Computer Fundamentals

Priti Srinivas Sajja
Professor
Department of Computer Science
Sardar Patel University

Visit pritisajja.info for details

Unit 4: Data Structure

- **Introduction**
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

- **Name: Dr. Priti Srinivas Sajja**
- **Communication:**
  - **Email : priti@pritisajja.info**
  - **Mobile : +91 9824926020**
  - URL :**http://pritisajja.info**
- ***Academic qualifications* : Ph. D in Computer Science**
- ***Thesis title*: Knowledge-Based Systems for Socio-**
- **Economic Rural Development (2000)**
- ***Subject area of specialization* : Artificial Intelligence**
- ***Publications* : 216 in Books, Book Chapters, Journals and in Proceedings of International and National Conferences**

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Unit 4: Data Structures

- Primitive and composite data types
- Arrays, stacks, queues, linked lists
- Binary trees, B-trees
- Hashing techniques
- Linear Search, Binary Search
- Bubble Sort

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- **Hashing**
- Search & Sort

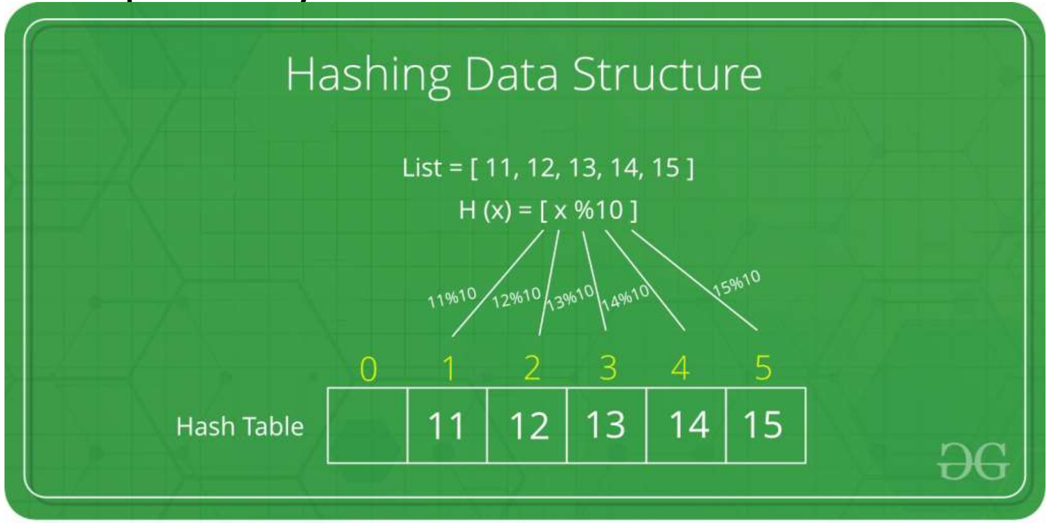
Hashing Data Structure

- Hashing is an important **Data Structure** which is designed to use a special function called the **Hash function** which is used **to map a given value with a particular key** for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used.
- Invented by *Hans Peter Luhn , an IBM Scientis (1953)*

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- **Hashing**
- Search & Sort

Hashing Example

- Let a hash function H(x) maps the value at the index **x%10** in an Array.
- For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.



Hashing Data Structure

List = [11, 12, 13, 14, 15]

