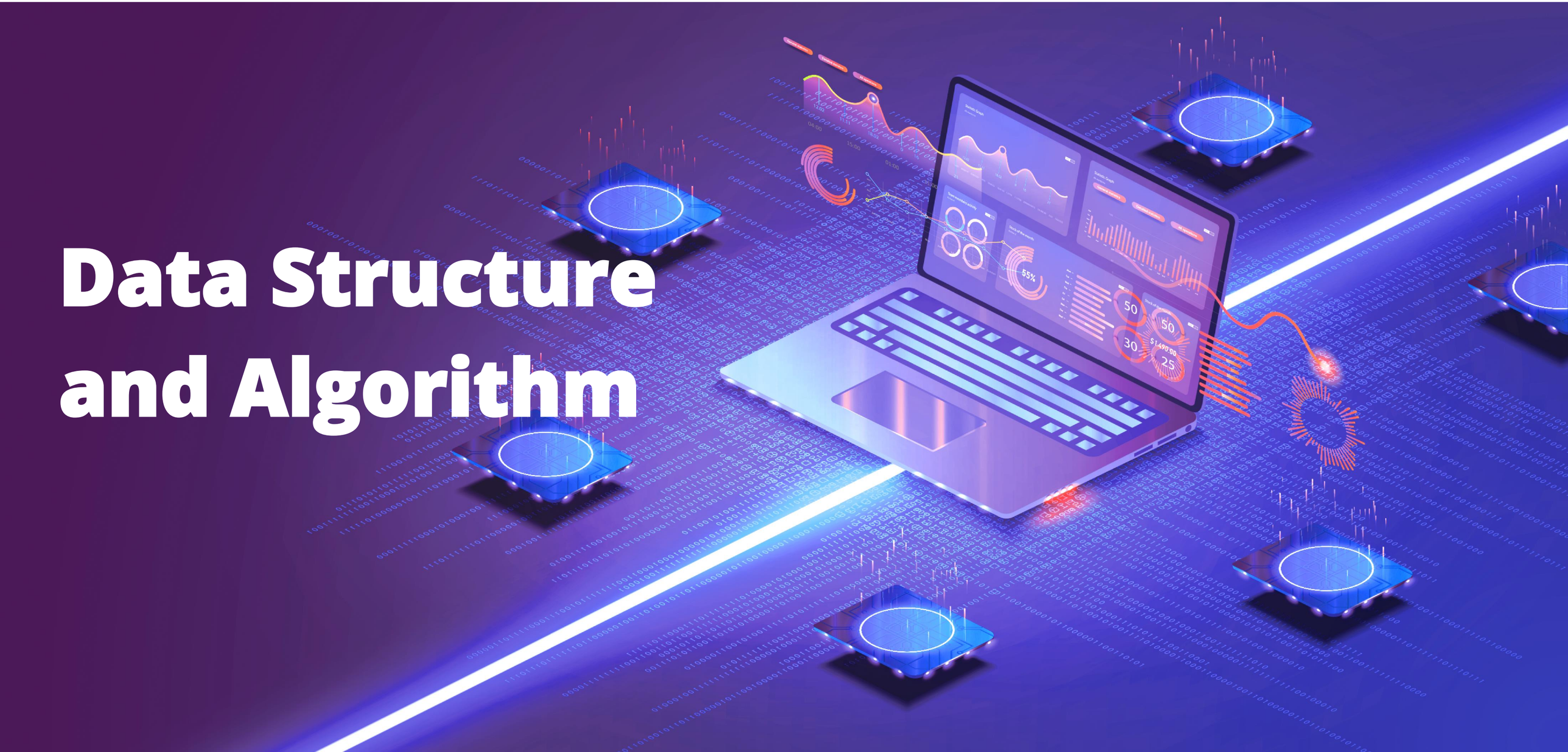
WILEY

ENABLING DISCOVERY | POWERING EDUCATION | SHAPING WORKFORCES

Data Structure and Algorithm



Objectives

By the end of this module, you will be able to:

- Search data by using linear search technique
- Search data by using binary search technique
- Store and search data by using hashing

Scenario

You are in California and suddenly you remember that your childhood friend **John Smith** lives here. But you don’t have his phone number. You pick up the telephone directory and start searching. Directories list the last name first. So you start searching **Smith** first. Search until the letter S starts showing up.

The above is an example of a sequential search. You started at the beginning of a sequence and went through each item one by one, in the order they existed in the list, until you found the item you were looking for.

Linear Search

Linear search:

- Is the simplest searching method.
- Is also referred to as sequential search.
- Involves comparing the items sequentially with the elements in the list.

