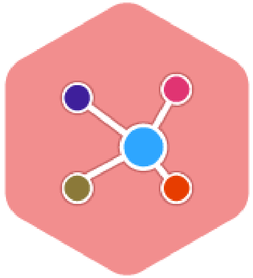
AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

CHAPTER 5
SORTING & SEARCHING



SUBJECT:DATA STRUCTURE CODE:3130702	PREPARED BY: ASST.PROF.PARAS NARKHEDE (CSE DEPARTMENT,ACET)	AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

SORTING

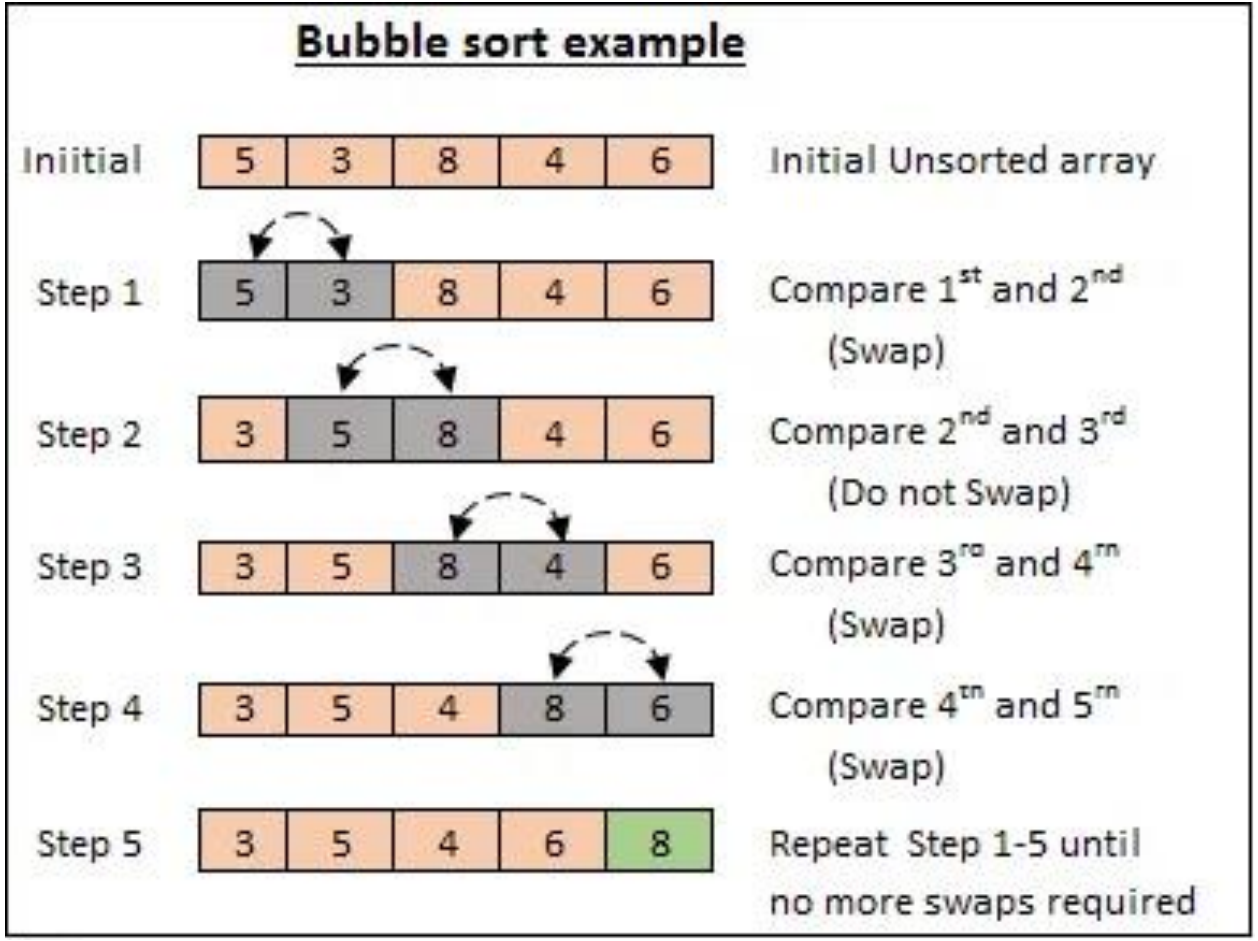
AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