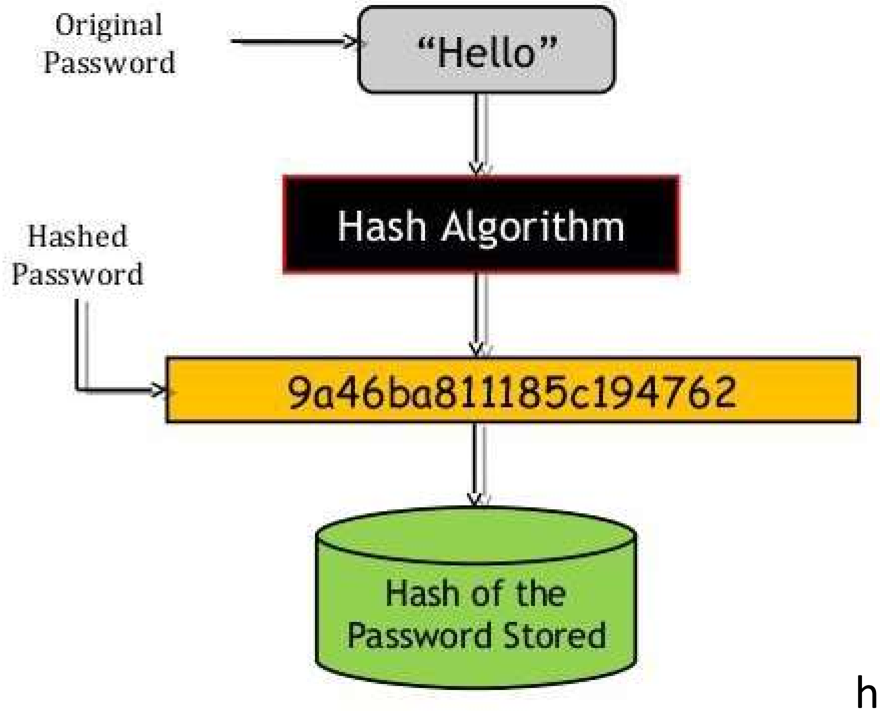
H (x) = [x %10]

0	1	2	3	4	5
	11	12	13	14	15

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- **Hashing**
- Search & Sort

Application: Hashing Password



How password is stored using hash

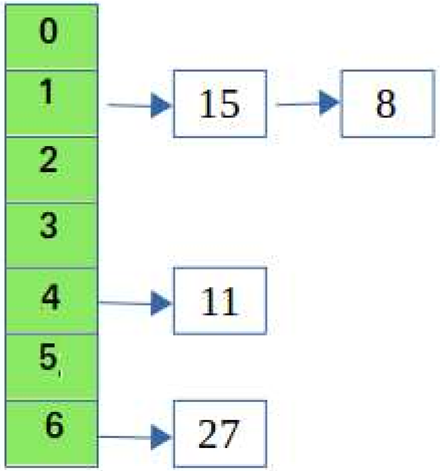
https://medium.com/

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- **Hashing**
- Search & Sort

Let's say hash table with 7 buckets (0, 1, 2, 3, 4, 5, 6)

Keys arrive in the Order (15, 11 , 27 , 8)

