SCJ2013 Data Structure & Algorithms

Searching Techniques: Sequential Search

Nor Bahiah Hj Ahmad & Dayang Norhayati A. Jawawi

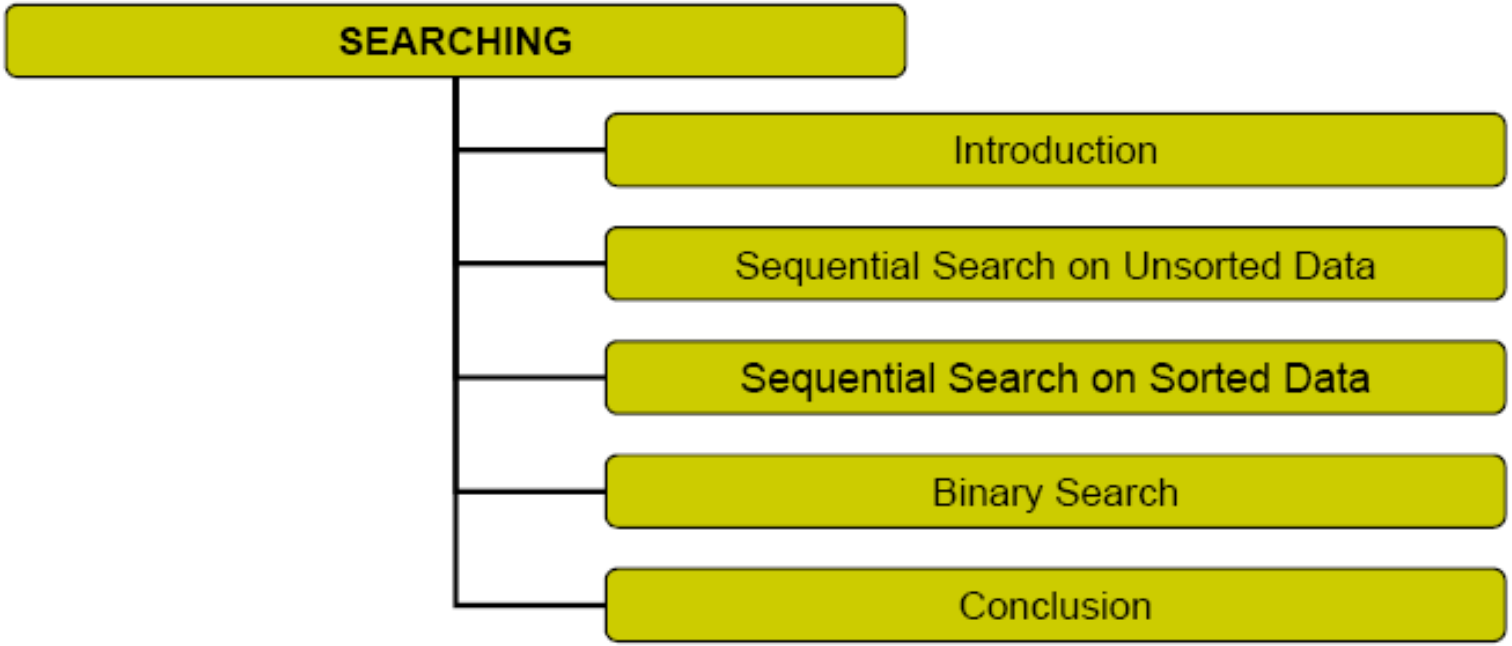
Objectives

At the end of the class, students are expected to be able to do the following:

- Understand the **searching technique concept** and the purpose of searching operation.
- Understand the implementation of basic searching algorithm;
  1. Sequential search.
     - Sequential search on unsorted data.
     - Sequential search on sorted data.
  2. Binary Search.
- Able to analyze the **efficiency** of the searching technique.
- Able to **implement** searching technique in problem solving.

1.0 Introduction

Class Content



Introduction

- **Searching Definition**
  - Clifford A. Shaffer[1997] define searching as a process to determine whether an element is a **member** of a certain data set.
  - The process of **finding the location** of an element with a specific value (key) within a collection of elements
  - The process can also be seen as an attempt to **search** for a certain record in a file.
    - Each record contains **data field** and **key field**
    - **Key field** is a group of characters or numbers used as an **identifier** for each record
    - Searching can done based on the key field.

