Faculty of Computing

Logo, company name

Description automatically generated

Data Structure and Algorithm

Lab Manual

**Lab 10: Recursion and Searching Algorithms**

**Objectives:**

* Understand and implement Linear searching
* Understand and implement Binary searching
* Understand recursion

**Introduction**

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

**Activity Time boxing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task No.** | **Activity Name** | **Activity time** | **Total Time** |
| 1 | Lab Manual Lecture | 20 mins |  |
| 2. | Example | 10 mins |  |
| 3 | Walkthrough Tasks | 10 mins |  |
| 4. | Lab Tasks | 9100 mins |  |
| 5. | Evaluation | 30 mins | 170 mins |

**Concept Map**

**Recursion:** The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc. The idea is to represent a problem in terms of one or more smaller problems and add one or more base conditions that stop the recursion. For example, we compute factorial n if we know factorial of (n-1). The base case for factorial would be n = 0. We return 1 when n = 0.

There are two types of recursion

1. **Direct recursion:** A function fun is called direct recursive if it calls the same function fun.
2. **Indirect recursion:** A function fun is called indirect recursive if it calls another function say fun\_new and fun\_new calls fun directly or indirectly.

**Properties:** A recursive function can go infinite like a loop. To avoid infinite running of recursive function, there are two properties that a recursive function must have

1. Base criteria − There must be at least one base criteria or condition, such that, when this condition is met the function stops calling itself recursively.
2. Progressive approach − The recursive calls should progress in such a way that each time a recursive call is made it comes closer to the base criteria.

**Implementation**

A picture containing text, electronics

Description automatically generatedMany programming languages implement recursion by means of stacks. Generally, whenever a function (caller) calls another function (callee) or itself as callee, the caller function transfers execution control to the callee. This transfer process may also involve some data to be passed from the caller to the callee. This implies, the caller function has to suspend its execution temporarily and resume later when the execution control returns from the callee function. Here, the caller function needs to start exactly from the point of execution where it puts itself on hold. It also needs the exact same data values it was working on. For this purpose, an activation record (or stack frame) is created for the caller function. This activation record keeps the information about local variables, formal parameters, return address and all information passed to the caller function.

**Factorial by recursion (Code)**

def factorial(n):

if n == 0:

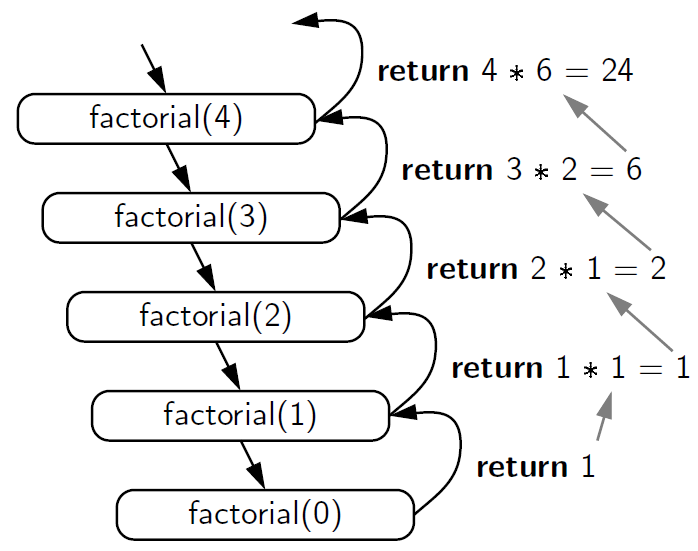
return 1

else:

return n \* factorial(n-1)

5! 5\*4\*3\*2\*1

0! 1



**Searching Algorithms**

1. **Linear 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.

Its best execution time is one, whereas the worst execution time is n, where n is the total number of items in the search array.

**Time Complexity**

* Best- case complexity = O(1) occurs when the search element is present at the first element in the search array.
* Worst- case complexity = O(n) occurs when the search element is not present in the set of elements or array.
* Average complexity = O(n) is referred to when the element is present somewhere in the search array.

