Data Structure

CSE 213

Submitted To :

Richard Philip
Lecturer Dept. Of CSE
City University

Submitted By :

Tasmina akter
ID No: 171442572
City University

Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way.

Data Structures is about rendering data elements in terms of some relationship, for better organization and storage.

For example, we have some data which has, player's name "Virat" and age 26. Here "Virat" is of String data type and 26 is of integer data type.

We are discuss some topics of data structure are given below :

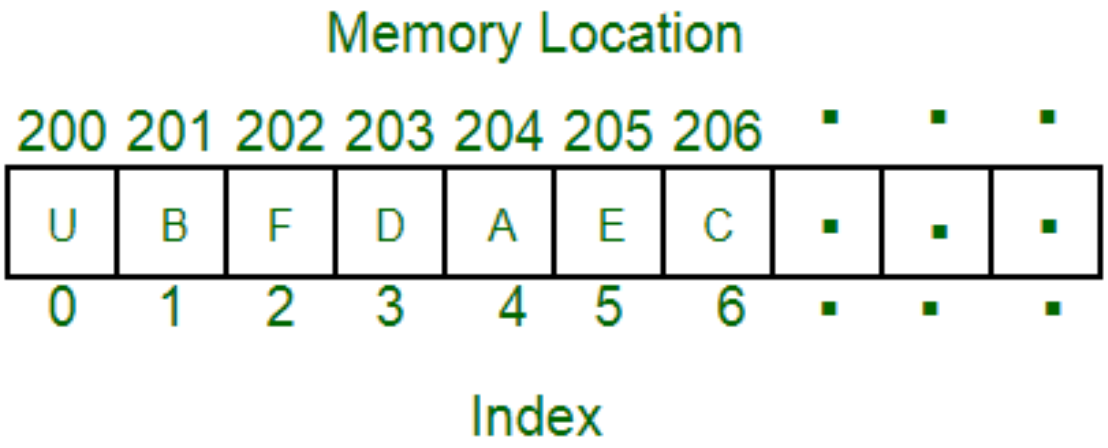
- Array
- Stack
- Queue
- Linked List

Mid Term:

2. Array

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value.

The memory location of the first element of the array



The above image can be looked as a top-level view of a staircase where you are at the base of the staircase. Each element can be uniquely identified by their index in the array.

There are two types of array:

1. 1D array
2. 2D array.

2.1 1D Array

An array is stored such that the position of each element can be computed from its index by a mathematical formula. The simplest type of data structure is a linear array, also called one-dimensional array.

