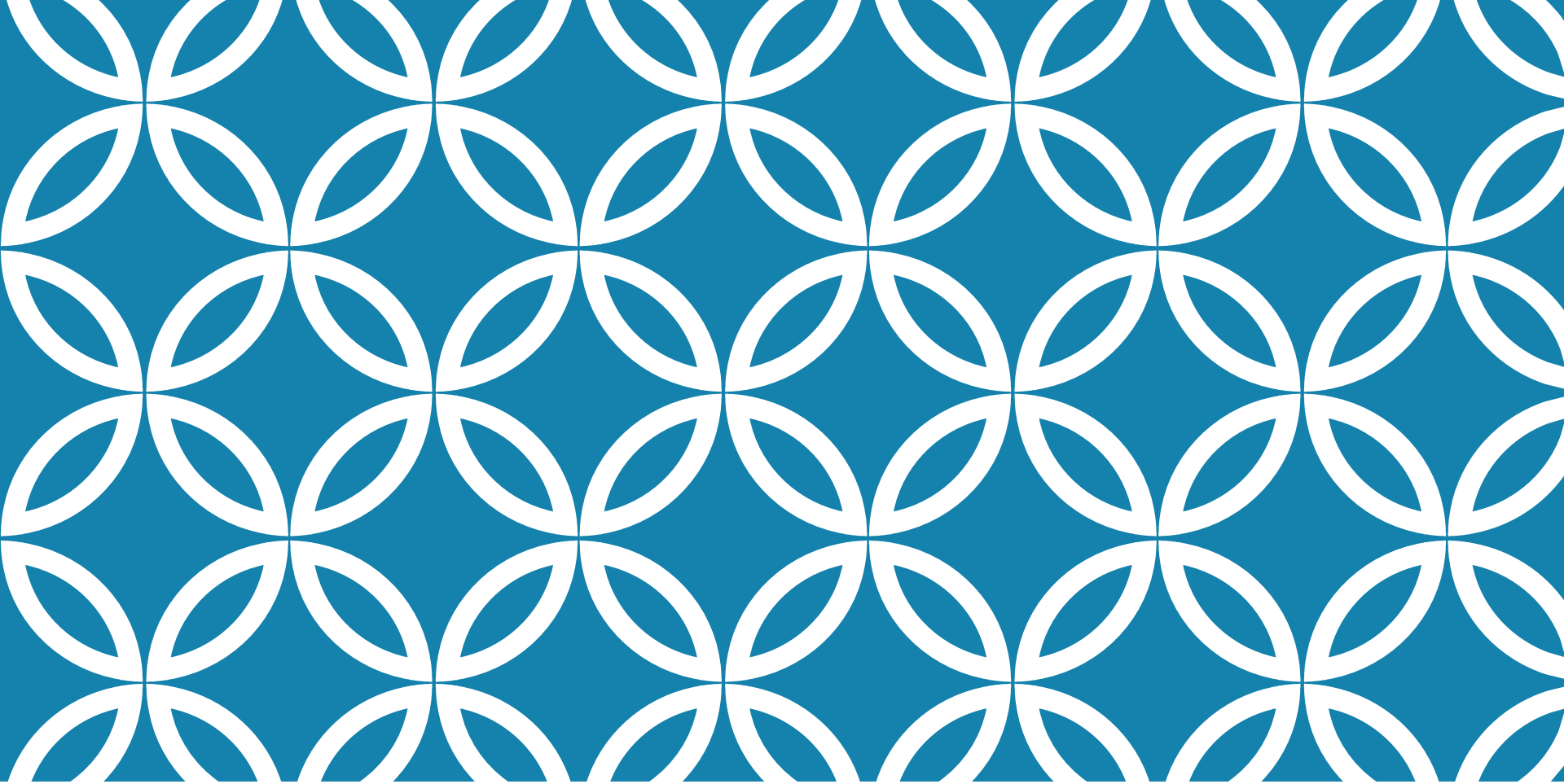


AST20105 DATA STRUCTURES & ALGORITHMS

CHAPTER 1 – INTRODUCTION TO DATA STRUCTURES & ALGORITHMS

Garret Lai

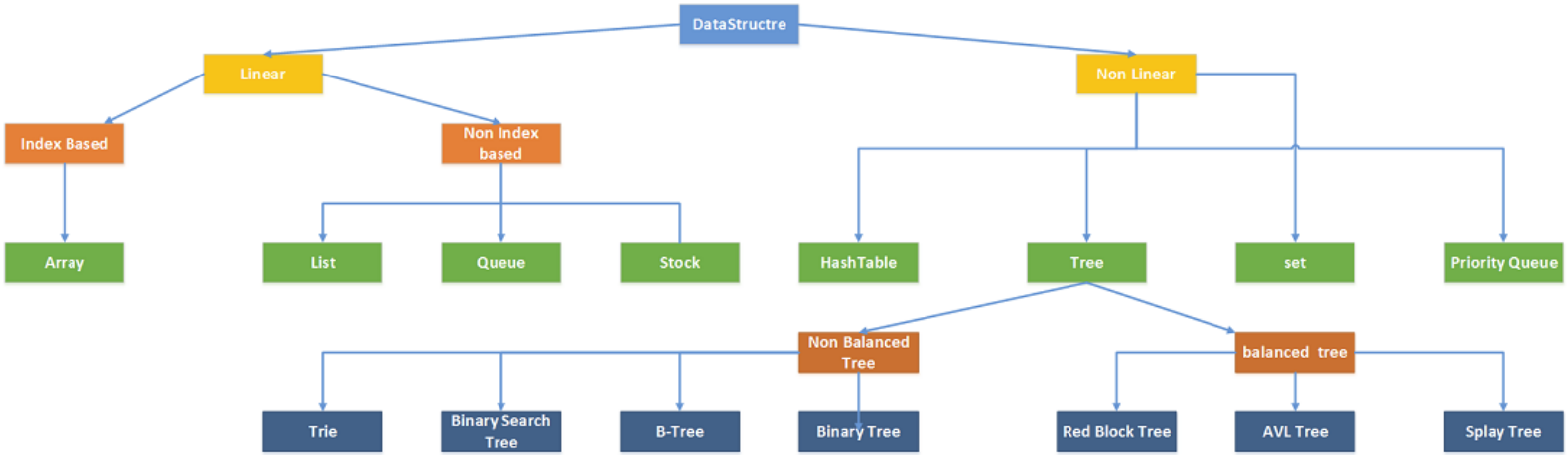
DATA STRUCTURE IS

