SEARCHING ALGORITHMS

INSTRUCTOR: KASHFIA SAILUNAZ

SLIDES ADAPTED FROM THE TEXTBOOK & ENSF 593/594 LECTURE03 BY MOHAMMAD MOSHIRPOUR

OUTLINE

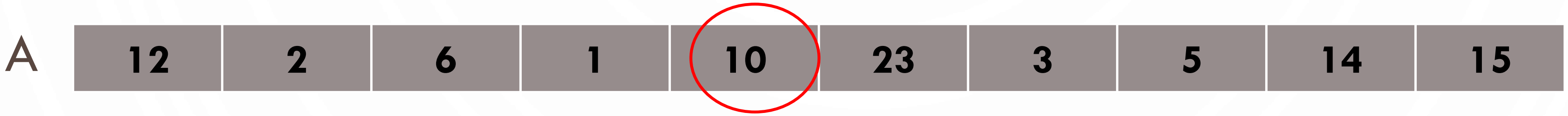
- Searching Algorithms
- Definitions
- Types
- Sequential/Linear Search
- Binary Search
- Interpolation Search

LEARNING OUTCOME

- At the end of this lecture, we will be able to-
  - Understand how search algorithms work, and
  - Define and execute different types of searching algorithms.

SEARCHING ALGORITHMS

- Algorithms to search or retrieve any elements in a data structure



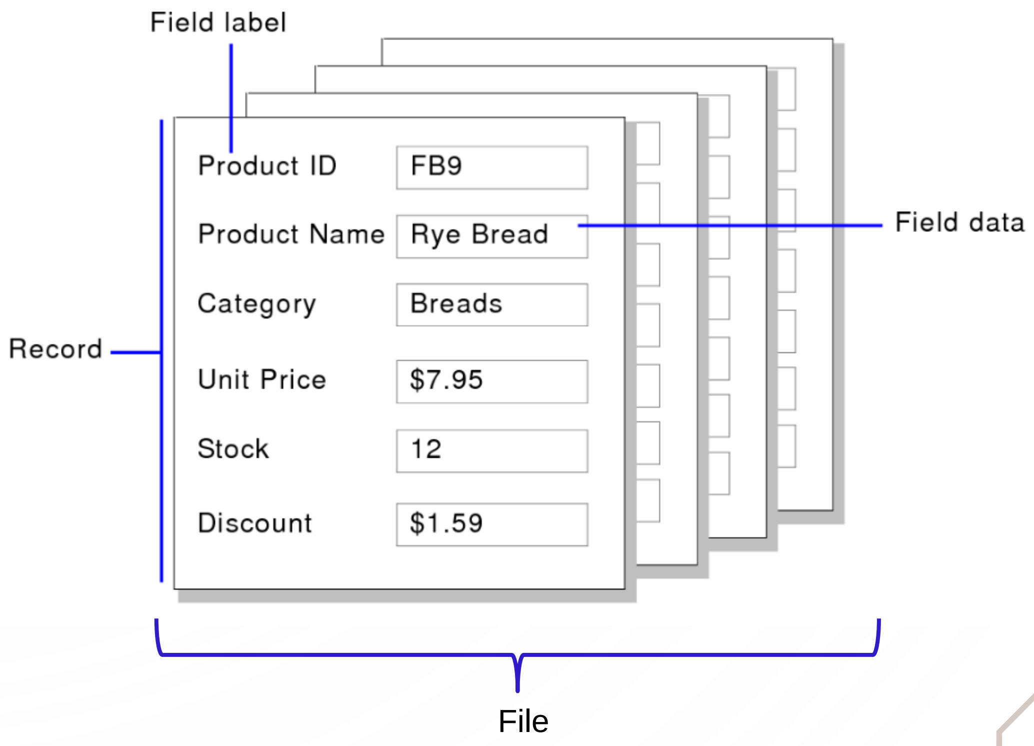
A									
12	2	6	1	10	23	3	5	14	15

Search for 10 - FOUND

Search for 4 - NOT FOUND

DEFINITIONS

- Data
  - Any independent observation or fact
- Record/Element
  - Data representing a particular object
  - Contains one or more fields
- Field
  - A specific part of a record
  - Contains one data component with type and size
- File
  - A collection of records