Implementing Linear Search

- The linear search would begin by comparing the required element with the first element in the list.
- If the values do not match:
  - The required element is compared with the second element in the list
- If the values still do not match:
  - The required element is compared with the third element in the list
- This process continues, until:
  - The required element is found or the end of the list is reached

Implementing Linear Search (cont.)

The following algorithm depicts the logic to search an employee ID in an array by using linear search:

```
1. Read the employee ID to be searched.
2.  Set i = 0.
3.  Repeat step 4 until i = n or arr[i] = employee ID.
4. Increment i by 1.
5.  If i = n:
      Display “Not Found”.
   Else:
      Display “Found”.
```

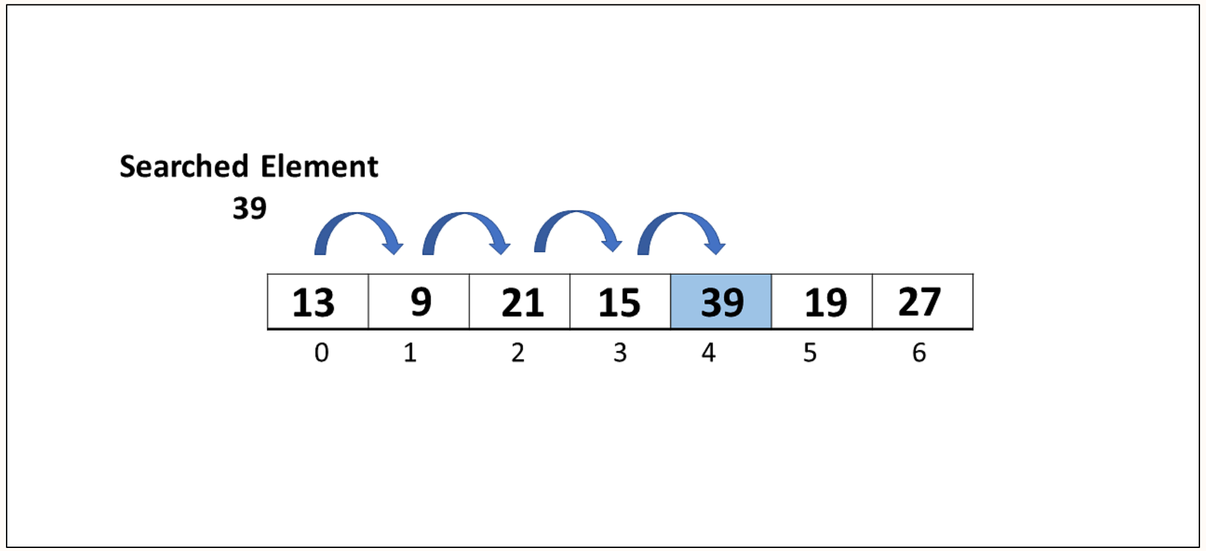
Implementing Linear Search (cont.)

- Input the number to be searched and store in variable n.
- Array **a** is initialized.
- Using for loop, perform the linear search.
- Check if n==a[i], if true print “Number found”.
- Also, return its index or position.
- Iterate till we found the desired number which is asked by the user.

```
n=int(input("Enter the
number to be searched (1-
10):"))

a=[1, 2, 4, 3,5,7,9,8,6,10 ]

for i in range(1,(len(a))):

    if n==a[i]:

        print("number found
at",i+1)
```

Implementing Linear Search (cont.)

- The efficiency of a searching algorithm is determined by the running time of the algorithm.
- The best case efficiency of linear search is therefore, O(1). The worst case efficiency of linear search is therefore, O(n).



Question

While implementing linear search, if you find that the element is not present in an array of size n, how many number of comparisons would you have made to search the required element in the given list?

- 1
- n-1
- n

Scenario

Consider an example where you have to search the name, **Steve**, in a telephone directory that is sorted alphabetically.

To search the name, Steve, by using binary search algorithm:

- You open the telephone directory at the middle to determine which half contains the name.
- Open that half at the middle to determine which quarter of the directory contains the name.

Performing Binary Search

Binary Search Algorithm:

- Is used for searching large lists.
- Searches the element in very few comparisons.
- Can be used only if the list to be searched is sorted.

Implementing Binary Search

- Consider an example where you have to search the name, Steve, in a telephone directory that is sorted alphabetically.

- To search the name, Steve, by using binary search algorithm:
  - You open the telephone directory at the middle to determine which half contains the name.
  - Open that half at the middle to determine which quarter of the directory contains the name.