A data structure is an **arrangement of data** in a computer's memory or even disk storage.

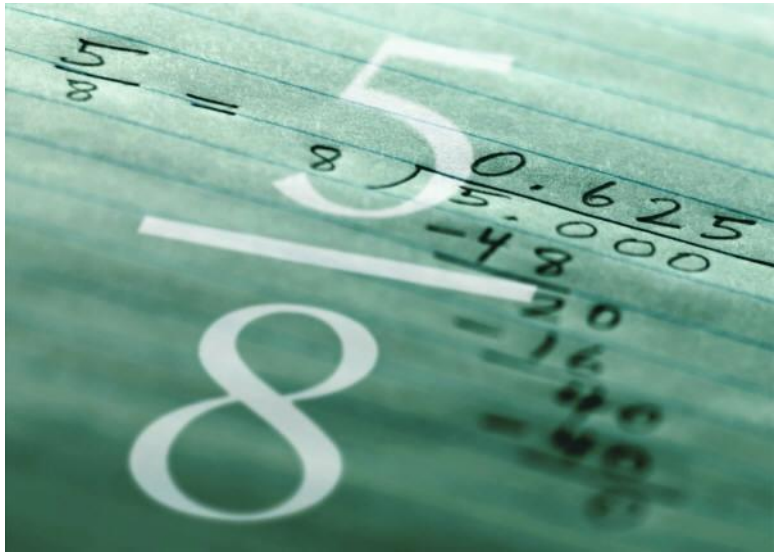
An example of several common data structures are

- arrays,
- linked lists,
- queues,
- stacks,
- binary trees, and
- hash tables.