SORTING

- Sorting refers to the operation or technique of arranging and rearranging sets of data in some specific order. A collection of records called a list where every record has one or more fields. The fields which contain a unique value for each record is termed as the *key* field. For example, a phone number directory can be thought of as a list where each record has three fields - 'name' of the person, 'address' of that person, and their 'phone numbers'. Being unique phone number can work as a key to locate any record in the list.
- Sorting is the operation performed to arrange the records of a table or list in some order according to some specific ordering criterion. Sorting is performed according to some key value of each record.
- The records are either sorted either numerically or alphanumerically. The records are then arranged in ascending or descending order depending on the numerical value of the key. Here is an example, where the sorting of a lists of marks obtained by a student in any particular subject of a class.

BUBBLE SORT

- ➔ Bubble Sort Algorithm is used to arrange N elements in ascending order, and for that, you have to begin with 0th element and compare it with the first element. If the 0th element is found greater than the 1st element, then the swapping operation will be performed, i.e., the two values will get interchanged. In this way, all the elements of the array get compared.


Bubble sort example
Iniitial
5 3 8 4 6
Initial Unsorted array
Step 1
5 3 8 4 6
Compare 1st and 2nd (Swap)
Step 2
3 5 8 4 6
Compare 2nd and 3rd (Do not Swap)
Step 3
3 5 8 4 6
Compare 3rd and 4th (Swap)
Step 4
3 5 4 8 6
Compare 4th and 5th (Swap)
Step 5
3 5 4 6 8
Repeat Step 1-5 until no more swaps required

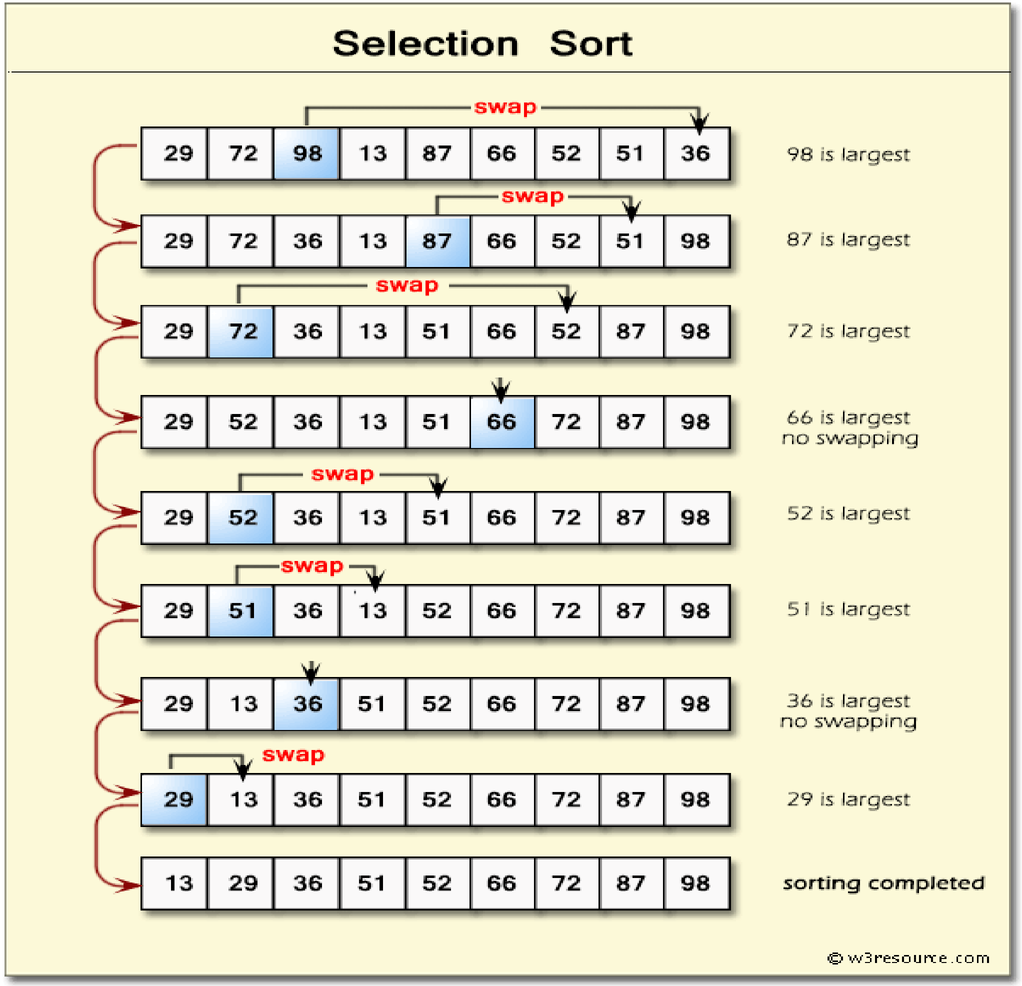
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