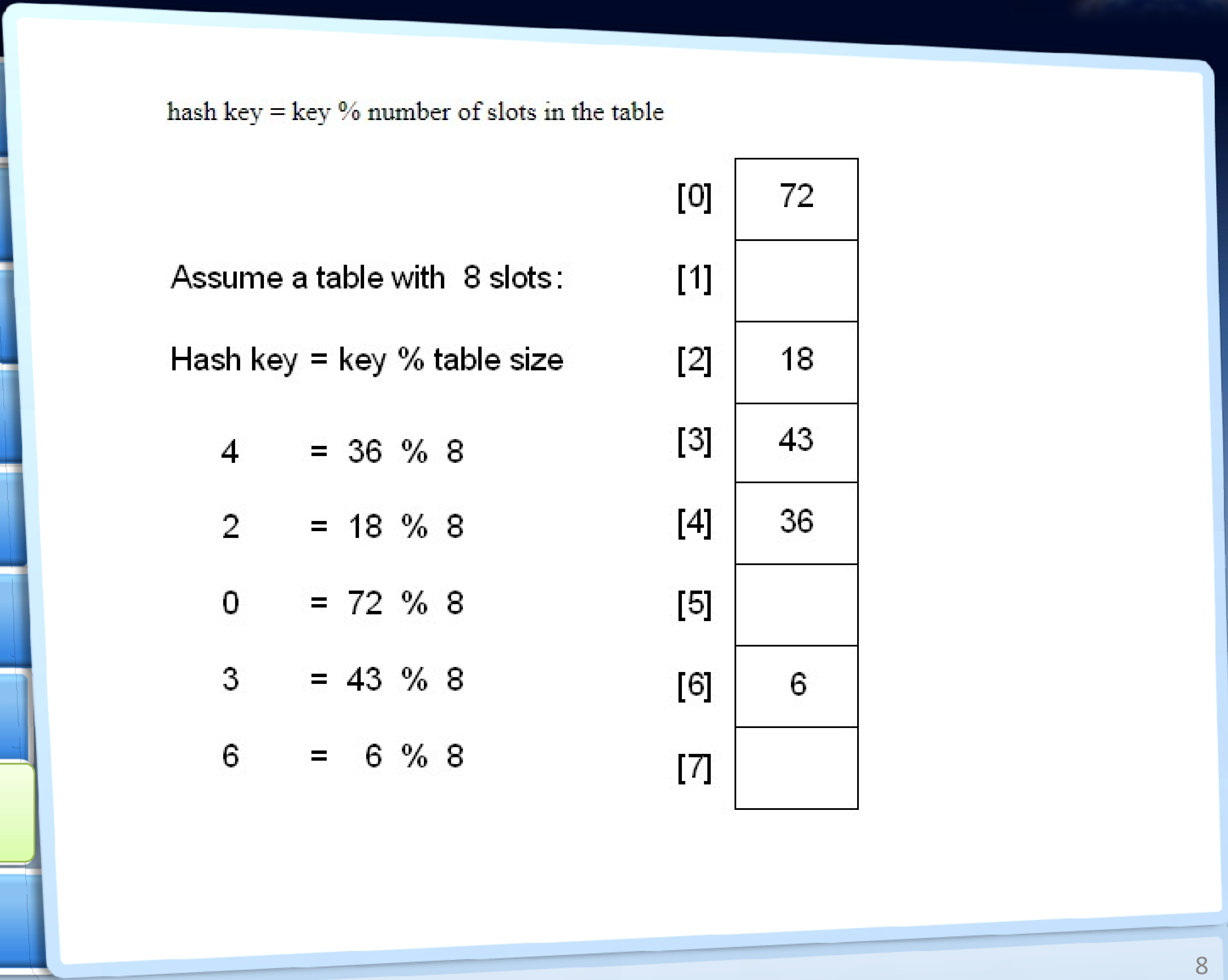


hashIndex = key % noOfBuckets

*In computing, the **modulo operation** returns the remainder or signed remainder of a division, after one number is divided by another (called the **modulus** of the **operation**).*

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- **Hashing**
- Search & Sort



hash key = key % number of slots in the table

Assume a table with 8 slots:

Hash key = key % table size

4 = 36 % 8

2 = 18 % 8

0 = 72 % 8

3 = 43 % 8

6 = 6 % 8

[0]	72
[1]	
[2]	18
[3]	43
[4]	36
[5]	
[6]	6
[7]	

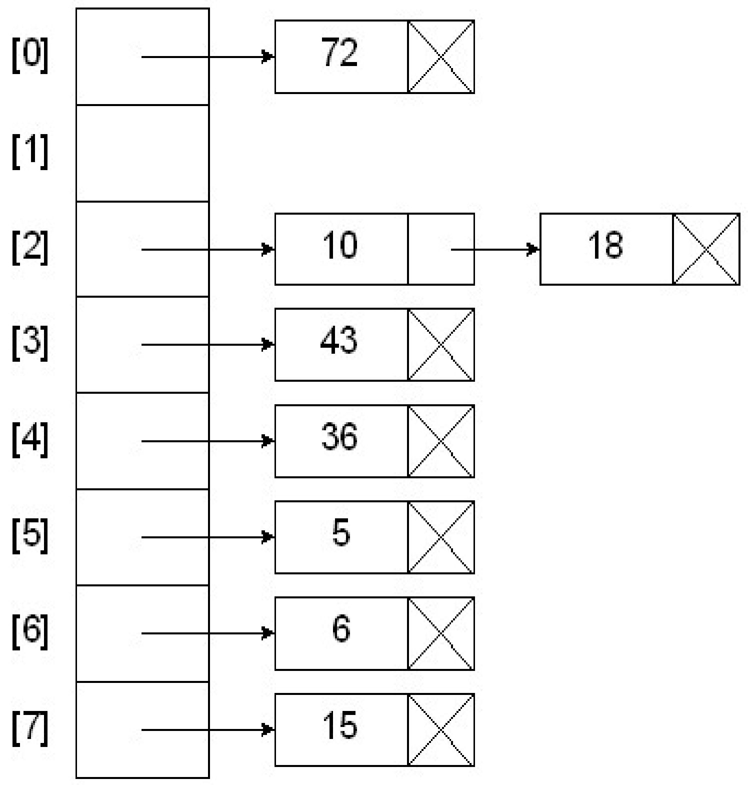
Unit 4: Data Structure

Introduction
Types of DT
Array
Stack
Queue
Linked Lists
Tree and Graph
Hashing
Search & Sort