TYPES OF DATA STRUCTURE



WHAT IS ALGORITHMS?



ALGORITHM IS

Algorithms are used to **manipulate the data** contained in the data structures as in

- Searching and sorting
- Insertion, deletion, and modification of data

DATA STRUCTURES & ALGORITHMS

Computer Program = Data Structures + Algorithms

DATA STRUCTURES

EXAMPLE OF DATA STRUCTURES

Name	**Relationship**
Alan	David's father
Alex	John's son
Betty	Alan's daughter
Candy	Nick's wife
David	Betty's brother
John	Alan's brother
Lisa	Alex's sister
Nick	Betty's grandfather
Ricky	Lisa's son
Susan	Betty's mother

EXAMPLE OF DATA STRUCTURES

From the table above

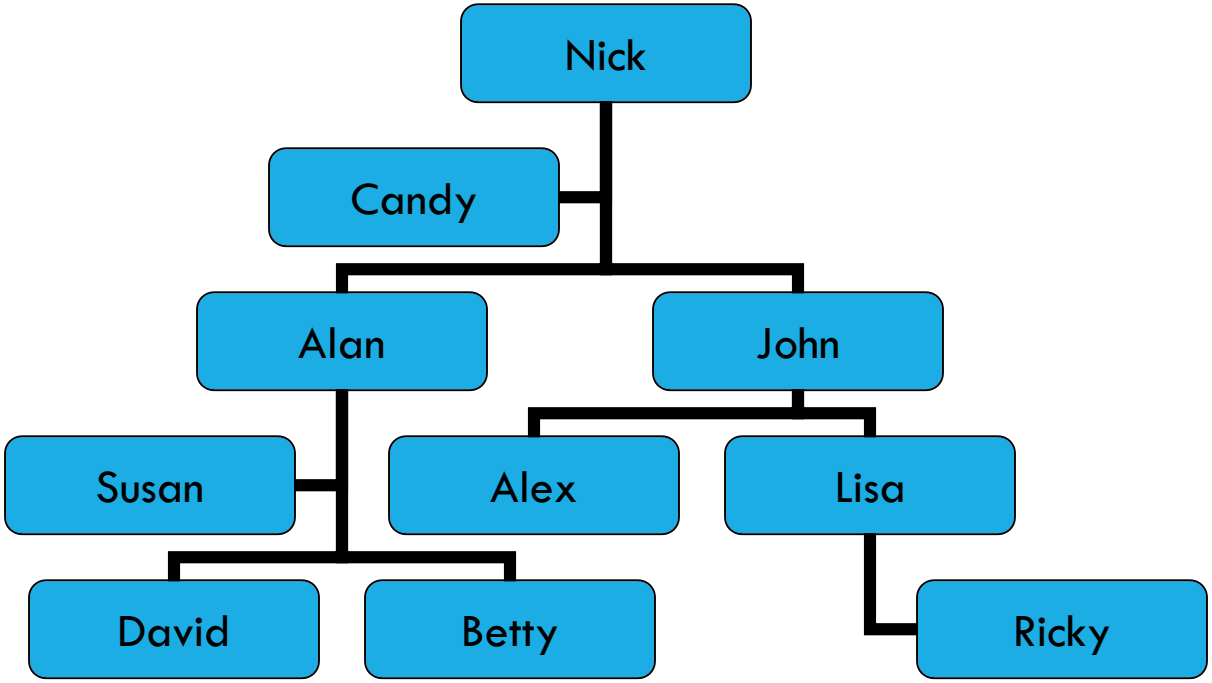
- Do you know the relationship between Alex and Nick?
- Can you find out the relationship between Candy and Ricky easily?