**Code:**

public static int search(int arr[], int x)

{

int n = arr.length;

for (int i = 0; i < n; i++)

{

if (arr[i] == x)

return i; }

return -1; }

// Driver code

public static void main(String args[])

{

int arr[] = { 2, 3, 4, 10, 40 };

int x = 10;

// Function call

int result = search(arr, x);

if (result == -1)

System.out.print(

"Element is not present in array");

else

System.out.print("Element is present at index "

+ result);

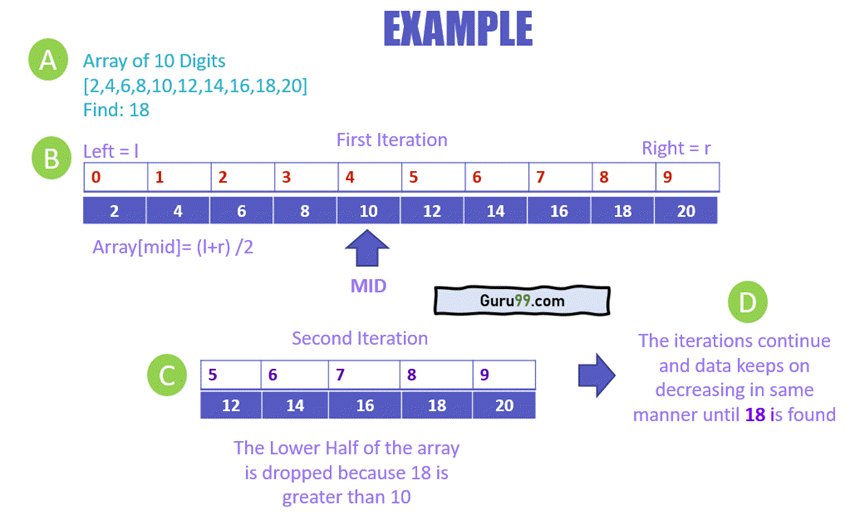
}

1. **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 0th element to the middle element which is the Center element (first half) another from the Center element to the last element (which is the 2nd 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.

* **Run-time complexity =** O(log n)



**Code**

class BinarySearch {

// Returns index of x if it is present in arr[l..

// r], else return -1

int binarySearch(int arr[], int l, int r, int x)

{

if (r >= l) {

int mid = l + r / 2;

// If the element is present at the middle itself

if (arr[mid] == x)

return mid;

// If element is smaller than mid, then it can only be present in left subarray

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

// Else the element can only be present

// in right subarray

return binarySearch(arr, mid + 1, r, x);

}

// We reach here when element is not present

// in array

return -1;

}

// Driver method to test above

public static void main(String args[])

{

BinarySearch ob = new BinarySearch();

int arr[] = { 2, 3, 4, 10, 40 };

int n = arr.length;

int x = 10;

int result = ob.binarySearch(arr, 0, n - 1, x);

if (result == -1)

System.out.println("Element not present");

else

System.out.println("Element found at index "

+ result);

}}

**Practice Tasks**

**Lab Task 1:**

A lottery ticket buyer purchases 10 tickets a week, always playing the same 10 5-digit “lucky” combinations. Write a program that initializes an array or a vector with these numbers and then lets the player enter this week’s winning 5-digit number. 55555

The program should perform a linear search through the list of the player’s numbers and report whether or not one of the tickets is a winner this week.

Here are the numbers:

13579 26791 26792 33445 55555

62483 77777 79422 85647 93121

If the user enters a number that is not in the array, the program should display a message indicating that the number is invalid.

Q2. Write a recursive function to perform binary search on a sorted array.

Q3. Write a recursive function to perform linear search on an array.

Q4. Write a recursive function to calculate the factorial of a number.

Q5. Write a recursive function to calculate the sum of elements in an array.

Percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

**Further Reading**

The slides and reading material can be accessed from the folder of the class instructor available at Moellim.

**Out comes**

The outcomes of this lab were:

* Understand and implement Linear searching
* Understand and implement Binary searching
* Understand recursion