Int main()

```
for(i=0; i<n; i++){
Scanf("%d",&a[i]);
}
```

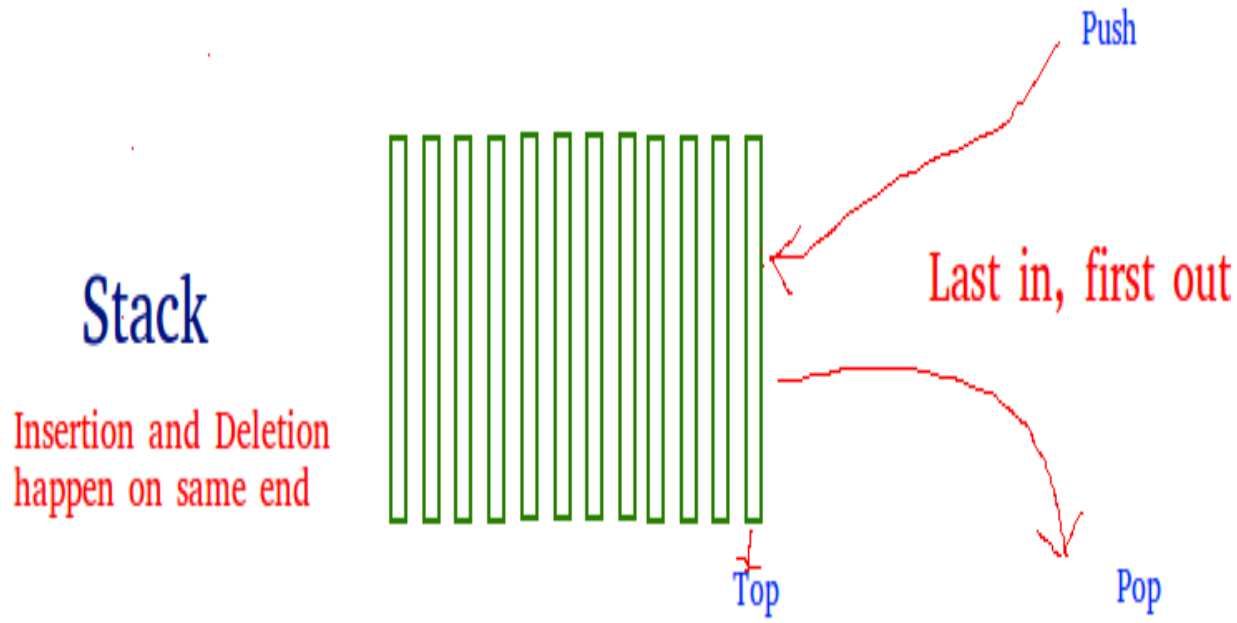
2.2 2D Array

Two-dimensional (2D) arrays are indexed by two subscripts, one for the row and one for the column.

```
int a[10][10]
for(i=0; i<rows; i++){
for(j=0;j<columns++){
}
}
```

3. Stack

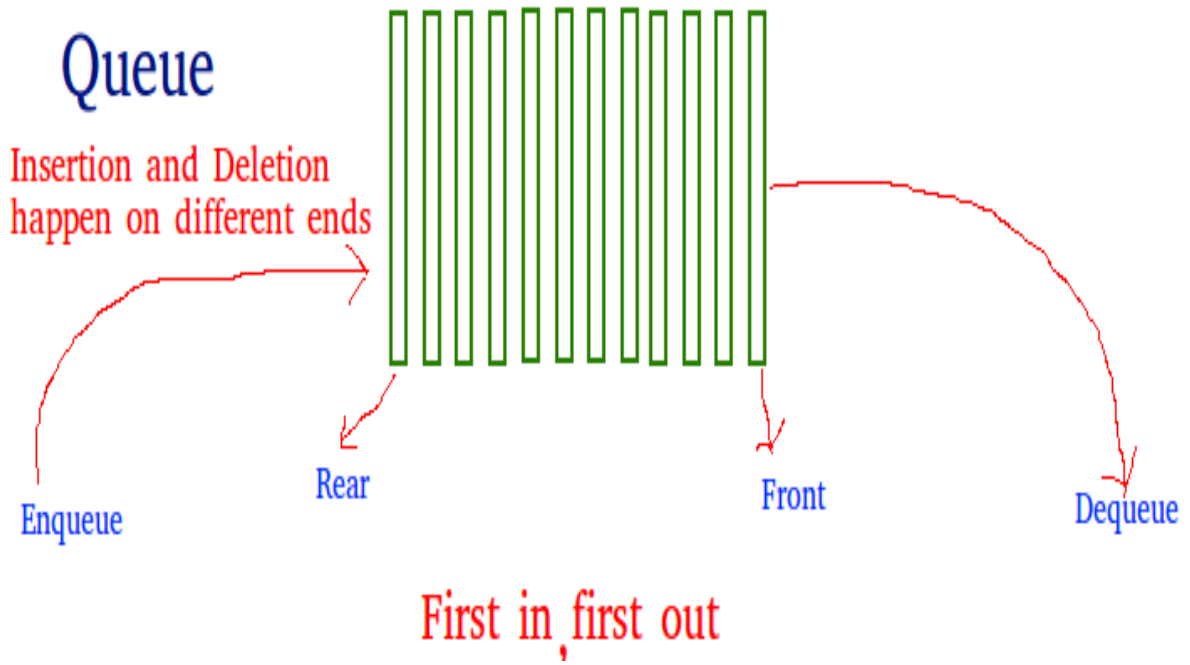
Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).



There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen. The plate which is at the top is the first one to be removed

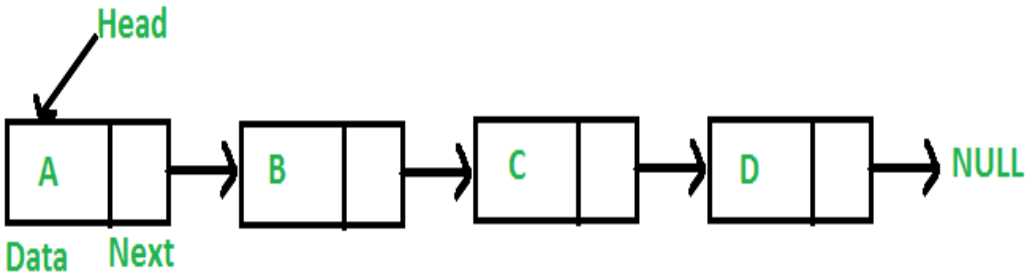
4. Queue

A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stack and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.