EXAMPLE OF DATA STRUCTURES

In this example

- You may think that the **tree diagram** is much **better** structure for showing the relationships

EXAMPLE OF DATA STRUCTURES



EXAMPLE OF DATA STRUCTURES

The list and the diagrams are examples of the **different data structures**

- The name can be stored in alphabetical order so that
  - We can locate the people very quickly
- The tree structure is a much better suited for showing the relationships

EXAMPLE OF DATA STRUCTURES

The data structures are important in **organizing information** in a computer

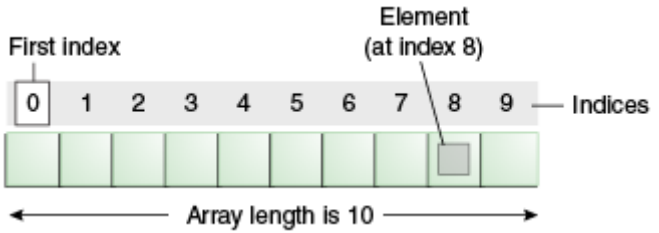
- Some of them are similar to the tree diagram
  - Good for **REPRESENTING RELATIONSHIPS** between different data
- Others are good for **ORDERING DATA** in a particular way like the list shown above

Each data structure has their own unique properties that make it well suited to give a certain view of the data

STORAGE CONTAINER

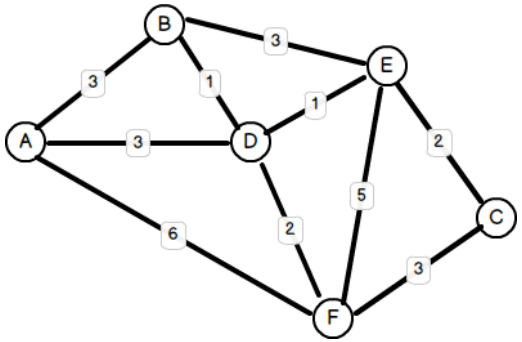
Linear Container

- Rooms in the container are like along a **line**
- Data are
  - Stored on the line and
  - Be scanned along the line



Non-linear Container

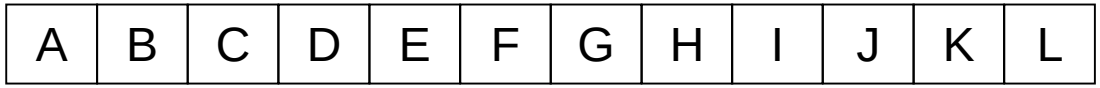
- Rooms in the container may be random on a line or on 2 or more dimensional space
- Data are
  - Stored and scanned with special algorithms



LINEAR CONTAINER

Array

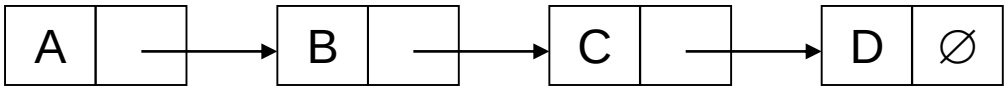
- A very basic data structure representing a group of similar elements, accessed by **INDEX**.
- Array data structure can be effectively stored inside the computer and provides **FAST ACCESS(?)** to the all its elements.



