ENSF 593/594
Data Structures – Searching

Mohammad Moshirpour

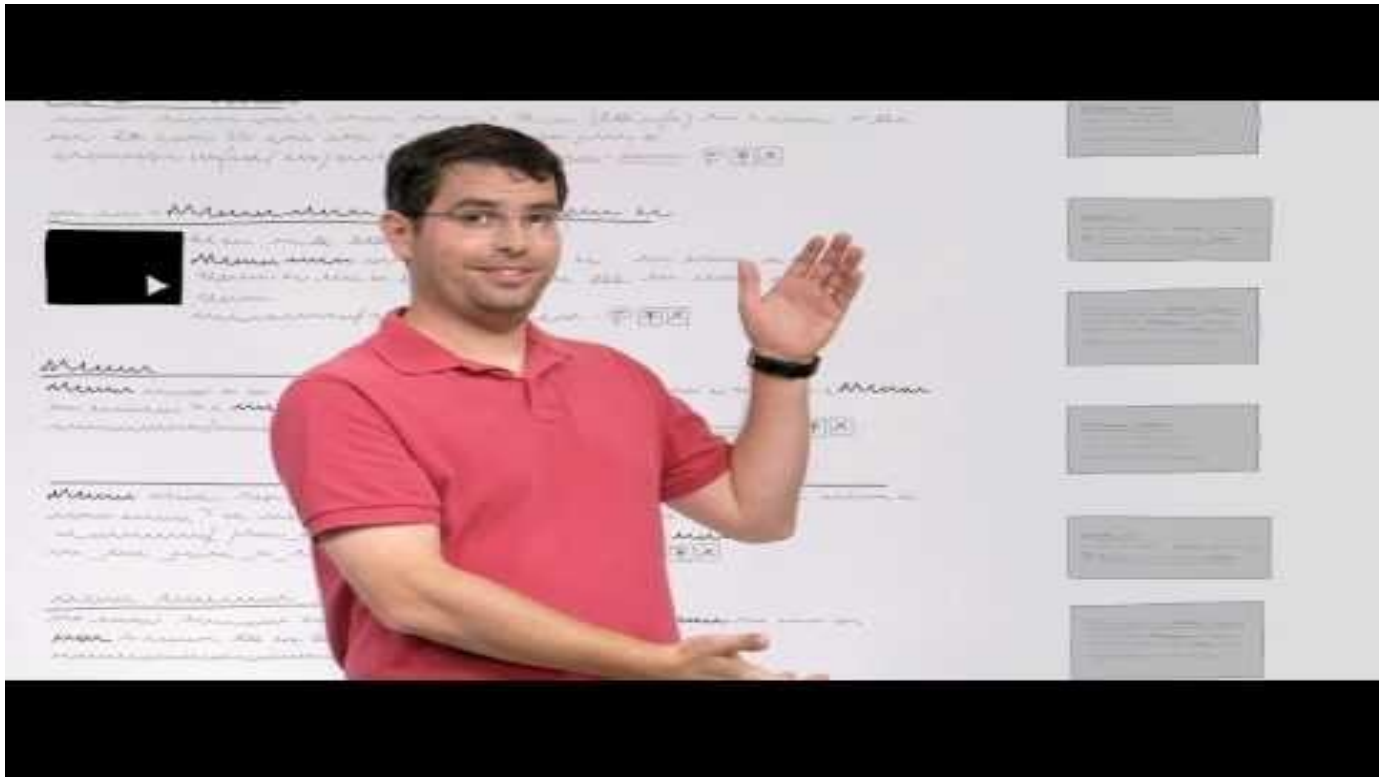
Outline

- Search Terminology
- Sequential Search
- Binary Search
- Interpolation Search

Goal

- In this lecture we are going to know about three search methods how they operate, on which kind of data they can be used and how efficient they are.