- Repeat this process until the name, Steve, is not found.
- Binary search reduces the number of pages to be searched by half each time.

Implementing Binary Search (cont.)

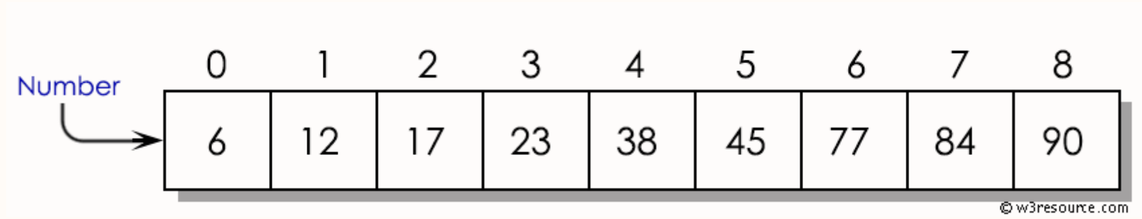
The following algorithm depicts the logic to search a desired element by using binary search:

```
1.  Accept the element to be searched.
2.  Set lowerbound = 0.
3.  Set upperbound = n - 1.
4.  Set mid = (lowerbound + upperbound)/2.
5.  If arr[mid] = desired element:
   a. Display “Found”.
   b. Go to step 10.
6.  If desired element < arr[mid]:
   Set upperbound = mid - 1.
```

Implementing Binary Search (cont.)

```
7.  If desired element > arr[mid]:
  Set lowerbound = mid + 1.
8.  If lowerbound <= upperbound:
  Go to step 4.
9.  Display “Not Found”.
10. Exit.
```

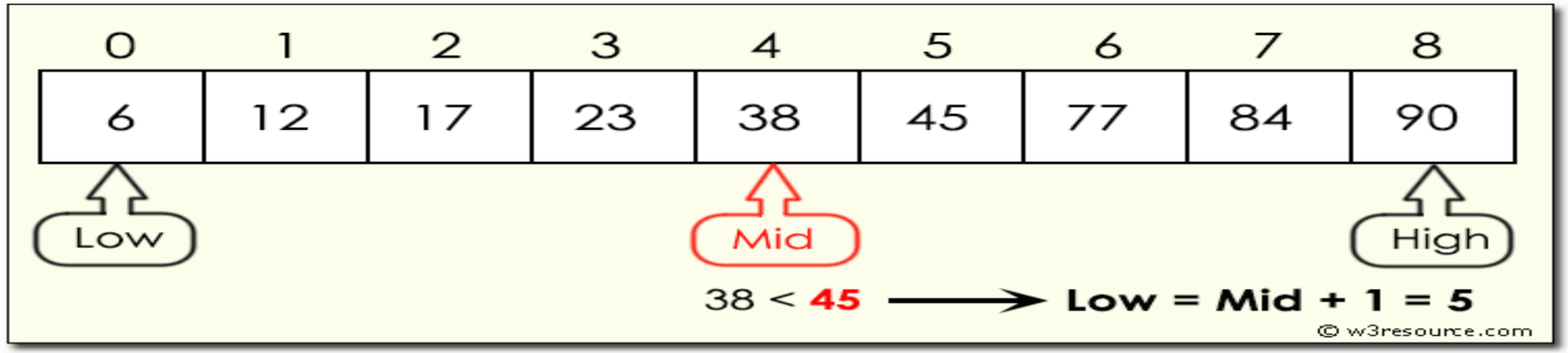
Implementing Binary Search



	Low	High	Mid
#1	0	8	4

Search (45)