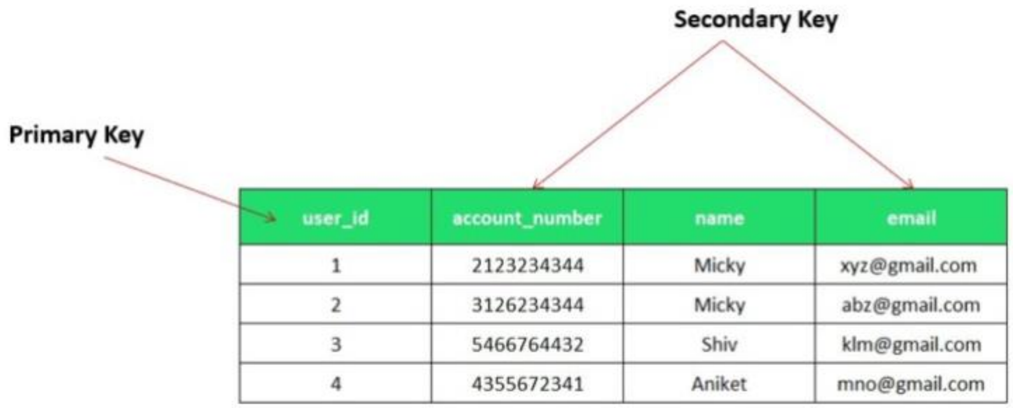


Product ID	FB9
Product Name	Rye Bread
Category	Breads
Unit Price	$7.95
Stock	12
Discount	$1.59

Source: https://fmhelp.filemaker.com/help/16/fmp/en/index.html#page/FMP_Help/adding-viewing-data.html

DEFINITIONS

- Key
  - Data field used to select or order records
  - May or may not be unique
- Primary Key
  - A field(s) used as a unique identifier of the records
- Secondary Key
  - A field used if more than one record have same values for primary key


Secondary Key
Primary Key

user_id	account_number	name	email
1	2123234344	Micky	xyz@gmail.com
2	3126234344	Micky	abz@gmail.com
3	5466764432	Shiv	klm@gmail.com
4	4355672341	Aniket	mno@gmail.com

Source: https://programmerbay.com/difference-between-primary-key-and-secondary-key/

DEFINITIONS

- Search
  - An operation that returns the pointer to a record (if matches a key value) or null (if there is no match)

A	**12**	**2**	**6**	**1**	**10**	**23**	**3**	**5**	**14**	**15**
index	0	1	2	3	4	5	6	7	8	9

10 FOUND at index 4.

- Sort
  - An operation that arranges the elements in an order (ascending or descending)

A (ascending)	**1**	**2**	**3**	**5**	**6**	**10**	**12**	**14**	**15**	**23**
A (descending)	**23**	**15**	**14**	**12**	**10**	**6**	**5**	**3**	**2**	**1**

TYPES OF SEARCHING ALGORITHMS

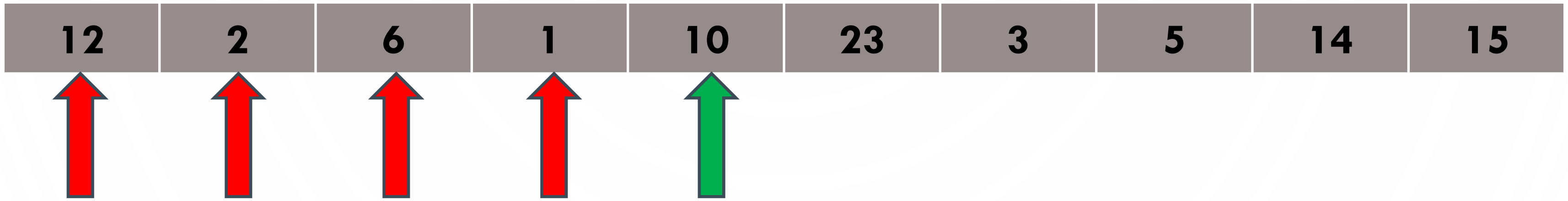
- Sequential/Linear
- Binary
  - Interpolation (a variant of binary)