How Google Search Works?



https://www.youtube.com/watch?v=BNHR6IQJGZs

Terminology

- ***Data:*** independent fact, observations, or event
- ***Record:*** the data pertaining to a unique object
  - Sometimes called and *element*
  - Consists of one or more *fields*
- ***Field:*** a constituent part of a record, usually consisting of a single data element
  - Has a specified *type* and *size*

Terminology (Cont'd)

- ***File:*** a collection of records
- ***Key:*** the data field used to select or order records
  - May or may not be unique for a set of records
- ***Primary Key:*** the field used first for selecting or sorting records
  - E.g. Last names in a telephone book
- ***Secondary Key:*** the field used if 2 or more records have equal primary keys
  - E.g. First name in a telephone book

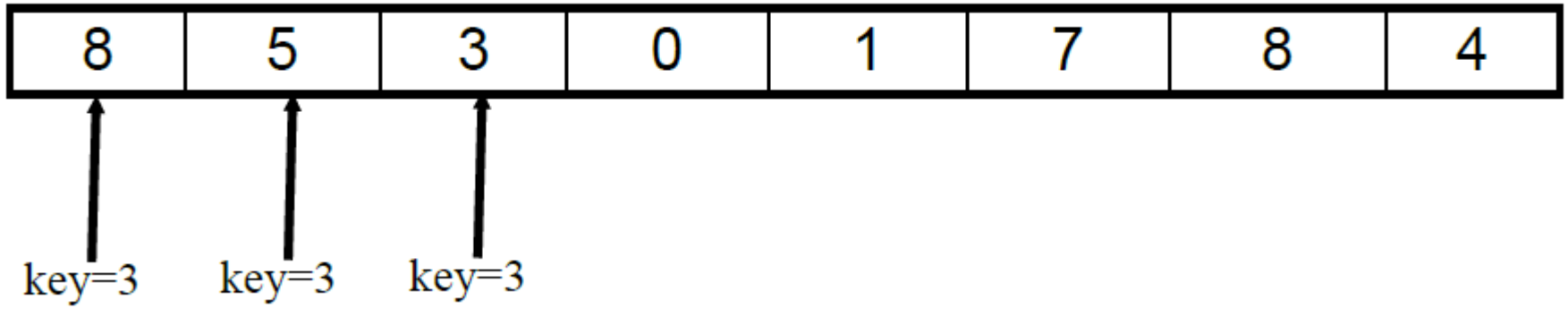
Terminology (Cont'd)

- ***Satellite Data:*** data in the non-key field, **not** used when sorting or selecting records
  - E.g. Phone numbers in a telephone book
- ***Search:*** an operation that returns a pointer to a record that matches a key value, or nil if there is no match
- ***Sorting:*** arranges the items of a list into ascending (or descending) order

Sequential Search

- **Basic idea:** starting at the beginning of a list, compare each successive item to a query key until we find a match or reach the end of the list
- Works on both sorted and unsorted lists
- Can be used on arrays and linked lists
- Likely to be the fastest algorithm on small lists (up to about 20 records)

Sequential Search (Cont'd)



- Very simple to implement
- Not the most efficient algorithm
- Also know as a linear search

Sequential Search (Cont'd)

- Java code to search an integer array:

```
int seqentialSearch(int[] array, int key)
{
      for (int i=0; i < array.length; i++)
            if (array[i] == key)
                  return i;  //success

      return -1; // failure key not found
 }
```

Sequential Search (Cont'd)

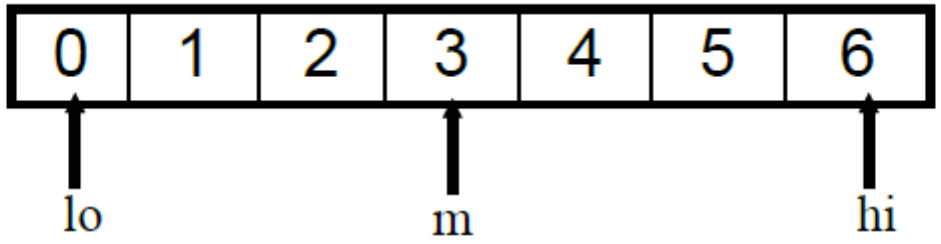
- In the best case we find a match immediately (perform 1 comparison)
- In the worst case (an unsuccessful search, or we find a match at the end), n comparisons occur
- On average, (n+1) / 2 comparisons are performed
- Is an *O(n)* algorithm

Binary Search

- Works only on a sorted array of records
- **Basic idea:**
  - Divide the array in half, by locating the middle item
    - If the query key matches this item, return
  - If the key is < middle item, divide the left subarray in half, applying the above steps recursively
  - If the key > middle item, recursively divide the right subarray in half
  - Keep halving subarrays until a match is found, or subdivision is no longer possible

Binary Search (Cont'd)

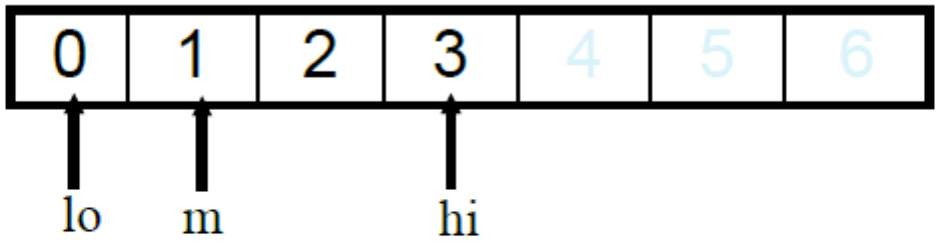
Key = 0



0 > 3?