5. Linked List

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:



In simple words, a linked list consists of nodes where each node contains a data field and a reference to the next node in the list.

Linked list are two types:

1. Single linked list ,
2. Double linked list.

5.1 Single linked list

It is the most common. Each node has data and a pointer to the next node.

Node is represented as:

```
struct node {
    int data;
    struct node *next;
}
```

5.2 Double linked list

We add a pointer to the previous node in a doubly linked list.

A node is represented as:

```
struct node {
    int data;
    struct node *next;
    struct node *prev;
}
```

Final Term:

6. Tree

Trees: Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures.
Tree Vocabulary: The topmost node is called root of the tree. The elements that are directly under an element are called its children. The element directly above something is called its parent. For example, ‘a’ is a child of ‘f’, and ‘f’ is the parent of ‘a’. Finally, elements with no children are called leaves.

```
     tree
     ----
      j    <-- root
    /   \
   f     k
 /   \     \
a     h     z    <-- leaves
```

Traversals

A traversal is a process that visits all the nodes in the tree. Since a tree is a nonlinear data structure, there is no unique traversal. We will consider several traversal algorithms with we group in the following two kinds

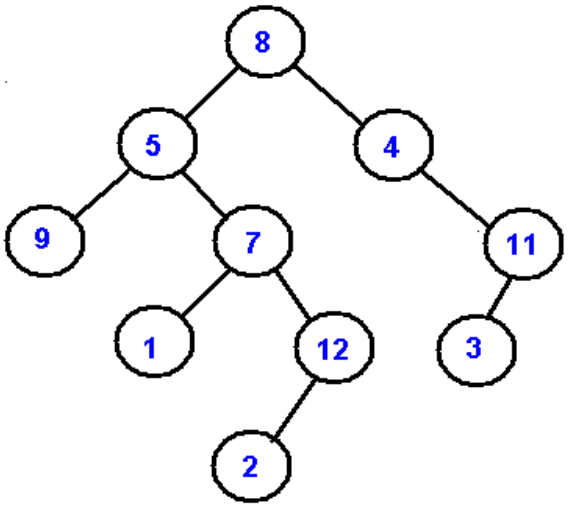
- depth-first traversal
- breadth-first traversal

There are three different types of depth-first traversals, :

- PreOrder traversal - visit the parent first and then left and right children;
- InOrder traversal - visit the left child, then the parent and the right child;
- PostOrder traversal - visit left child, then the right child and then the parent;