ALGORITHM FOR BUBBLE SORT

```
➔ algorithm Bubble_Sort(list)
➔ Pre: list != fi
➔ Post: list is sorted in ascending order for all values
➔ for i <- 0 to list:Count - 1
➔ for j <- 0 to list:Count - 1
➔ if list[i] < list[j]
➔ Swap(list[i]; list[j])
➔ end if
➔ end for
➔ end for
➔ return list
➔ end Bubble_Sort
```

SELECTION SORT

- The selection is a straightforward process of sorting values. In this method, to sort the data in ascending order, the 0th element is compared with all other elements. If the 0th element is found to be greater than the compared element, the two values get interchanged. In this way after the first iteration, the smallest element is placed at 0th position. The technique is repeated until the full array gets sorted.



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

ALGORITHM FOR SELECTION SORT

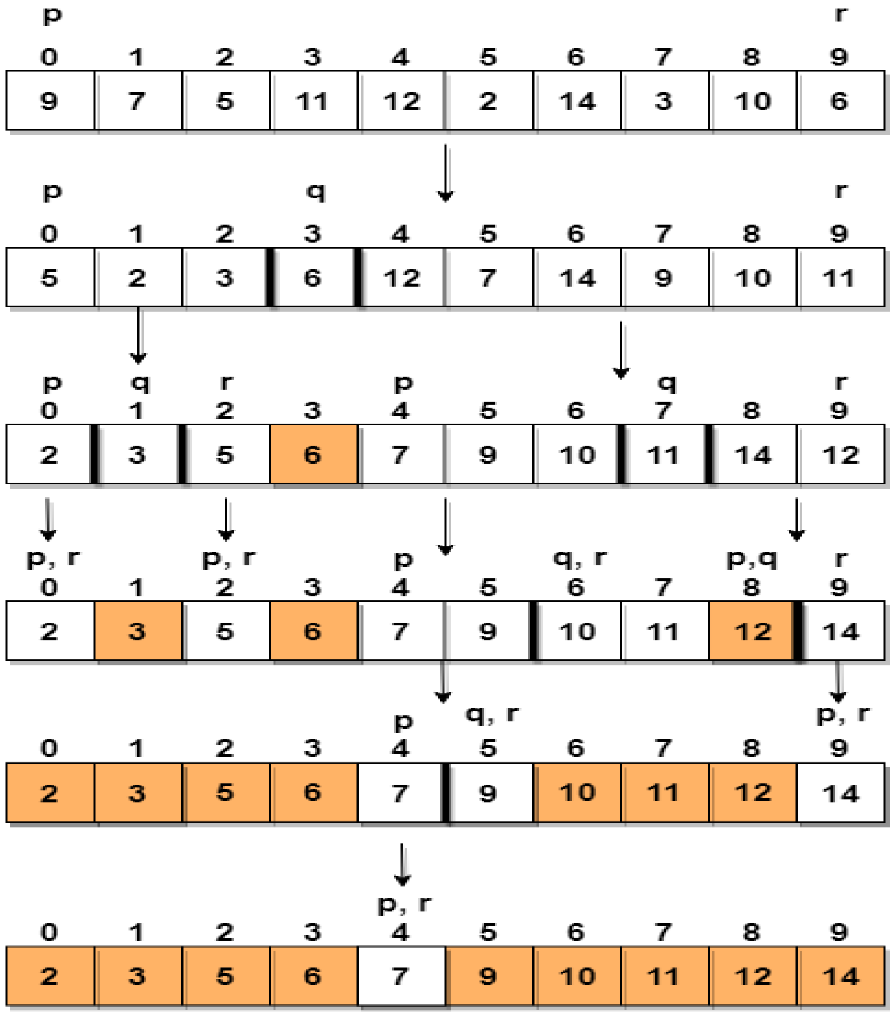
- ➔ Set MIN to location 0
- ➔ Search the minimum element in the list
- ➔ Swap with value at location MIN
- ➔ Increment MIN to point to next element
- ➔ Repeat until list is sorted