- Hashing with chains

Hash key = key % table size

4 = 36 % 8
2 = 18 % 8
0 = 72 % 8
3 = 43 % 8
6 = 6 % 8
2 = 10 % 8
5 = 5 % 8
7 = 15 % 8



Unit 4: Data Structure

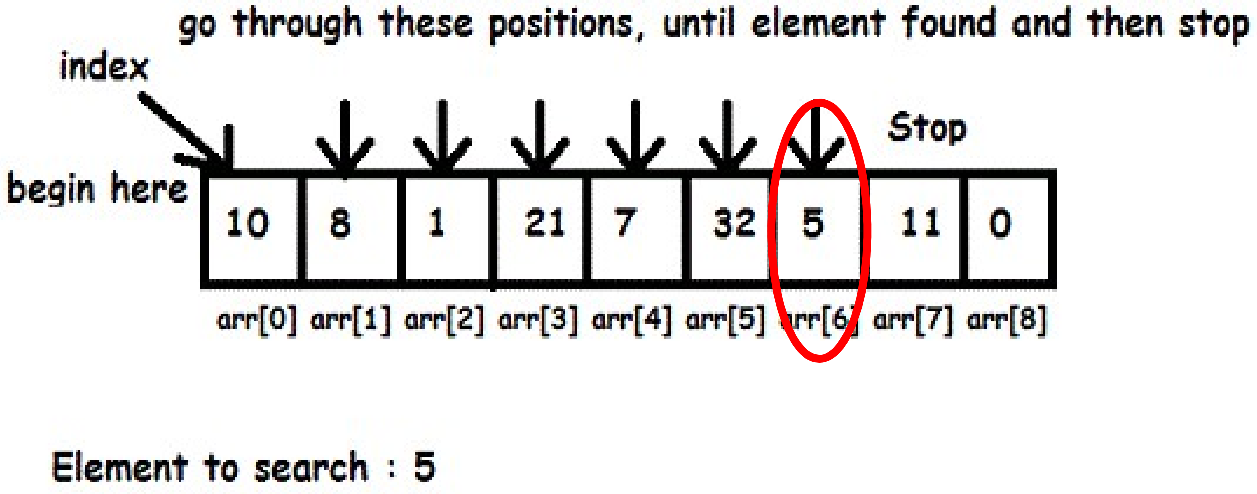
Introduction
Types of DT
Array
Stack
Queue
Linked Lists
Tree and Graph
Hashing
Search & Sort

Linear Search

- The **most basic** type of searching algorithm.
- **Sequentially moves** through the data looking for a matching value.
- It **begins with the first** element, checking it every data until you find what you’re looking for.
- In complexity terms this is an **O(n)** search.
- The time taken to search the list is in parallel with the size of the list.

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- **Search & Sort**



Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Binary Search

- Binary search is an efficient algorithm for finding an item from a **<u>sorted</u>** list of items.
- It works by **repeatedly dividing in half** the portion of the list that could contain the item, until you've narrowed down the possible locations to just one.

