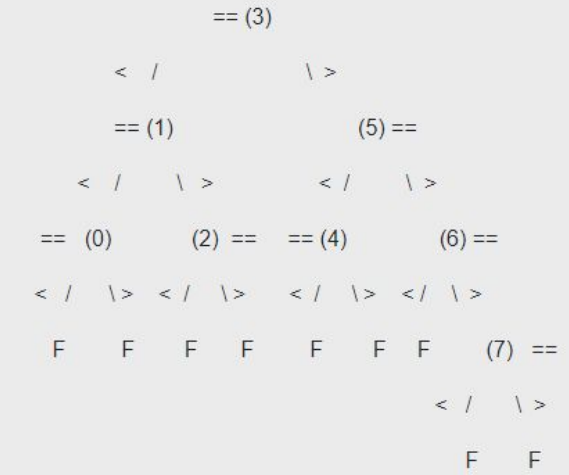
Solution



Which index or indexes have the maximum number of comparisons for a **successful** search? How many comparisons?

Q1 continued

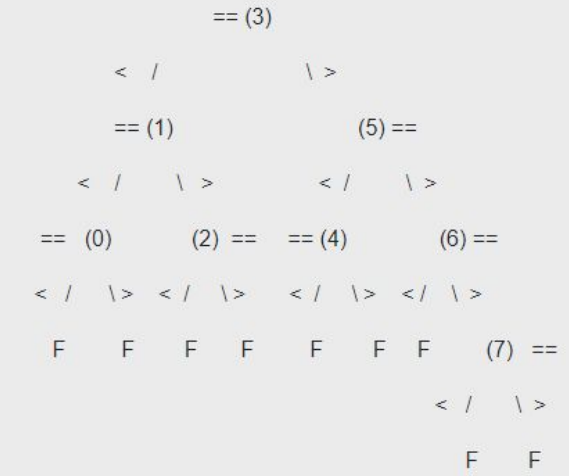
Solution



What is the least number of comparisons for **unsuccessful** search? How many comparisons?

Q1 continued

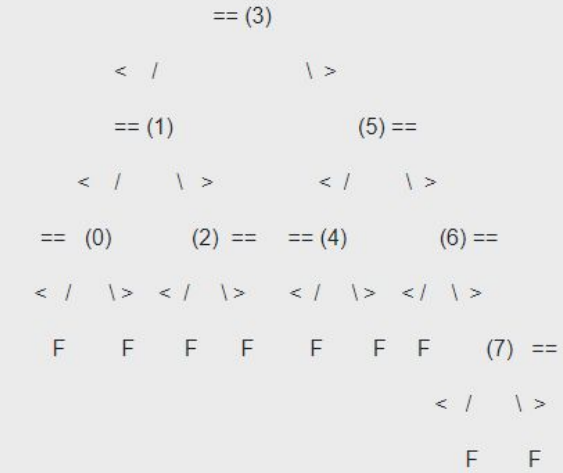
Solution



What is the maximum number of comparisons for **unsuccessful** search? How many comparisons?

Q1 continued

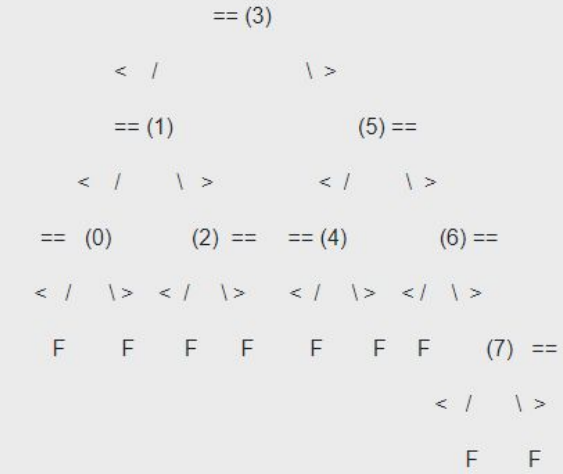
Solution



What is the average number of comparisons for successful searches?