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

QUICK SORT

- Quick sort is one of the most famous sorting algorithms based on divide and conquers strategy which results in an O(n log n) complexity. So, the algorithm starts by picking a single item which is called pivot and moving all smaller items before it, while all greater elements in the later portion of the list. This is the main quick sort operation named as a partition, recursively repeated on lesser and greater sublists until their size is one or zero - in which case the list is wholly sorted. Choosing an appropriate pivot, as an example, the central element is essential for avoiding the severely reduced performance of O(n2).

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

ALGORITHM FOR QUICK SORT

➔ algorithm Quick_Sort(list)
➔ Pre: list 6= fi
➔ Post: the list has been sorted in ascending order
➔ if list.Count = 1 // list already sorted
➔ return list
➔ end if
➔ pivot <- Median_Value(list)
➔ for i <- 0 to list.Count - 1
➔ if list[i] = pivot
➔ equal.Insert(list[i])

ALGORITHM FOR QUICK SORT

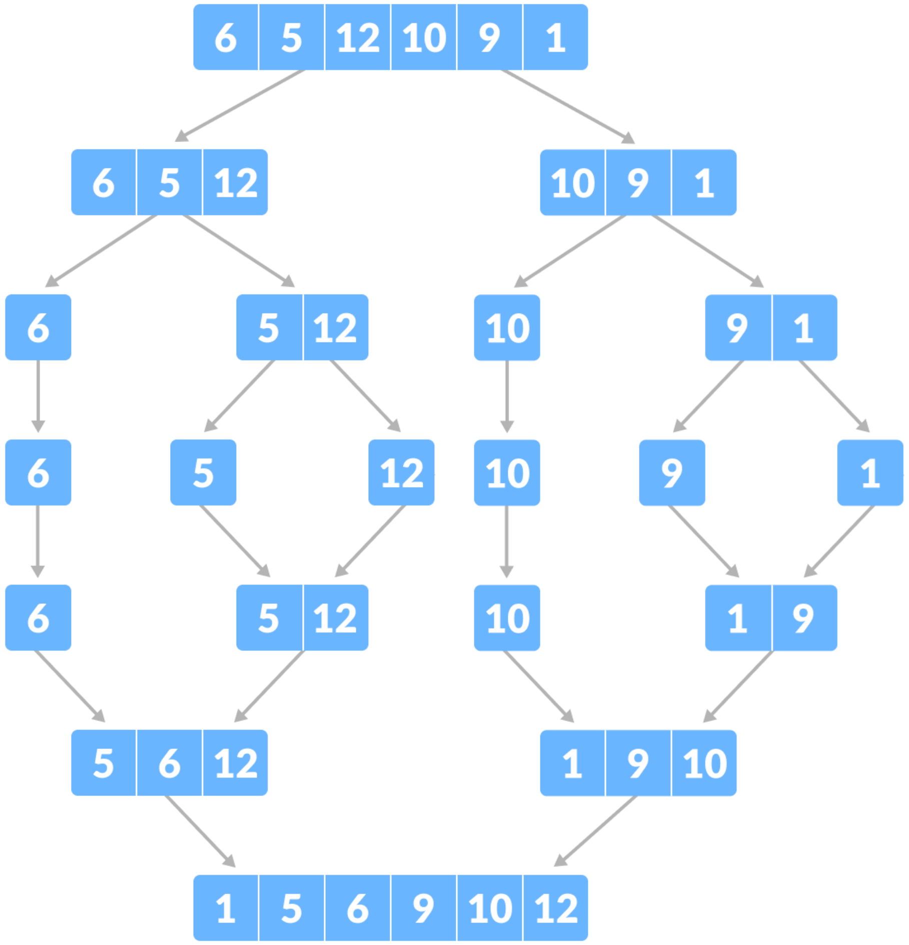
```
➔ end if
➔ if list[i] < pivot
➔ less.Insert(list[i])
➔ end if
➔ if list[i] > pivot
➔ greater.Insert(list[i])
➔ end if
➔ end for
➔ return Concatenate(Quick_Sort(less), equal, Quick_Sort(greater))
➔ end Quick_sort
```