Unit 4: Data Structure

Introduction

Types of DT

Array

Stack

Queue

Linked Lists

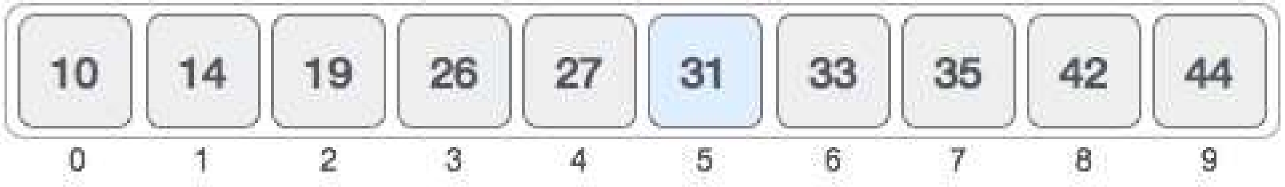
Tree and Graph

Hashing

Search & Sort

Binary Search

- The following is our sorted array and let us assume that we need to search the location of value 31 using binary search.

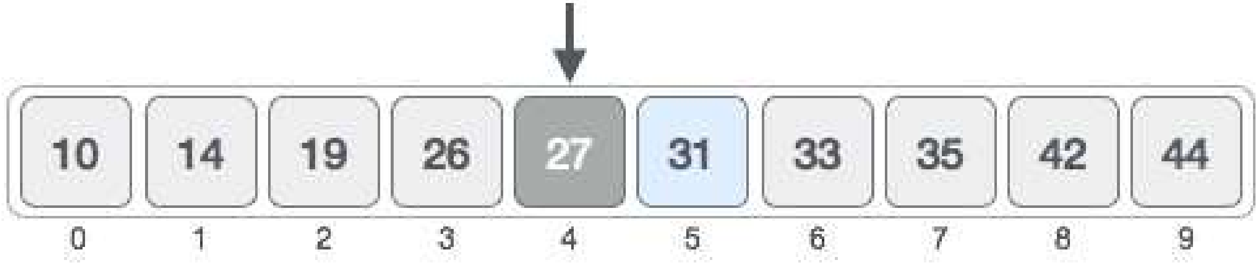


10	14	19	26	27	31	33	35	42	44
0	1	2	3	4	5	6	7	8	9

- First, we shall determine half of the array by using this formula –
- mid = low + (high - low) / 2
- Here it is, 0 + (9 - 0) / 2 = 4 (integer value of 4.5).
- So, 4 is the mid of the array.

Binary Search

- Now we compare the value stored at location 4, with the value being searched, i.e. 31.



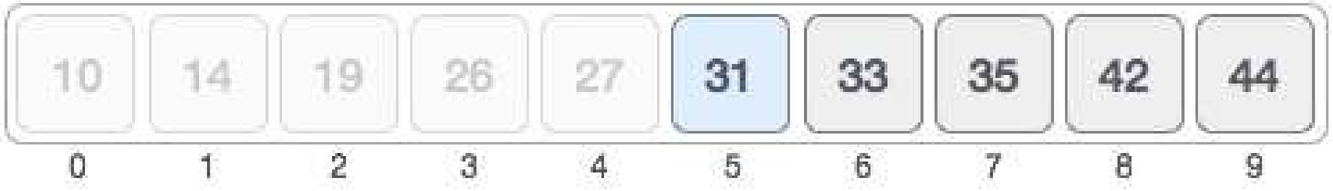
10	14	19	26	27	31	33	35	42	44
0	1	2	3	4	5	6	7	8	9

- The value at location 4 is 27, which is not a match
- As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.

Introduction
Types of DT
Array
Stack
Queue
Linked Lists
Tree and Graph
Hashing
Search & Sort

Binary Search

- The current situation is like this.



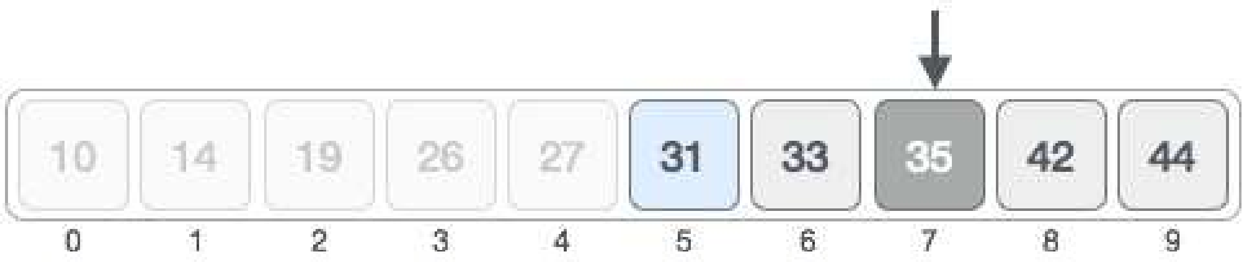
10	14	19	26	27	31	33	35	42	44
0	1	2	3	4	5	6	7	8	9

- We change our low to mid + 1 and find the new mid value again.
- low = mid + 1
- New mid = low + (high - low) / 2
- Our new mid is 7.
- Compare the value stored at location 7 with our target value 31.