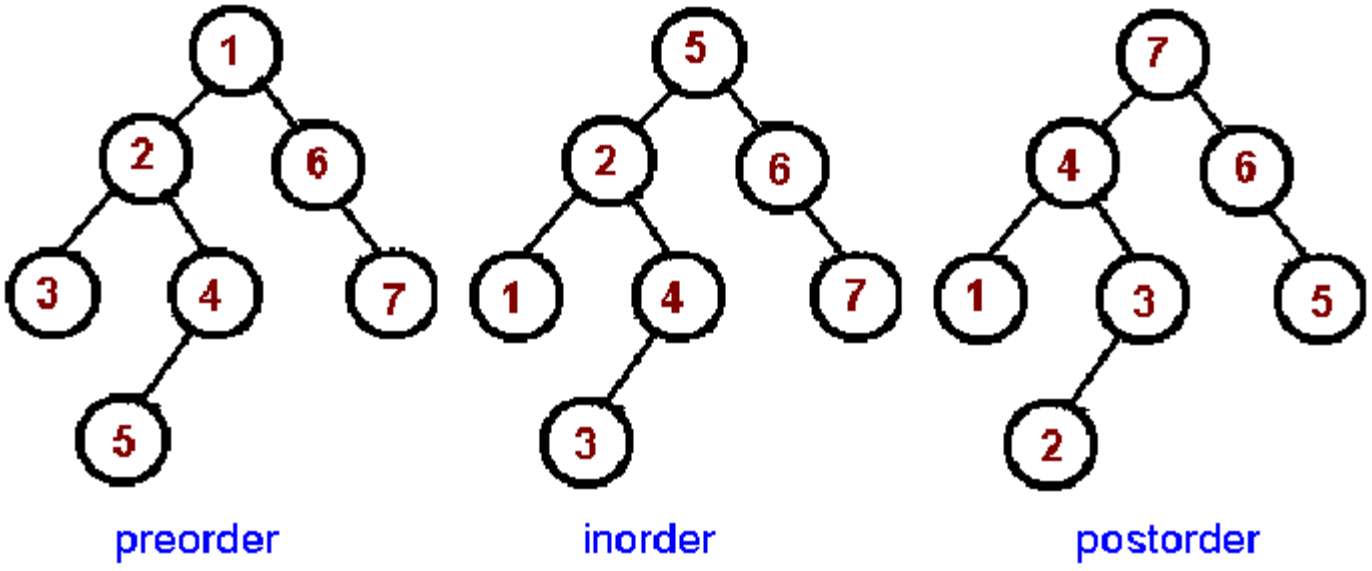
There is only one kind of breadth-first traversal--the level order traversal. This traversal visits nodes by levels from top to bottom and from left to right.

As an example consider the following tree and its four traversals:

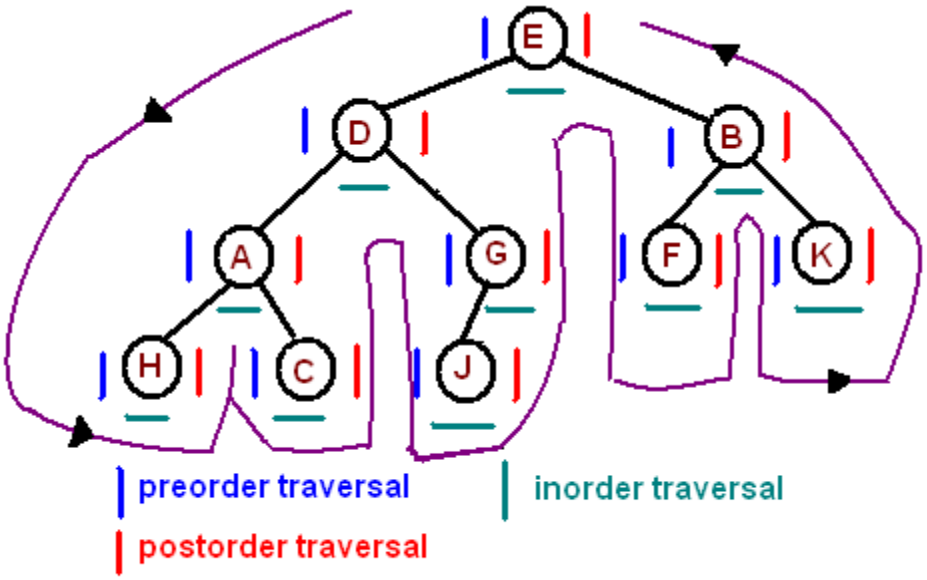


PreOrder - 8, 5, 9, 7, 1, 12, 2, 4, 11, 3
InOrder - 9, 5, 1, 7, 2, 12, 8, 4, 3, 11
PostOrder - 9, 1, 2, 12, 7, 5, 3, 11, 4, 8
LevelOrder - 8, 5, 4, 9, 7, 11, 1, 12, 3, 2

In the next picture we demonstarte the order of node visitation. Number 1 denote the first node in a particular traversal and 7 denote the last node.



These common traversals can be represented as a single algorithm by assuming that we visit each node three times. An *Euler tour* is a walk around the binary tree where each edge is treated as a wall, which you cannot cross. In this walk each node will be visited either on the left, or under the below, or on the right. The Euler tour in which we visit nodes on the left produces a preorder traversal. When we visit nodes from the below, we get an inorder traversal. And when we visit nodes on the right, we get a postorder traversal.



7. Selection Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.
2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element from the unsorted subarray is picked and moved to the sorted subarray.