SEQUENTIAL/LINEAR SEARCH

- A linear/sequential search starts from the beginning of the list of components and compares each item in a sequence to check until it matches to the search query key or the list ends
- Applied on arrays and linked lists
- Items can be sorted or unsorted
- Faster searching algorithm for small lists
- Simple to implement
- Not very efficient for larger lists

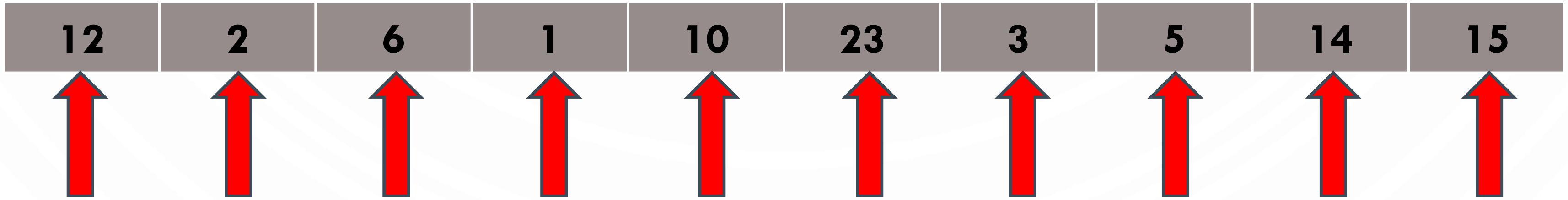
SEQUENTIAL/LINEAR SEARCH

Search key = 10



12	2	6	1	10	23	3	5	14	15

Search key = 20

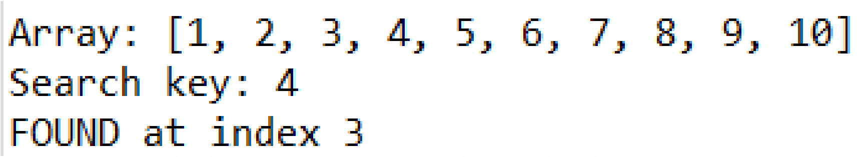


12	2	6	1	10	23	3	5	14	15

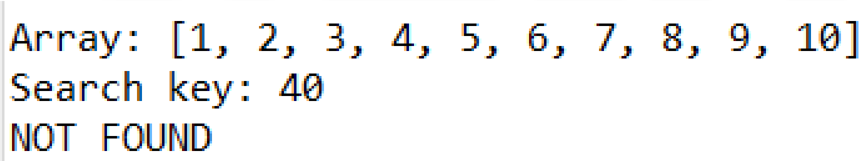
SEQUENTIAL/LINEAR SEARCH

- Sample Code in Java:

```java
int linearSearch(int[] array, int key)
{
        for(int i = 0; i < array.length; i ++)
        {
                if(array[i] == key)
                        return i; //key FOUND and index returned
        }
        return -1; //key NOT FOUND and -1 returned
}
```



```
Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Search key: 4
FOUND at index 3
```



```
Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Search key: 40
NOT FOUND
```

SEQUENTIAL/LINEAR SEARCH

- Best case
  - Search item matches the first array element
  - Only 1 comparison
- Worst case
  - Search item matches the last array element
  - n comparisons
- Average case
  - On average (n+1)/2 comparisons
- Complexity: O(n)

BINARY SEARCH

- Searches items in only sorted arrays/lists
- Can be solved both iteratively and recursively
- Algorithm:
  - Input: Sorted array, Search key
  - Divide the array into two parts by finding the middle ‘mid’ item
    - If the key matches ‘mid’, return
  - If key < ‘mid’, then divide the left subarray into two parts following the previous steps
  - If key > ‘mid’, then divide the right subarray into two parts following the previous steps
  - Keep diving the subarrays until the key is found OR no more division is possible

- mid = (low + high)/2

BINARY SEARCH

- Search key = 10

1	2	3	5	6	10	12	14	15
low				mid				high

10=6?