LINEAR CONTAINER

Linked List

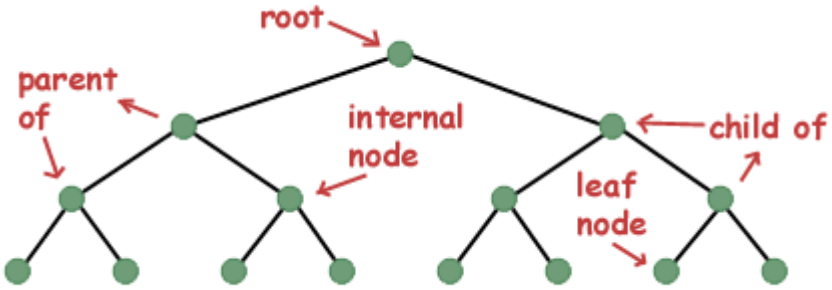
- Linked list is a very important dynamic data structure.
- Every element contains some data and a link to the next element, which allows to keep the structure.

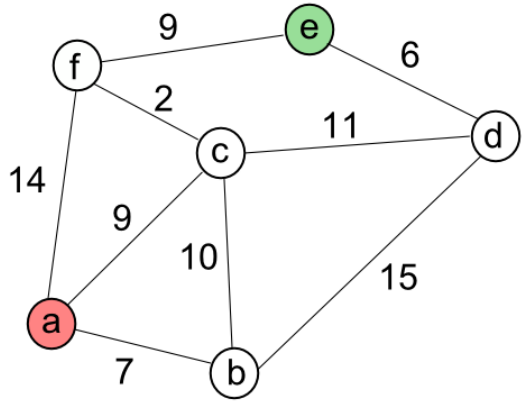


NON-LINEAR CONTAINER

Tree/Graph structures

- Rooms of the container are like a tree/graph
  - Which contains nodes and branches/edges
- Follow the branches of the tree to store and scan data

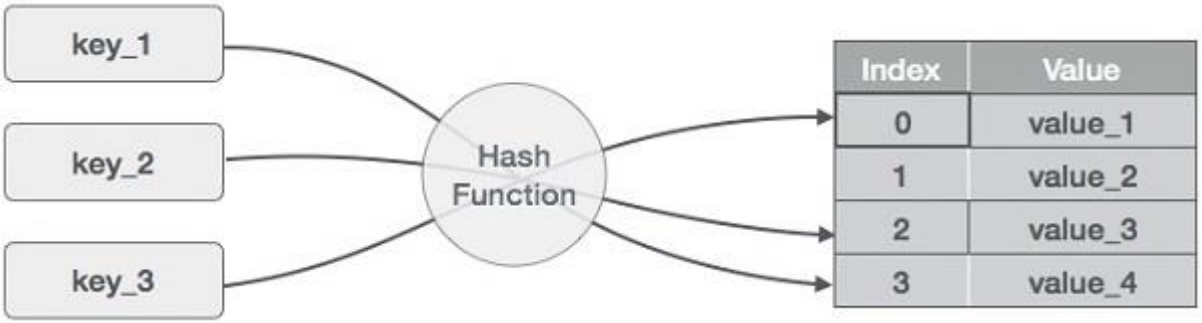




NON-LINEAR CONTAINER

Formula based containers

- Rooms of the container may be linear or any other kinds
- Use formula to calculate the position where the data should be stored



ALGORITHM ANALYSIS

ALGORITHM ANALYSIS

What is an algorithm?

Why do we want to analyze one?

ALGORITHM ANALYSIS

An algorithm is a **step-by-step procedure** for accomplishing some end.

ALGORITHM ANALYSIS

An algorithm can be given in many ways.

- For example, it can be written down in English (or French, or any other "natural"' language).
- However, we are interested in algorithms which have been precisely specified using an appropriate mathematical formalism - such as **a programming language**.

EXAMPLE OF ALGORITHM

Searching a book in library

- Say, you are searching the textbook
- Input is?
- Output is?
- Procedures are?