Following example explains the above steps:

```
arr[] = 64 25 12 22 11


// Find the minimum element in arr[0...4]
// and place it at beginning
11 25 12 22 64

// Find the minimum element in arr[1...4]
//   and   place   it   at   beginning   of
arr[1...4]
11 12 25 22 64
```

```
// Find the minimum element in arr[2...4]
//   and   place   it   at   beginning   of
arr[2...4]
11 12 22 25 64

// Find the minimum element in arr[3...4]
//   and   place   it   at   beginning   of
arr[3...4]
11 12 22 25 64
```

8. Buble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.
Example:
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.
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.
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)

9. Insertion Sort

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort.
The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$, where **n** is the number of items.

10. Merge Sort

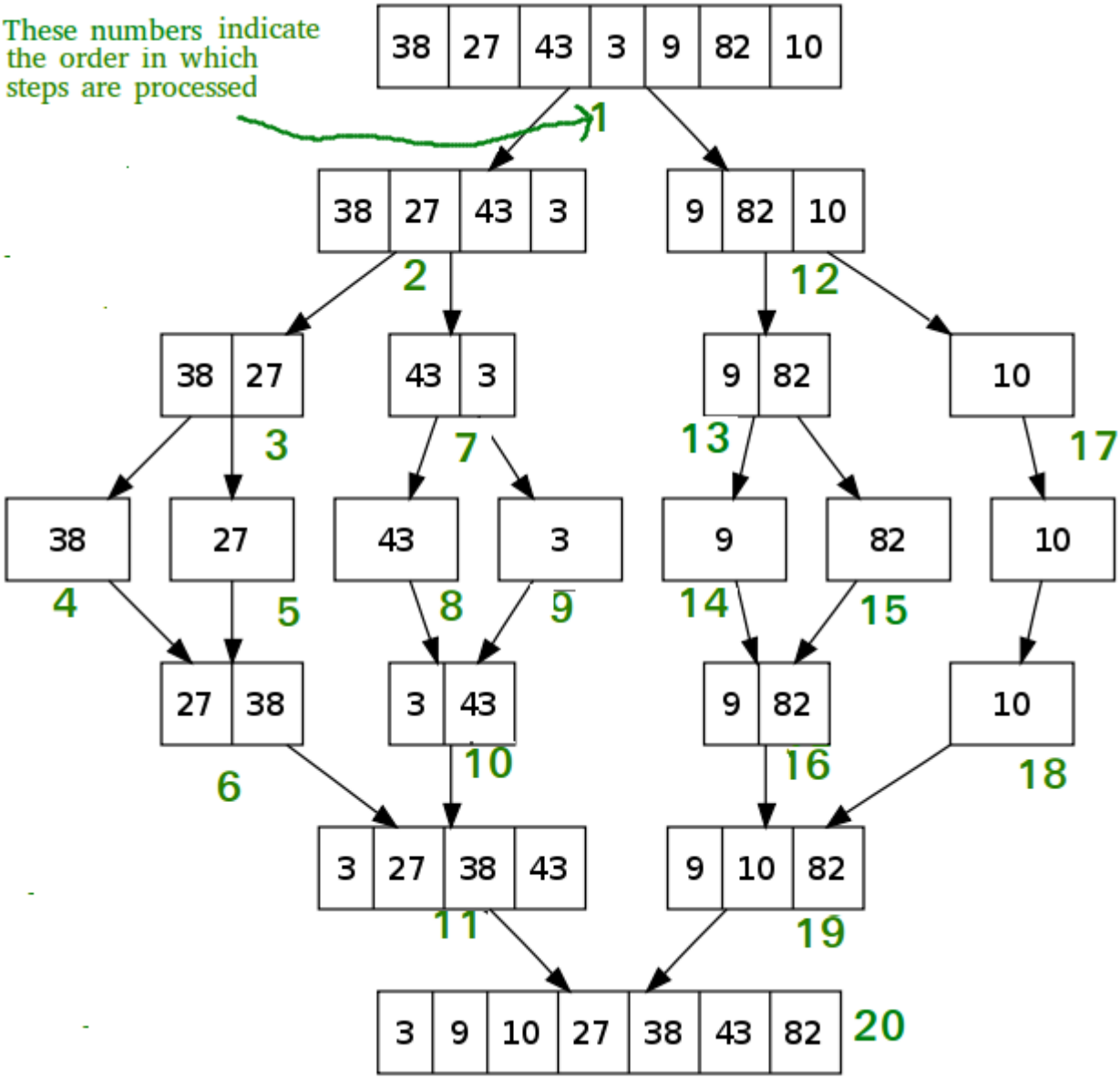
Like QuickSort, Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one. See following C implementation for details.