Binary Search (Cont'd)

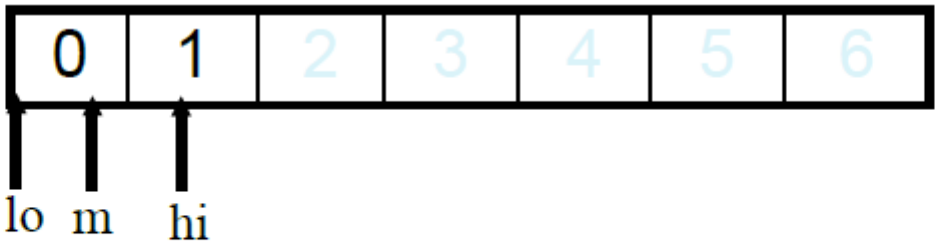
Key = 0



0 > 1?

Binary Search (Cont'd)

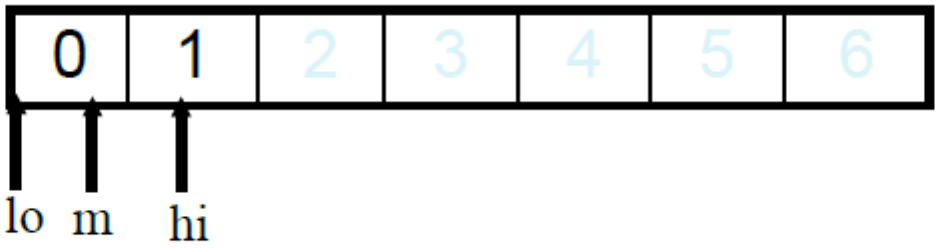
Key = 0



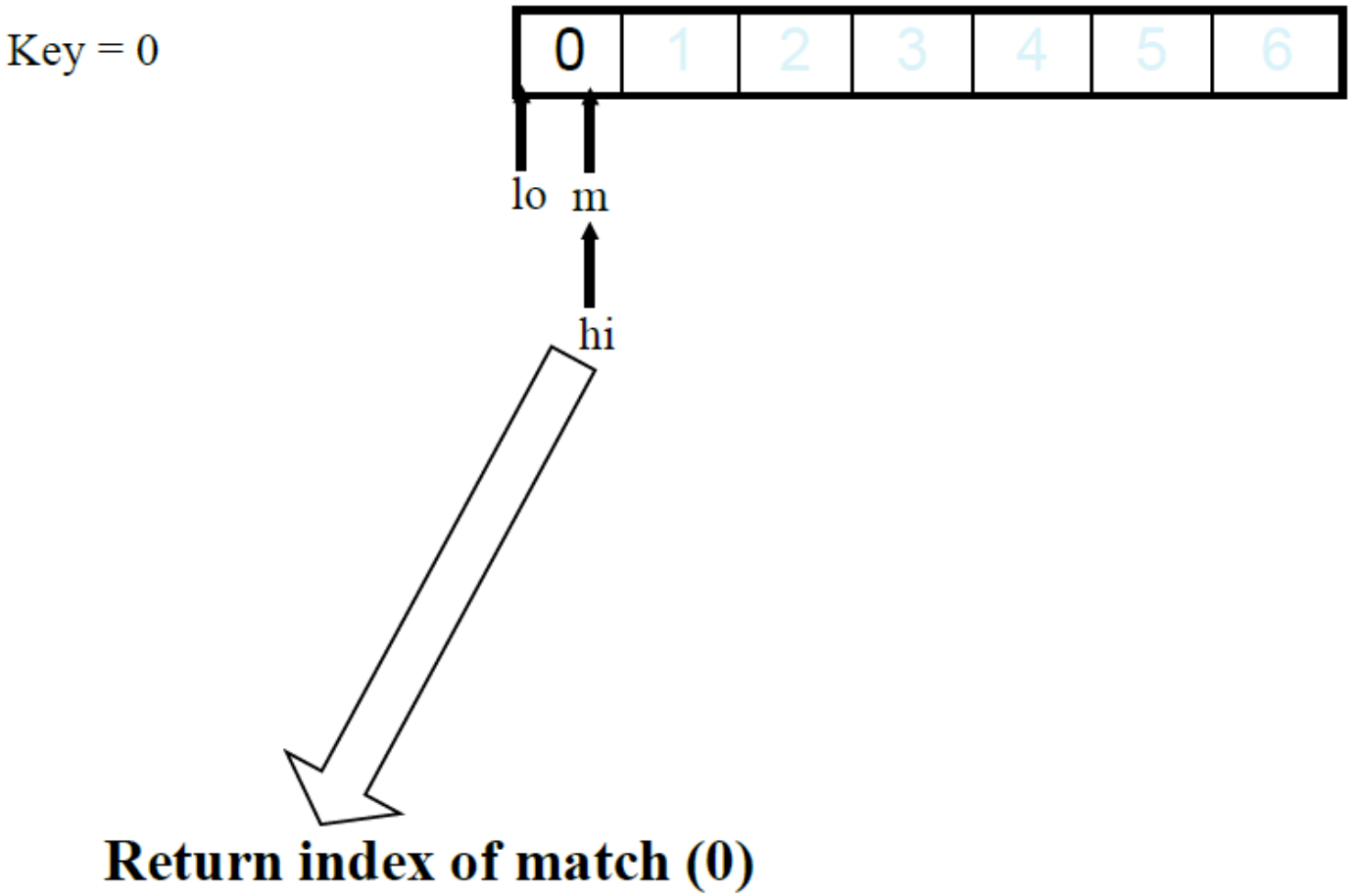
0 > 0?