Example: Table of Employee Record

Index	employeeID	employeeIC	empName	Post
[0]	1111	701111-11-1234	Ahmad Faiz Azhar	Programmer
[1]	122	800202-02-2323	Mohd. Azim Mohd. Razi	Clerk
[2]	211	811003-03-3134	Nurina Raidah Abdul Aziz	System Analyst

Searching can be done based on certain field:

empID, or **empl_IC**, or **empName**

To search **empID** = 122, give us the record value at index 1.

To search **empID** = 211, give us the record value at index 1.

Introduction

- Among Popular searching techniques:
  - **Sequential** search
  - **Binary** Search
  - **Binary Tree** Search
  - **Indexing**

- Similar with sorting, Searching can also be implemented in two cases, **internal** and **external** search.

Introduction

- Similar with sorting, Searching can also be implemented in two cases, **internal** and **external** search.
  - **External search** – only implemented if searching is done on a **very large size** of data. Half of the data need to be processed in **RAM** while half of the data is in the **secondary storage**.
  - **Internal search** – searching technique that is implemented on a **small size** of data. All data can be load into **RAM** while the searching process is conducted.

The data stored in an array

2.0 Basic Sequential Search

Basic Sequential Search

- Basic sequential search usually is implemented to search item from **unsorted** list/ array.
- The technique can be implemented on a **small size** of list. This is because the **efficiency** of sequential search is **low** compared to other searching techniques.
- In a sequential search:
  1. Every element in the array will be **examine sequentially**, starting from the first element.
  2. The process will be **repeated** until the **last element** of the array or until the searched data is **found**.

Basic Sequential (BS) Search

- Used for searching that involves records stored in the main memory (RAM)
- The **simplest search algorithm**, but is also the slowest
- Searching strategy:
  1. **Examines** each element in the array one by one (sequentially) and compares its value with the one being looked for – the search key
  2. Search is successful if the search key **matches** with the value being compared in the array. Searching process is **terminated**.
  3. else, if no matches is found, the search process is **continued to the last** element of the array. Search is **failed** array if there is no matches found from the array.

Basic Sequential Search Function

```
int SequenceSearch( int   search_key,
                    const int array [ ],
                    int   array_size )
{   int p;
    int index =-1; //-1 means record is not found
    for ( p = 0; p < array_size; p++ ){
        if ( search_key == array[p] ){
            indeks = p;//assign current array index
            break;
        }//end if
      } //end for
   return index;
} //end function
```

Every element in the array will be examined until the search key is found

Or until the search process has reached the last element of the array

BS Search implementation – Search key = 22

```
int SequenceSearch( int   search_key,
                    const int array [ ],
                    int   array_size )
{   int p;
    int index =-1;
    //-1 means record is not found
    for ( p = 0; p < array_size; p++ ){
       if ( search_key == array[p] ){

           indeks = p;
          //assign current array index
           break;
       }//end if
     } //end for
   return index;
} //end function
```

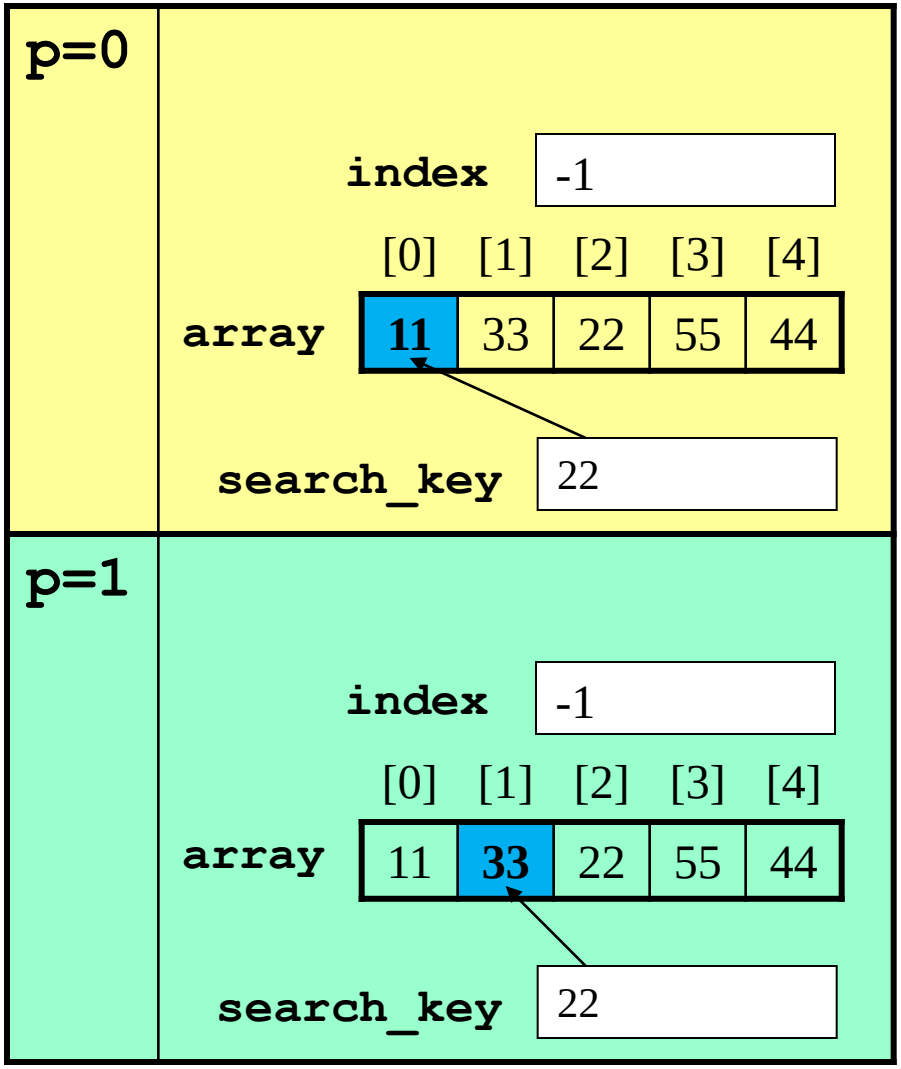
index -1

	[0]	[1]	[2]	[3]	[4]
array	11	33	22	55	44

search_key 22

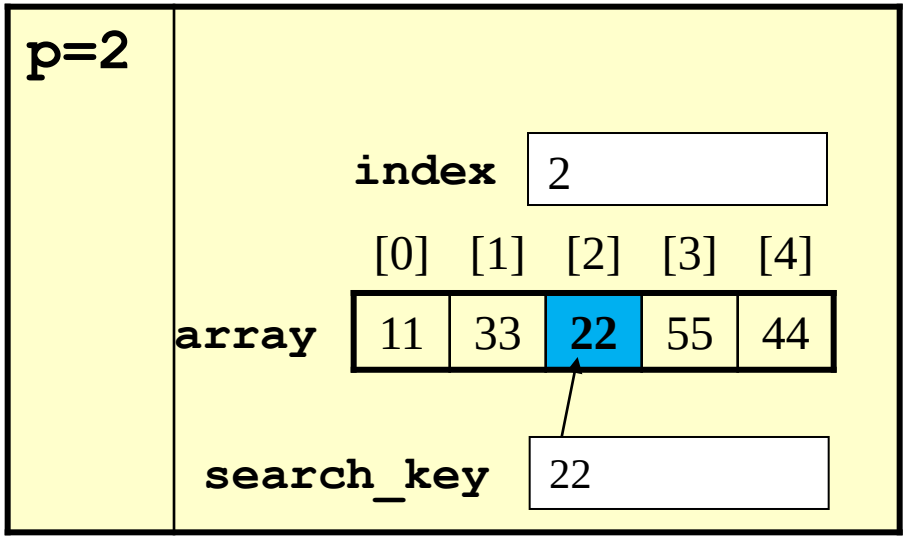
BS Search implementation – Search key = 22

```
int SequenceSearch( int   search_key,
                    const int array [ ],
                    int   array_size )
{ int p;
  int index =-1;
  //-1 means record is not found
  for ( p = 0; p < array_size; p++ ){
     if ( search_key == array[p] ){
          indeks = p;
         //assign current array index
          break;
    }//end if
  } //end for
   return index;
} //end function
```



BS Search implementation – Search key = 22

```
int SequenceSearch( int   search_key,
                    const int array [ ],
                    int   array_size )
{ int p;
  int index =-1;
  //-1 means record is not found
  for ( p = 0; p < array_size; p++ ){
     if ( search_key == array[p] ){
         indeks = p;
        //assign current array index
         break;
    }//end if
  } //end for
   return index;
} //end function
```

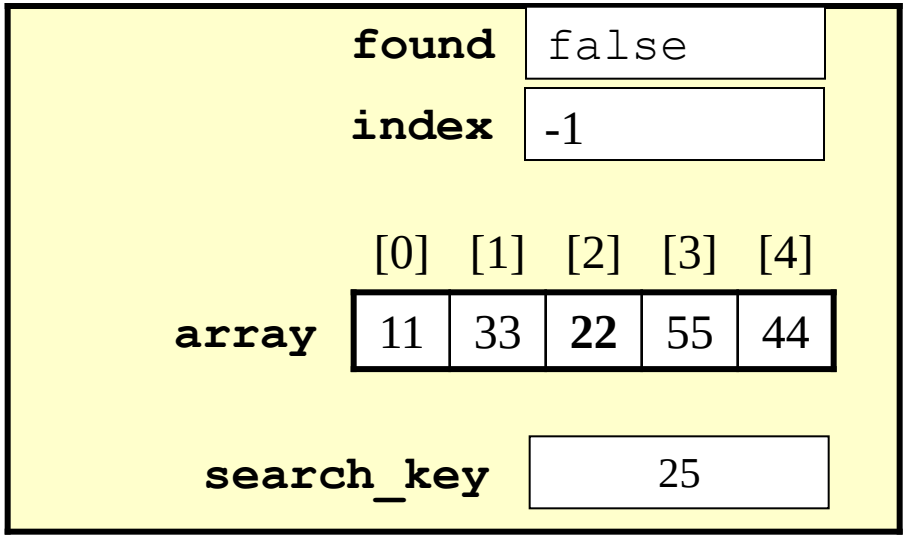


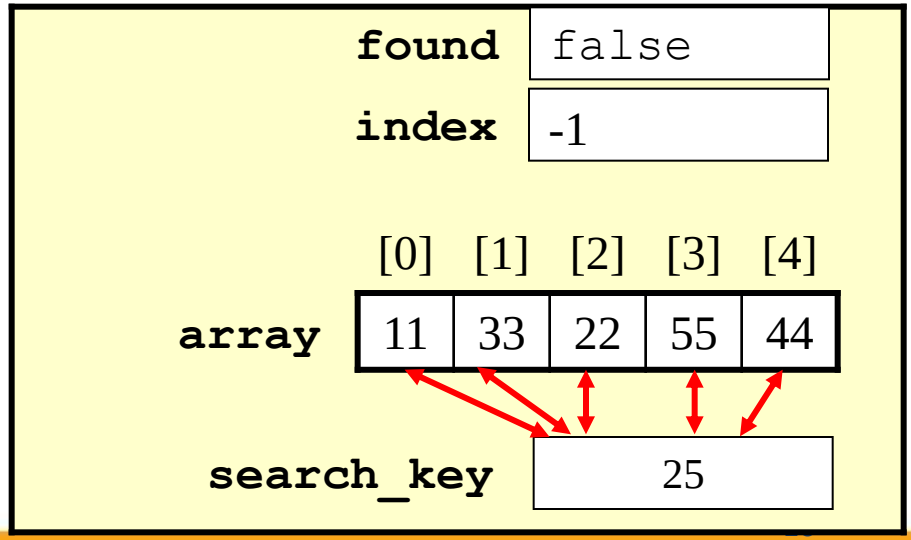
Search for key 22 is successful
&
return 2

BS Search implementation – Search key = 25

```
int SequenceSearch( int   search_key,
                    const int array [ ],
                    int   array_size )
{ int p;
  int index =-1;
  //-1 means record is not found
  for ( p = 0; p < array_size; p++ ){
     if ( search_key == array[p] ){
         indeks = p;
        //assign current array index
         break;
    }//end if
  } //end for
   return index;
} //end function
```

p=0,1,2,3,4 => search key is not matches
Search is unsuccessful





Sequential Search Analysis

- Searching time for sequential search is O(n).
- If the searched key is located at the end of the list or the key is not found, then the loop will be repeated based on the number of element in the list, O(n).
- If the list can be found at index 0, then searching time is, O(1).

Improvement of Basic Sequential Search Tech.

- **Problem:**
  - Search key is compared with all elements in the list, **O(n)** time consuming for large datasets.

- **Solution:**
  - The efficiency of basic search technique can be improved by searching on a **sorted list**.
  - For searching on ascending list, the search key will be compared one by one until :
    1. the searched key is **found**.
    2. Or until the searched **key value is smaller than the item compared** in the list.

=> This will minimize the searching process.

Sequential Searching on Sorted Data

```
int SortedSeqSearch ( int search_key, const int
array[ ],
                int array_size)
{   int p;
    int index = -1; //-1 means record not found
    for ( p = 0; p < array_size; p++ )
    {   if (search_key < array [p] )
            break;
           // loop repetition terminated
           // when the value of search key is
           // smaller than the current array element
        else if (search_key == array[p])
        {
            index = p;  // assign current array index
            break;
        } // end else-if
    }//end for
    return index; // return the value of index
} //end function
```

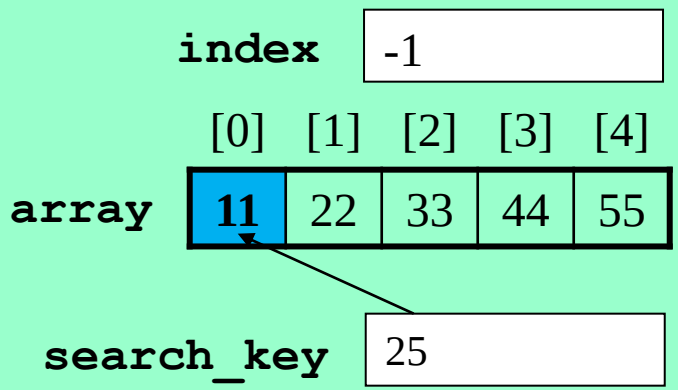
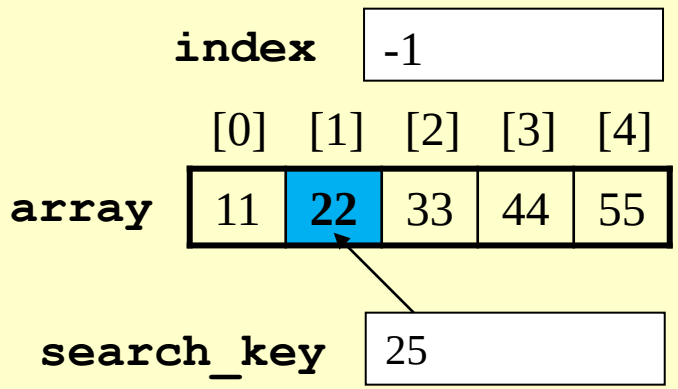
Steps to Execute Sequential Search Function on a Sorted List

Assume:

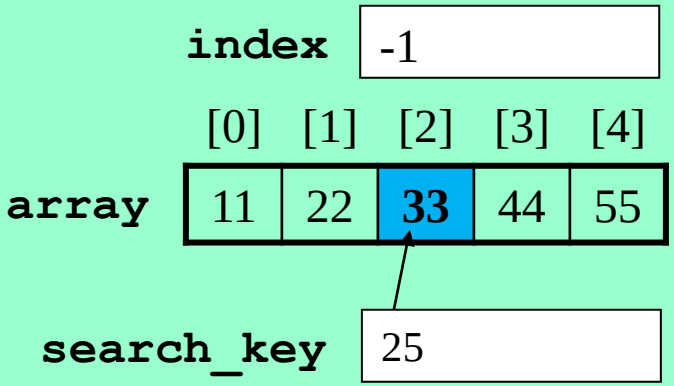
- **`search_key`** = 25
- **`array_size`** = 5

Step 1	**index** -1 [0] [1] [2] [3] [4] **array** 11 22 33 44 55 **search_key** 25	Initial value for variable **`index`** and **`array`** elements

Steps to Execute Sequential Search Function on a Sorted List

Step 2		p = 1 **search_key** is compared with the first element in the **array**
Step 3		p = 2 **search_key** is compared with the second element in the **array**

Steps to Execute Sequential Search Function on a Sorted List

Step 4		p = 3 **`search_key`** is compared with the third element in the **`array`** • the value of **`search_key`** is smaller than the current element of array • loop repetition is terminated using "**`break`**" statement

Steps to Execute Sequential Search Function on a Sorted List

- **Conclusion:**
  - If the elements in the list is not in a sorted (asc/desc) order, loop will be repeated based on the number of elements in the list
  - When the list is not sorted the loop is repeated 5 times, compared to 3 times if the list is in sorted order as shown in the previous example.
  - If the list is sorted in descending order, change operator **"<"** to operator **">"** in the loop **`for`**