$$mid = \left[\frac{low + high}{2} \right]$$

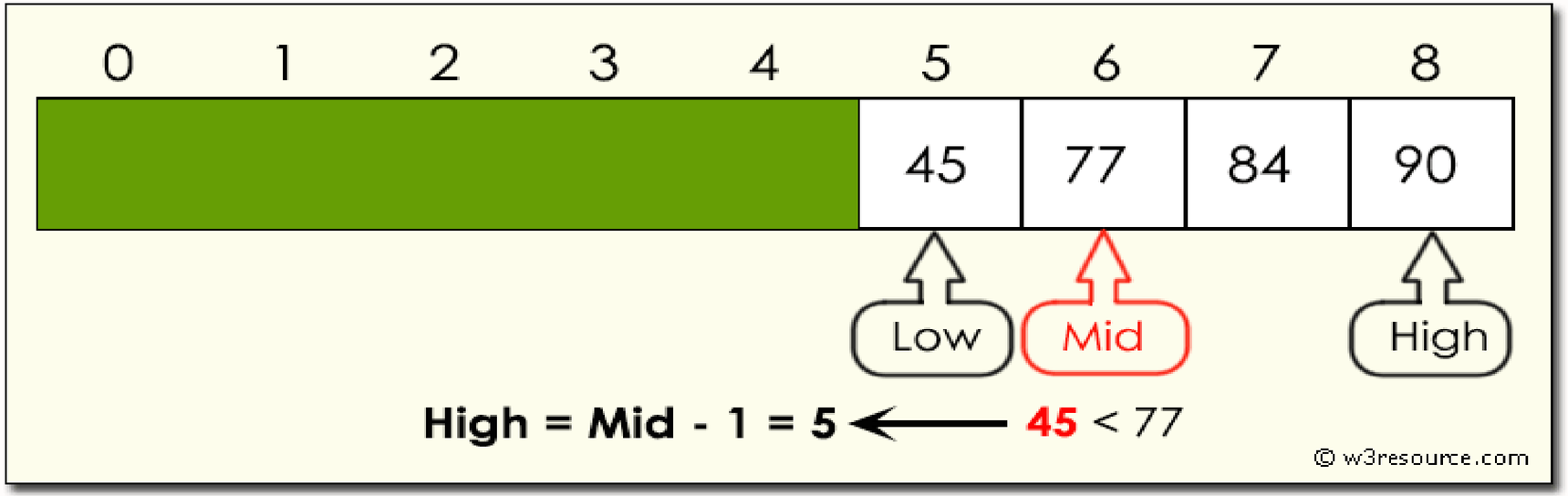


Implementing Binary Search

	Low	High	Mid
#1	0	8	4
#2	5	8	6

Search (45)

$$mid = \left[\frac{low + high}{2} \right]$$

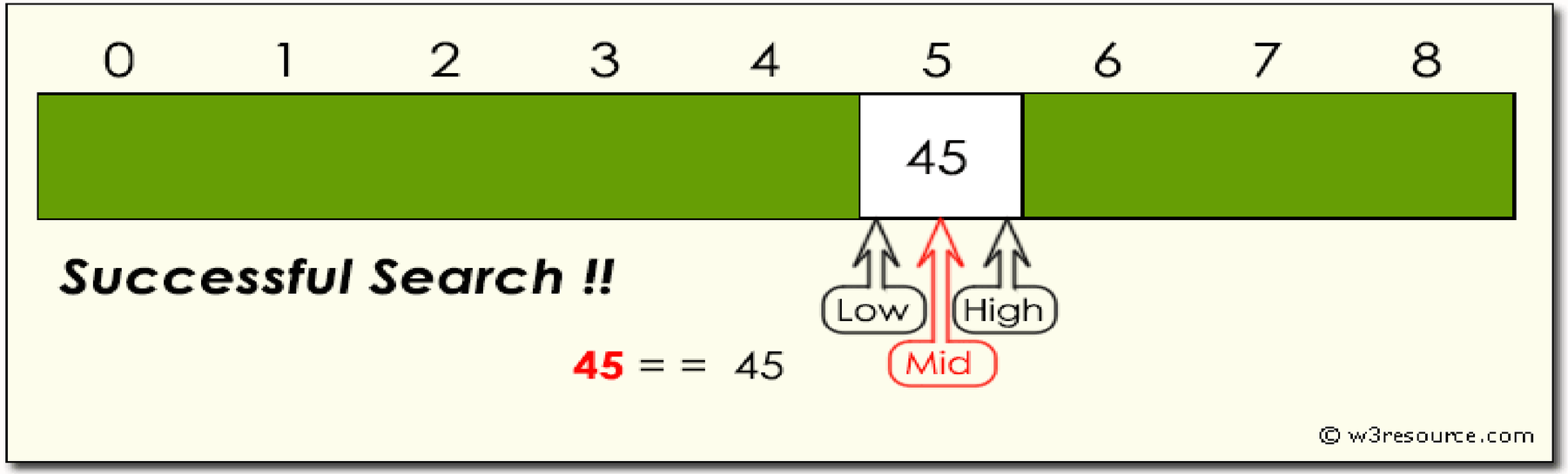


Implementing Binary Search

	Low	High	Mid
#1	0	8	4
#2	5	8	6
#3	5	5	5

Search (45)

$$mid = \left[\frac{low + high}{2} \right]$$



Implementing Linear Search

- Input the number to be searched from the user and store in variable n.
- Array **a** is initialized.
- Using for loop, perform the linear search.
- Check if n==a[i], if true print “Number found”.
- Also, return its index or position.
- Iterate till we found the desired number which is asked by the user.

```
def binary_search(item_list,item): first = 0
last = len(item_list)-1 found = False while(
first<=last and not found): mid = (first +
last)//2 if item_list[mid] == item : found =
True else: if item < item_list[mid]: last =
mid - 1 else: first = mid + 1 return found
print(binary_search([1,2,3,5,8], 6))
print(binary_search([1,2,3,5,8], 5))
```

Activity: Performing Binary Search

Write a program to search a number in an array that contains a maximum of 20 elements by using binary search.

Assume that the array elements are entered in the ascending order and all the array elements are unique.

The program should also display the total number of comparisons made.

Question

To implement _____ search algorithm, the list should be sorted.

Scenario

When you have to search for the element corresponding to a given key value in a given list of elements, you would search sequentially through the list until the element with the desired key value is found.

This method is very time-consuming, especially if the list is very large.

It would be easier to find the element if it had an address.

Hashing

Binary search algorithm has the following disadvantages:

- It works only on sorted lists.
- It requires a way to directly access the middle element of the list.

An alternate searching algorithm that **overcomes these limitations** and provides good efficiency is **hashing**.

Defining Hashing

- The fundamental principle of hashing is to **convert a given key value to an offset address** to retrieve an element.

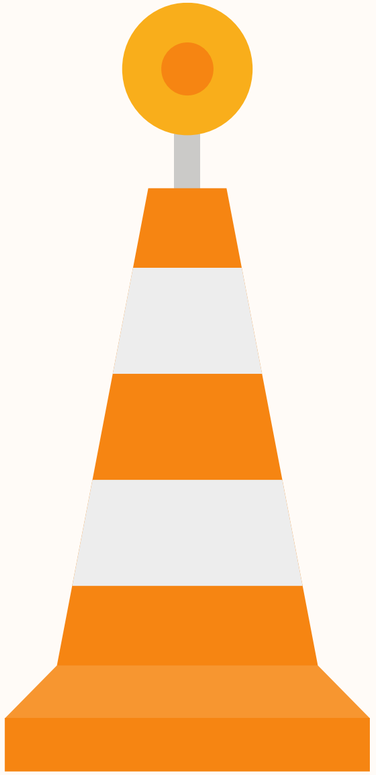
- **Conversion of a key to an address is done by a relation** (formula), which is known as a **hashing function**.

- The process of searching an element by using hashing can be summarized as:
  - Given a key, the hash function converts it into a hash value (location) within the range 1 to n, where n is the size of the storage (address) space that has been allocated for the records.
  - The element is then retrieved at the location generated.

Limitations of Hashing

Two limitations of hashing are:

- It may result in a collision.
- It does not allow sequential access.



Collision and How to Resolve it

- A situation in which an **attempt is made to store two keys at the same position** is known as collision.
- Two records cannot occupy the same position.
- Some methods of resolving collision are **chaining** and **open addressing**.
- The chaining method uses:
  - **Links/pointers to resolve** hash clashes.
  - Coalesced chaining and separate chaining techniques to resolve a collision.
- In the open addressing method:
  - Elements that produce a collision **are stored at an alternate position** in the hash table.
  - An alternate location is obtained by searching the hash table. This process is called **probing**.

Probing

The following probing sequence can be used to search an empty position in the hash table:

- Linear probing
- Quadratic probing
- Double hashing

Determining the Efficiency of Hashing

- Searching becomes faster using hashing as compared to any other searching method.
- The efficiency of hashing is ideally O(1).
- However, because of any collision, the efficiency of hashing gets reduced.
- The efficiency of hashing, in this case, depends on the quality of the hash function.

Question

What is direct addressing?

Question

Which of the following is not a technique to avoid a collision?

A. Make the hash function appear random
B. Use the chaining method
C. Use uniform hashing
D. Increasing hash table size


Questions?

Summary

In this session, you learned:

- The best case of efficiency of linear search is O(1), and the worst case of efficiency of linear search is O(n).
- To apply binary search algorithm, you should ensure that the list to be searched is sorted.
- The best case of efficiency of binary search is O(1), and the worst case of efficiency of binary search is O(log n).
- The fundamental principle of hashing is to convert a given key value to an offset address to retrieve a record.
- In hashing, conversion of a key to an address is done by a relation (formula), which is known as a hashing function.
- The situation in which the hash function generates the same hash value for two or more keys is called collision.
- Some methods of resolving collision are chaining and open addressing.