EXAMPLE OF ALGORITHM

Algorithm 1

- Input: Title/Author/Keywords/Call number
- Output: Get the book
- Procedures:
  1. Go to the library
  2. Scan all books in the library from the leftmost bookshelf
  3. Compare the book information with the input
  4. If they are matched, get the book. Otherwise, compare with the next book

EXAMPLE OF ALGORITHM

Algorithm 2

- Input: Title/Author/Keywords/Call number
- Output: Get the book
- Procedures:
  1. Go to the library
  2. Use a computer to find the call number of the book
  3. Directly go to the bookshelf stated in the call number
  4. Compare the book information with the input
  5. If they are matched, get the book. Otherwise, compare with the next book

ALGORITHM ANALYSIS

In order to learn more about an algorithm, we can "**analyze**" it.

By this we mean to study the **specification** of the algorithm and to **draw conclusions** about how the implementation of that algorithm - the program - will perform in general.

WHAT CAN WE ANALYSIS

Determine the **RUNNING TIME** of a program as a function of its inputs;

Determine the **total or maximum MEMORY SPACE** needed for program data;

Determine the **total size** of the program code;

WHAT CAN WE ANALYSIS

Determine whether the program **correctly computes** the desired result;

Determine the **complexity** of the program

- e.g., how easy is it to read, understand, and modify;

Determine the **robustness** of the program

- e.g., how well does it deal with unexpected or erroneous inputs?

ALGORITHM ANALYSIS

In this course,

- We are concerned primarily with the **RUNNING TIME***.
- We also consider the **MEMORY SPACE** needed to execute the program.

BE REMINDED!!!

- Running Time here we will mention IS NOT the exact amount of time needed to operate a program.
- What we are concerned are:
  - How is the running time of a program being affected if the size of data (n) is getting bigger and bigger? (So, we are talking about performance tendency)
  - Which part of program will play significant role for difference size of n?

WHY STUDY DATA STRUCTURES AND ALGORITHMS?

WHY STUDY DATA STRUCTURES AND ALGORITHMS?

Most computer systems (essentially computer programs) spend their time doing storing, accessing, and manipulating data

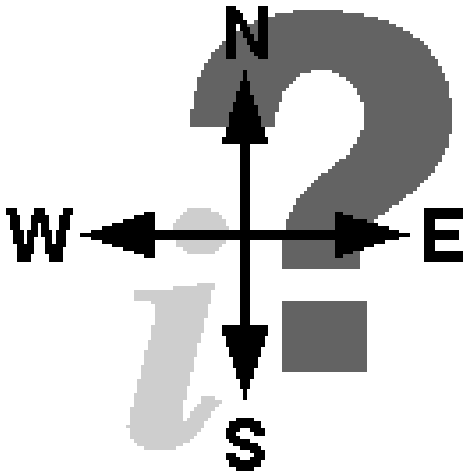
- Networking
  - What paths should a message be sent?
  - How can this set of paths could be determined?
- Information retrieval
  - Search engine like Google store and search contents of web
- Computer Graphics
  - Designing a virtual reality system for an walkthrough of building. Given the location of the viewer, what portions of the building are visible to the viewer?

If data are stored and organized in a clever manner, then data could be accessed and manipulated efficiently

→ Efficient systems / programs

CHOICE OF DATA STRUCTURES

- Different data structures have different strengths and weaknesses and we never got a data structure that are good in all cases
- Selecting the most appropriate data structure to store application's data is important as it affects the operation and performance of applications
- When you want to solve a problem programmatically, you need to think what are the suitable / appropriate data structures to use



EXAMPLE: SEARCHING OF DATA

Problem:

- Given an array of n integers and an integer key for searching, find the index (location) of the integer key in the array

Data structure:

- Array is used to store n integers

Algorithm:

- Step by step procedure for finding the index of the integer key in the array

SEARCHING OF DATA USING SEQUENTIAL SEARCH

Sequential Search (also called Linear Search)

- Sequentially walk through the array, starting with the first element
- Compares each element with the integer key and stops when
  - The value of the key is the same as the element **OR**
  - Reach the end of the array
- The index of the element at the stopping position is the result.