Q1 continued

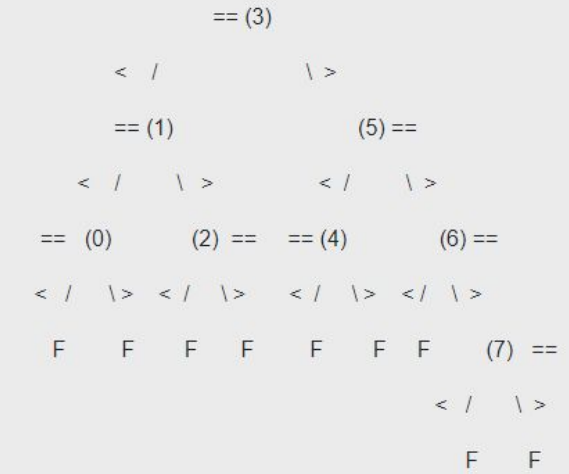
Solution



What is the average number of comparisons for unsuccessful search if we don't know how many values are being searched?

Q1 continued

Solution



What is the average number of comparisons for unsuccessful search. Assume the search values include 5 numbers that:

1. 3 numbers that are greater than the value at index 7
2. 2 numbers that are greater than the value at index 3 but is smaller than the value at index 5

Q1 - Decision Tree Solution (Part 2)

1. Index 4, 1 comparison
2. Index 0, 7 comparisons
3. 6 comparisons
4. 8 comparisons
5. Average = (1 node * 1 comparison) + (2 nodes * 3 comparisons) + (4 nodes * 5 comparisons) + (1 node * 7 comparisons) = 34 comparisons / 8 (number of elements)
6. We don't know, but the average will be between 6 and 8 comparisons
7. 24 comparisons [3 numbers will fail with 8 comparisons (to the left of index 0)] + 12 comparisons [2 numbers will fail with 6 comparisons (to the right of index 3)]. average = 36 comparisons / 5 number

Q2 (Optional) - Count Keys (Use the coding file!)

Instrument the following binary search implementation to use a Counter to count the total number of keys examined during all searches and then print the total after all searches are completed. Hint: create a Counter in main() and pass it as an argument to indexOf().

```
public static int indexOf (int [] a, int key) {
    int lo = 0;
    int hi = a.lenghth - 1;

    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;

        if (key == a[mid]) return mid;
        if (key < a[mid]) hi = mid - 1;
        else lo = mid + 1;
    }
    return -1;  // key is not present in array a
}
```

Code Template - Part 1

```
public class Counter {

	private final String name;
	private int count = 0;

	public Counter (String id) {
		name = id;
	}

	// fill code here
	public void increment() {
		count++;
	}

}
```

Code Template - Part 2

```
public static int indexOf(int[] a, int key, Counter c) {

        int lo = 0;
        int hi = a.length - 1

        while (lo <= hi) {
                int mid = (lo) + (hi - lo) / 2;
                c.increment(); // update your counter

                if (key < a[mid]) hi = mid - 1; // code here
                else if (key > a[mid]) lo = mid + 1;
                else return mid; // code here
        }

        return -1;
}
```

Q2 Solution (Part 2)

```java
public class Counter {

    private final String name;     // counter name
    private int count = 0;         // current value

    /**
     * Initializes a new counter starting at 0, with the given id.
     *
     * @param id the name of the counter
     */
    public Counter(String id) {
        name = id;
    }

    /**
     * Increments the counter by 1.
     */
    public void increment() {
        count++;
    }

    /**
     * Returns the current value of this counter.
     *
     * @return the current value of this counter
     */
    public int tally() {
        return count;
    }

    /**
     * Returns a string representation of this counter.
     *
     * @return a string representation of this counter
     */
    public String toString() {
        return count + " " + name;
    }
}
```

Q2 Solution (part 1)

```
public static int indexOf(int[] a, int key, Counter c) {
        int lo = 0;
        int hi = a.length - 1;
        while (lo <= hi) {
            int mid = lo + (hi - lo) / 2;
    c.increment();

            if      (key < a[mid]) hi = mid - 1;
            else if (key > a[mid]) lo = mid + 1;
            else return mid;
        }
        return -1;
    }
```

Exam 1

- Questions?

Good Work!

Go to https://dynrec.cs.rutgers.edu/live/

Enter the Quiz Code: SPT7