Binary Search (Cont'd)

Key = 0



Binary Search (Cont'd)



Binary Search (Cont'd)

- Java code (iterative implementation):

```
int binarySearch(int[] arr, int key)
{
    int lo= 0, mid, hi = arr.length - 1;
    while (lo <= hi) {
        mid = (lo + hi) / 2;
        if (key < arr[mid])
            hi = mid - 1;
        else if (arr[mid] < key)
            lo = mid + 1;
        else return mid;          // success: key found
    }
    return -1;  // failure: key not found
}
```

Binary Search (Cont'd)

- Java code (recursive implementation):

```
int binarySearch(int[] arr, int first, int last, int key)
{
    if (first <=last)
        int mid = (first + last)/2;
    if (key == arr[mid])
        return mid;      // success: key found
    else if (key < arr[mid])
        return binarySearch(arr, first, mid-1, key);
    else if (arr[mid] < key)
        return binarySearch(arr, mid+1, last, key);
    }
    return -1;  // failure: key not found
}
```

Binary Search (Cont'd)

- Efficiency:
  - Best case: *O(1)*
  - Worst case: *O(lg n)*
    - If we double n, this adds only 1 more comparison
  - Average case: *O(lg n)*
    - In practice, about 2 times faster than worst case
- Is far better than sequential search, especially for lists larger than about 50 items

Interpolation Search

- Is a variant of binary search, where the “midpoint” is set to where the item is likely to occur
  - E.g. If searching a telephone book for “Farr”, we would start looking ~ ¼ through
- Assumes that the data in the array is sorted
  - Caution: is rarely true

Interpolation Search (Cont'd)

- **Substitute:**

```
mid = (lo + hi) / 2
```

with:

```
p = (key - arr[lo]) / (arr[hi] - arr[lo]);
mid = lo + ceiling((hi - lo) * p);
```

- In practice, is little better than the binary search because of the interpolation calculations

Summary

- *Sequential search* is the simplest search to implement but it's not efficient on big number of data.
- *Binary search* is a more efficient algorithm than sequential search but it needs that data be sorted.
- *Interpolation search* is a variant of binary search. It tries to find a better position to divide data and then search among them.

Review Questions

- What is the difference between primary and secondary key?
- What are data, record and field?
- Explain sequential search algorithm.
- What are time complexities of sequential search in best, average, and worst case?
- Explain binary search algorithm.
- What are time complexities of binary search in best, average, and worst case?
- Explain interpolation search algorithm.
- What are time complexities of interpolation search in best, average, and worst case?

Any questions?