SEARCHING OF DATA USING SEQUENTIAL SEARCH

Demonstration:
Given the following array and a key 10, find the index of the key in the array. If it is not found, return -1

Index	0	1	2	3	4	5	6	7	8
Value	-1	1	2	5	8	10	21	60	80
	↑	↑	↑	↑	↑	↑			
	-1==10? No :(	1==10? No :(	2==10? No :(	5==10? No :(	8==10? No :(	10==10? Yes :D			

Result: Index is 5

Number of Key Comparisons Required? 6

SEQUENTIAL SEARCH (C++ CODE)

```
/*
   - Searches arr[first] .. array[last] for key
   - Returns: index of the matching element if it finds key,
              otherwise -1
   - Parameters
        arr: array of int values
        first, last: lower and upper subscript bounds
        key: value to search for
*/

int sequentialSearch(int arr[], int first, int last, int key)
{
   for(int index=first; index<=last; index++)
   {
      if(key == arr[index])
         return index; // found! return index
   }
   return -1; // key is not found
}
```

HOW FAST IS A SEQUENTIAL SEARCH?

Best case:

- The first element in the array is the key
- Only one key comparison is required

Worst case:

- The last element in the array is the key or the key is not in the array
- An array with n elements took n key comparisons
- Sequential search algorithm requires n key comparisons where n is the number of elements in an array

SEARCHING OF DATA USING BINARY SEARCH

Any better way? Yes!!!

Binary Search:

- Inspect the middle element of the **SORTED** array
- Check if the element value is greater than or equal to the key that we are looking for
  - If so, we let the left half of the array be the new set to search
  - If not, we let the right half of the array be the new set to search
- Repeat this process until either the element we are looking for is located or the array cannot be further divided

SEARCHING OF DATA USING BINARY SEARCH

Demonstration: Given the following array and a key 10,
find the index of the key in the array. If it is not found, return -1

Index	0	1	2	3	4	5	6	7	8
Value	-1	1	2	5	8	10	21	60	80

Binary search:

- Step 1: Inspect the middle element of the sorted array
  - Middle element with index = 4 (since (0+8)/2) and the value is 8
  - Is 8 >= 10? No
  - Conclusion: Right half

Index	0	1	2	3	4	5	6	7	8
Value	-1	1	2	5	8	10	21	60	80

↑ :(
8 >= 10? No
Right Half

SEARCHING OF DATA USING BINARY SEARCH

Binary search:

- Step 2: Inspect the middle element of the right half of the sorted array
  - Middle element with index = 6 (since (5 + 8)/2) and the value is 21
  - Is 21 >= 10? Yes
  - Conclusion: Left half

Index	0	1	2	3	4	5	6	7	8
Value	-1	1	2	5	8	10	21	60	80

↑:(
21 >= 10? Yes
Left Half

SEARCHING OF DATA USING BINARY SEARCH

Binary search:

- Step 3: Since there is only one element left in the array, inspect that element and see if it is the key
  - Is 10 == 10? Yes
  - Done!

Index	0	1	2	3	4	5	6	7	8
Value	-1	1	2	5	8	10	21	60	80

↑:(
10 == 10? Yes
Done!

Result: Index is 5

Number of comparisons for binary search in this case = 3

BINARY SEARCH (C++ CODE)

```
/*
   - Searches sortedArr[first] .. sortedArr[last] for key
   - Returns: index of the matching element if it finds key,
              otherwise -1
   - Parameters
        sortedArr: array of int values sorted in ascending order
        first, last: lower and upper subscript bounds
        key: value to search for
*/

int binarySearch(int sortedArr[], int first, int last, int key) {
   int mid, position = -1;
   bool found = false;
   while(!found && first <= last) {
      mid = (first + last) / 2; // compute the mid index
      if(sortedArr[mid] == key) {
         found = true;
         position = mid;
      }
      else if(sortedArr[mid] > key)
         last = mid - 1;  // repeat search in left half
      else
         first = mid + 1; // repeat search in right half
   }
   return position;
}
```

PERFORMANCE OF BINARY SEARCH

From the example, we are able to observe that most of the array is not searched at all

This saves a lot of time and makes the binary search efficient

Question:
What is the criteria for binary search to be applicable???