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

MERGE SORT

- Merge sort is another sorting technique and has an algorithm that has a reasonably proficient space-time complexity - O(n log n) and is quite trivial to apply. This algorithm is based on splitting a list, into two comparable sized lists, i.e., left and right and then sorting each list and then merging the two sorted lists back together as one.
- Merge sort can be done in two types both having similar logic and way of implementation. These are:
  - Top down implementation
  - Bottom up implementation

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

ALGORITHM FOR MERGE SORT

➔ algorithm Merge_Sort(list)
➔ Pre: list 6= fi
➔ Post: list has been sorted into values of ascending order
➔ if list.Count = 1 // when already sorted
➔ return list
➔ end if
➔ m <- list.Count = 2
➔ left <- list(m)
➔ right <- list(list.Count - m)
➔ for i <- 0 to left.Count - 1

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

ALGORITHM FOR MERGE SORT

```
➔ left[i] <- list[i]
➔ end for
➔ for i <- 0 to right.Count -1
➔ right[i] <- list[i]
➔ end for
➔ left <- Merg_Sort(left)
➔  right <- Merge_Sort(right)
➔  return MergeOrdered(left, right)
➔  end Merge_Sort
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

SEARCHING

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

SEARCHING

- Searching is an operation or a technique that helps finds the place of a given element or value in the list. Any search is said to be successful or unsuccessful depending upon whether the element that is being searched is found or not. Some of the standard searching technique that is being followed in the data structure is listed below:
  - Linear Search or Sequential Search
  - Binary Search

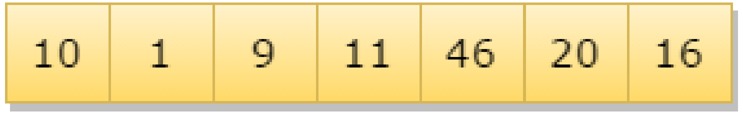
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

SEQUENTIAL SEARCH

- This is the simplest method for searching. In this technique of searching, the element to be found in searching the elements to be found is searched sequentially in the list. This method can be performed on a sorted or an unsorted list (usually arrays). In case of a sorted list searching starts from 0th element and continues until the element is found from the list or the element whose value is greater than (assuming the list is sorted in ascending order), the value being searched is reached.
- As against this, searching in case of unsorted list also begins from the 0th element and continues until the element or the end of the list is reached