Introduction
Types of DT
Array
Stack
Queue
Linked Lists
Tree and Graph
Hashing
Search & Sort

Binary Search

- The current situation is like this.



10	14	19	26	27	31	33	35	42	44
0	1	2	3	4	5	6	7	8	9

- The value stored at location 7 is not a match, rather it is more than what we are looking for.
- So, the value must be in the lower part from this location.

Introduction

Types of DT

Array

Stack

Queue

Linked Lists

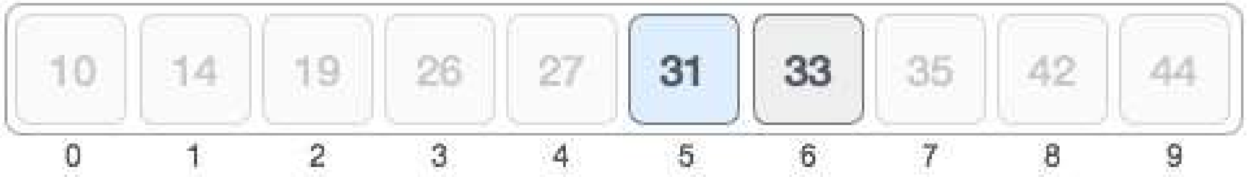
Tree and Graph

Hashing

Search & Sort

Binary Search

- The current situation is like this.



10	14	19	26	27	31	33	35	42	44
0	1	2	3	4	5	6	7	8	9

- Hence, we calculate the mid again. This time it is 5.
- compare the value stored at location 5 with our target value. We find that it is a match.

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- **Search & Sort**

Binary Search

```
Procedure binary_search
A ← sorted array ,
n ← size of array
x ← value to be searched
Set Low= 1 ,
Set High = n
while x not found
    if High< Low EXIT: x does not exists.
    set mid= Low + ( High - Low ) / 2
    if A[mid] < x set Low= Mid + 1
    if A[mid] > x set High = mid - 1
    if A[mid] = x  then exit
end while
end procedure
```

Bubble Sort

- It is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.
- **First Pass:**
(**5 1** 4 2 8) –> (**1 5** 4 2 8)
- Here, algorithm compares the first two elements, and swaps since 5 > 1.
- (1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4
- (1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2
- (1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Unit 4: Data Structure

Introduction
Types of DT
Array
Stack
Queue
Linked Lists
Tree and Graph
Hashing
Search & Sort

Second Pass:

- (**1 4** 2 5 8) –> (**1 4** 2 5 8)
- (1 **4 2** 5 8) –> (1 **2 4** 5 8), Swap since 4 > 2
- (1 2 **4 5** 8) –> (1 2 **4 5** 8)
- (1 2 4 **5 8**) –> (1 2 4 **5 8**)
- Now, the array is already sorted, but our algorithm does not know if it is completed.
- The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Unit 4: Data Structure

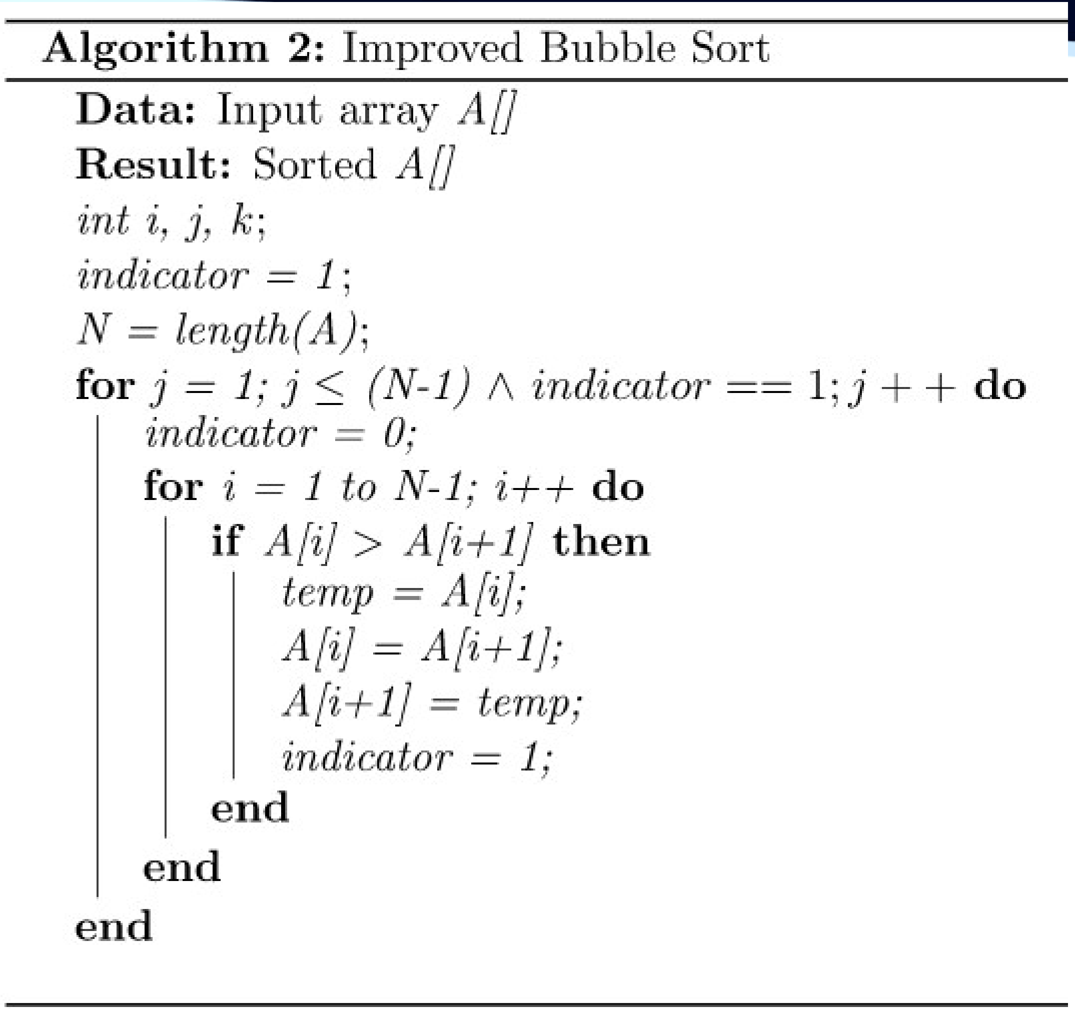
- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Third Pass:

- (**1 2** 4 5 8) –> (**1 2** 4 5 8)
- (1 **2 4** 5 8) –> (1 **2 4** 5 8)
- (1 2 **4 5** 8) –> (1 2 **4 5** 8)
- (1 2 4 **5 8**) –> (1 2 4 **5 8**)

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- **Search & Sort**



```
Algorithm 2: Improved Bubble Sort
  Data: Input array A[]
  Result: Sorted A[]
  int i, j, k;
  indicator = 1;
  N = length(A);
  for $j = 1; j \leq (N\text{-}1) \wedge indicator == 1; j++$ do
      indicator = 0;
      for i = 1 to N-1; i++ do
          if A[i] > A[i+1] then
              temp = A[i];
              A[i] = A[i+1];
              A[i+1] = temp;
              indicator = 1;
          end
      end
  end
```

https://www.baeldung.com

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Main References

- Tremblay J. & Sorenson P. G. : An Introduction to Data Structures with Applications, 2nd Edition, McGraw-Hill International Edition, 1987.

Other References

- Code project.com,
- Tutorialspoint.com,
- Geeksforgeeks.com ,
- Amazing anmimatins.com
- Clipartoff.com
- etc.