Answer:
The input array MUST BE SORTED!

HOW FAST IS A BINARY SEARCH?

Best case:

- The middle element in the array is the key
- Only one key comparison is required

Worst case:

- The only remaining element for searching in the array is the key or the key is not in the array
- It is fairly easy to observe that an array with 11 elements took 4 comparisons

HOW FAST IS A BINARY SEARCH?

Worst case (Cont'd)

- How about an array with 32 elements?
  - 1st comparison: The array has 16 elements remain for searching
  - 2nd comparison: The array has 8 elements remain for searching
  - 3rd comparison: The array has 4 elements remain for searching
  - 4th comparison: The array has 2 elements remain for searching
  - 5th comparison: The array has 1 element remain for searching

Answer:
5 comparisons

- How about an array with 250 elements?
  - 1st comparison: The array has 125 elements remain for searching
  - 2nd comparison: The array has 63 elements remain for searching
  - 3rd comparison: The array has 32 elements remain for searching
  - 4th comparison: The array has 16 elements remain for searching
  - 5th comparison: The array has 8 elements remain for searching
  - 6th comparison: The array has 4 elements remain for searching
  - 7th comparison: The array has 2 elements remain for searching
  - 8th comparison: The array has 1 element remain for searching

Answer:
8 comparisons

HOW FAST IS A BINARY SEARCH?

Worst case (Cont'd)

- How about an array with 512 elements?
  - 1st comparison: The array has 256 elements remain for searching
  - 2nd comparison: The array has 128 elements remain for searching
  - 3rd comparison: The array has 64 elements remain for searching
  - 4th comparison: The array has 32 elements remain for searching
  - 5th comparison: The array has 16 elements remain for searching
  - 6th comparison: The array has 8 elements remain for searching
  - 7th comparison: The array has 4 elements remain for searching
  - 8th comparison: The array has 2 elements remain for searching
  - 9th comparison: The array has 1 element remain for searching

Answer: 9 comparisons

HOW FAST IS A BINARY SEARCH?

Worst case (Cont'd)

- Observation:
  - An array with 11 elements took 4 comparisons
  - An array with 32 elements took 5 comparisons
  - An array with 250 elements took 8 comparisons
  - An array with 512 elements took 9 comparisons
- What is the pattern?
  - $32 = 2^5$ and $512 = 2^9$
  - $8 < 10 < 16$ ($2^3 < 10 < 2^4$)
  - $128 < 250 < 256$ ($2^7 < 250 < 2^8$)
- What is the conclusion?
  - Binary search algorithm requires $\log_2(n)$ times comparison where n is the number of elements in an array

SEQUENTIAL SEARCH VS. BINARY SEARCH

- Is it suggested that binary search is always more efficient than sequential search?
  - It depends…
- The benefit of binary search over sequential search becomes significant for an array over about 100 elements
- For an array with small number of elements, sequential search may be faster because the speed of the simple increment is more efficient compared with the divisions needed in binary search
- Why is that?

SEQUENTIAL SEARCH VS. BINARY SEARCH

- The performance of binary search comes at a price, since the array must be sorted first
- As sorting is not a fast operation, it may not be worth the effort to sort when there are only a few searches
- Note that binary search is not appropriate for container with no random access for the middle item (e.g. linked list)

QUESTION

When you analyze the performance of algorithm, you never mention the average case. Why?

Answer:

- Algorithm's average case performance is difficult to compute
- The computation is related to probabilities of problem sizes and problems of a given size to determine how it will act on average
- We will not worry about calculating the average-case performance in this course

CHAPTER 1 END