10>6?

10<6?

1	2	3	5	6	10	12	14	15
					low	mid		high

10=12?

10>12?

10<12?

BINARY SEARCH

- Search key = 10

1	**2**	**3**	**5**	**6**	**10**	**12**	**14**	**15**
					low	mid		high

10=12?
10>12?
10<12?

1	**2**	**3**	**5**	**6**	**10**	**12**	**14**	**15**
					low mid	high		

10=10?
10>10?
10<10?

Return index of 10 (index: 5)

BINARY SEARCH

- Sample Code in Java (Iterative Approach):

```
int binarySearchIterative(int[] array, int key)
{
        int low = 0, mid, high = array.length-1;
        while (low <= high)
        {
                mid = (low + high)/2;
                if(key < array[mid])
                        high = mid - 1;
                else if(array[mid] < key)
                        low = mid + 1;
                else
                        return mid; //key FOUND and index returned
        }
        return -1; //key NOT FOUND and -1 returned
}
```

BINARY SEARCH

- Sample Code in Java (Recursive Approach) :

```java
int binarySearchRecursive(int[] array, int low, int high, int key)
{
        if(low <= high)
        {
                int mid = (low + high)/2;
                if(key == array[mid])
                        return mid; //key FOUND and index returned
                else if(key < array[mid])
                        return binarySearchRecursive(array, low, mid-1, key);
                else if(array[mid] < key)
                        return binarySearchRecursive(array, mid+1, high, key);
        }
        return -1; //key NOT FOUND and -1 returned
}
```

BINARY SEARCH

- Best case
  - Found the item in the middle of the array - O(1)
- Worst case
  - Found the item at the last subdivision – O(lg n)
- Average case
  - Possibly about 2 times faster than worst case – O(lg n)
- Complexity : O(lg n)

INTERPOLATION SEARCH

- A variant of binary search
- Assumes items are sorted
  - That may or may not be true for the given arrays/lists
- Sets the midpoint according to the search key by finding a location where the search key is most likely to exist
- Finds a probe position instead of mid position
- Various interpolation formula are used
- One possible formula:

pos = (key - array[low]) / (array[high] - array[low])

mid = low + [((high - low) * pos)]

INTERPOLATION SEARCH

- Can be solved both iteratively and recursively
- Algorithm:
  - Divide the array into two parts by finding the probe position 'mid' using the formula
    - If the key matches the 'mid' item, return
  - If key < 'mid' item, then divide the left subarray into two parts following the previous steps
  - If key > 'mid' item, then divide the right subarray into two parts following the previous steps
  - Keep diving the subarrays until the key is found OR no more division is possible

INTERPOLATION SEARCH

- Sample Code in Java (Iterative Approach & Recursive Approach)

Try it yourself

INTERPOLATION SEARCH

- Best case
  - Found the item in first mid - O(1)
- Worst case
  - Found the item at the last subdivision – O(n)
- Average case
  - With uniform distribution of data – O(lg (lg n))

- Complexity : O(n)

SUMMARY

- Different searching algorithms have their own advantages and limitations and should be chosen based on the input size, type, distributions, etc.
- Sequential/linear search is the simplest searching algorithm and works well with small amount of data, but not efficient for larger dataset.
- Binary search performs better with large number of data compared to linear search, but the data need to be sorted.
- Interpolation search is a variant of binary search and works by trying to find better division points of the data to accelerate the search process.

THANK YOU