MergeSort(arr[], l, r)
If r > l

1. Find the middle point to divide the array into two halves:
   middle m = (l+r)/2
2. Call mergeSort for first half:
   Call mergeSort(arr, l, m)
3. Call mergeSort for second half:
   Call mergeSort(arr, m+1, r)
4. Merge the two halves sorted in step 2 and 3:
   Call merge(arr, l, m, r)

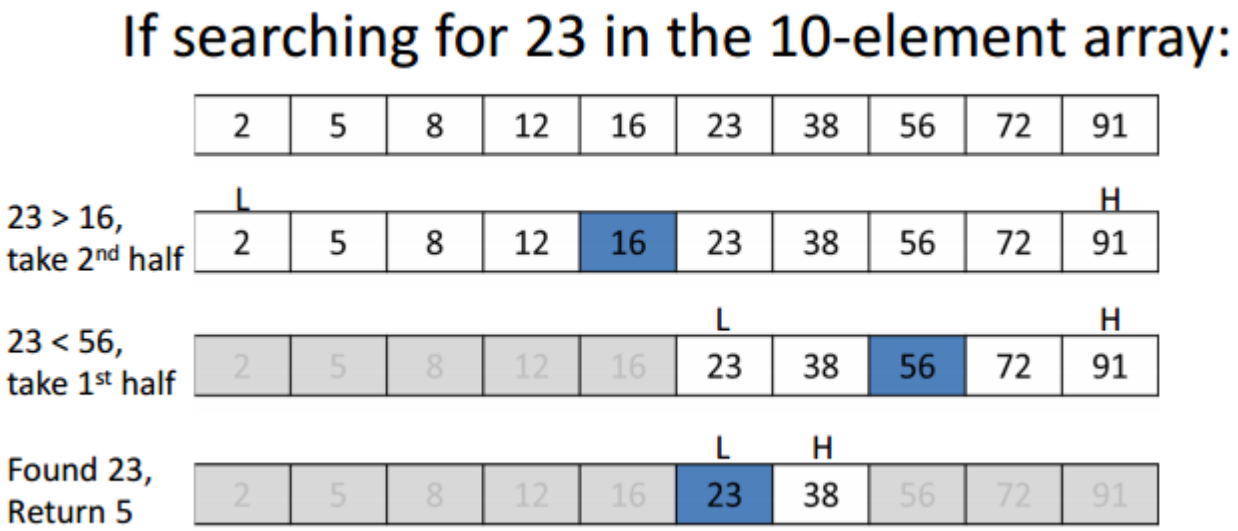
The following diagram from wikipedia shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}. If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge

processes comes into action and starts merging arrays back till the complete array is merged.



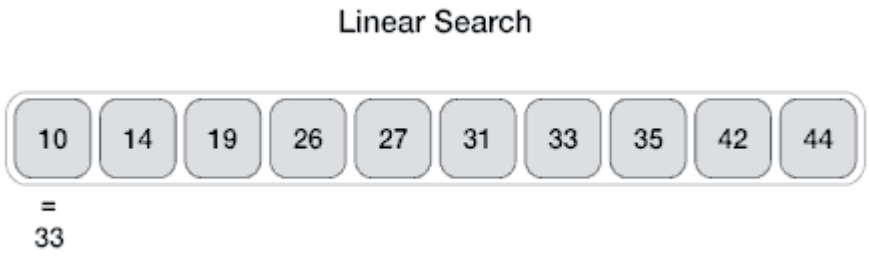
11. Binary Search

Given a sorted array arr[] of n elements, write a function to search a given element x in arr[].
A simple approach is to do linear search.The time complexity of above algorithm is O(n). Another approach to perform the same task is using Binary Search.
Binary Search: Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.
Example :



12. Linear Search

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.



Algorithm

Linear Search (Array A, Value x)

Step 1: Set i to 1
Step 2: if i > n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit

References

- GeeksforGeeks
  https://www.geeksforgeeks.org/data-structures/
- Programiz
  https://www.programiz.com/