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

EXAMPLE



One-Dimensional Array having 7 Elements

The list given below is the list of elements in an unsorted array. The array contains ten elements. Suppose the element to be searched is '46', so 46 is compared with all the elements starting from the 0th element, and the searching process ends where 46 is found, or the list ends.

The performance of the linear search can be measured by counting the comparisons done to find out an element. The number of comparison is 0(n).

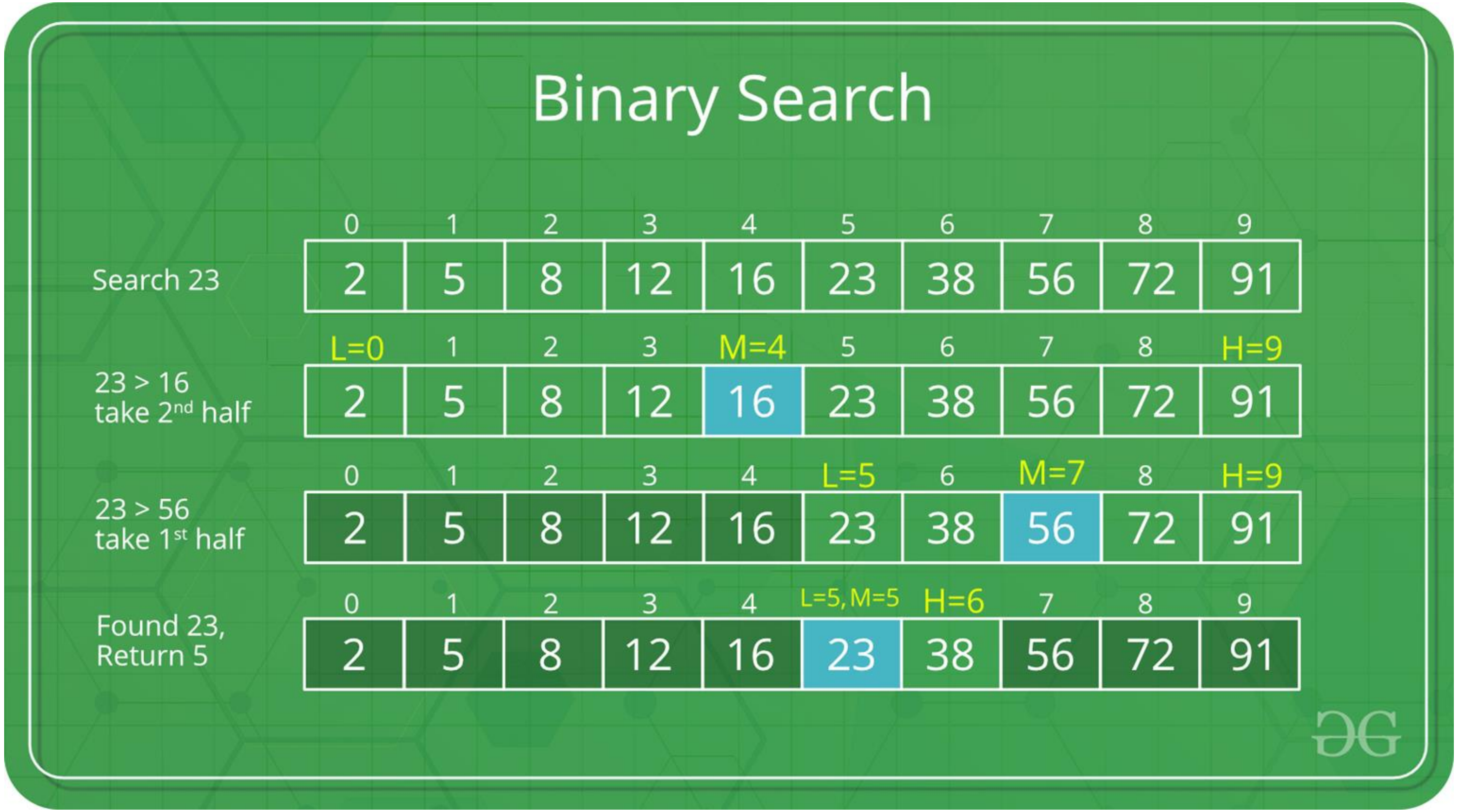
ALGORITHM FOR SEQUENTIAL SEARCH

- It is a simple algorithm that searches for a specific item inside a list. It operates looping on each element O(n) unless and until a match occurs or the end of the array is reached.
- algorithm Seqnl_Search(list, item)
- Pre: list != ;
- Post: return the index of the item if found, otherwise: 1
- index <- fi
- while index < list.Cnt and list[index] != item //cnt: counter variable
- index <- index + 1
- end while
- if index < list.Cnt and list[index] = item
- return index
- end if
- return: 1
- end Seqnl_Search

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

BINARY SEARCH

- Binary search is a very fast and efficient searching technique. It requires the list to be in sorted order. In this method, to search an element you can compare it with the present element at the center of the list. If it matches, then the search is successful otherwise the list is divided into two halves: one from the 0^{th} element to the middle element which is the center element (first half) another from the center element to the last element (which is the 2^{nd} half) where all values are greater than the center element.
- The searching mechanism proceeds from either of the two halves depending upon whether the target element is greater or smaller than the central element. If the element is smaller than the central element, then searching is done in the first half, otherwise searching is done in the second half.



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

ALGORITHM FOR BINARY SEARCH

- algorithm Binary_Search(list, item)
- Set L to 0 and R to n: 1
- if L > R, then Binary_Search terminates as unsuccessful
- else
- Set m (the position in the mid element) to the floor of (L + R) / 2
- if Am < T, set L to m + 1 and go to step 3
- if Am > T, set R to m: 1 and go to step 3
- Now, Am = T,
- the search is done; return (m)

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY