Exercise 2: E-commerce Platform Search Function

Q). Explain Big-O notation and how it helps in analyzing algorithms.

-> Big-O notation is a mathematical notation used in computer science to describe the upper bound of an algorithm's time or space complexity as the input size grows. It focuses on how the algorithm's resource usage(time or memory) scales with the input, rather than providing exact measurements. This helps in comparing the efficiency of different algorithms and choosing the most suitable one for a given task.

Q). Describe best, average and worst-case scenarios for search operations and compare the time complexities of linear and binary search algorithms.

-> The two most commonly used searching algorithms are Linear Search Algorithm and Binary Search Algorithm.Their best, average and worst-case time complexities are :

Linear Search:

1. Best Case : O(1) as the key might be present at the first index.

2. Average Case : O(N) as the key might be present anywhere in the array or list.

3. Worst Case : O(N) as the key might be present at the last index, i.e completely opposite to the end of where the search in the list has started, where N is the size of the list.

Auxiliary space : O(1) as except for the variable to iterate through the list, no other variable is used.

Binary Search:

1. Best Case : O(1) as the key might be present at the middle of the list and binary search starts from the middle.

2. Average Case : O(log N) as it divides the array in half at each step.

3. Worst Case : O(log N) as it divides the array in half at each step, where N is the size of the list.

Auxiliary space : O(1) as no extra space is used.

Code :

package Algorithms.Exercise2;

import java.util.\*;

public class Product {

int productId;

String productName,category;

public void linearSearch(int s,Product list[])

{

int f=0;

for(int i=0;i<5;i++)

{

if(list[i].productId==s)

{

System.out.println("Product found");

System.out.println("Displaying details...");

System.out.println(list[i].productId);

System.out.println(list[i].productName);

System.out.println(list[i].category);

f=1;

break;

}

}

if(f==0)

{

System.out.println("Product not found");

}

}

public void binSearch(int s, Product list[])

{

int temp;

String t;

//implementing Selection Sort technique

for(int i=0;i<list.length-1;i++)

{

int min=i;

for(int j=i+1;j<list.length;j++)

{

if(list[j].productId<list[min].productId)

{

min=list[j].productId;

}

}

temp=list[i].productId;

list[i].productId=list[min].productId;

list[min].productId=temp;

t=list[i].productName;

list[i].productName=list[min].productName;

list[min].productName=t;

t=list[i].category;

list[i].category=list[min].category;

list[min].category=t;

}

//Now implementing Binary search on sorted array

int l=0,u=list.length-1,mid=0,f=0;

while(l<=u)

{

mid=(l+u)/2;

if(list[mid].productId==s)

{

System.out.println("Product found");

f=1;

break;

}

else if(s<list[mid].productId)

{

u=u-mid;

}

else if(s>list[mid].productId)

{

l=l+mid;

}

}

if(f==1)

{

System.out.println("Displaying details...");

System.out.println(list[mid].productId);

System.out.println(list[mid].productName);

System.out.println(list[mid].category);

}

else

{

System.out.println("Product not found");

}

}

public static void main(String[] args)

{

Scanner in=new Scanner(System.in);

int s,size,ch;

System.out.println("Enter total no. of products : ");

size=in.nextInt();

Product list[]=new Product[size];

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

{

list[i]=new Product();

}

System.out.println("Enter details of products : ");

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

{

System.out.println("Enter id, name and category of product "+(i+1)+": ");

list[i].productId=in.nextInt();

in.nextLine();

list[i].productName=in.nextLine();

list[i].category=in.nextLine();

}

do

{

System.out.println("Enter product id to be searched : ");

s=in.nextInt();

Product obj=new Product();

obj.linearSearch(s, list);

Product obj1=new Product();

obj1.binSearch(s, list);

System.out.println("Press 1 to continue or 0 otherwise : ");

ch=in.nextInt();

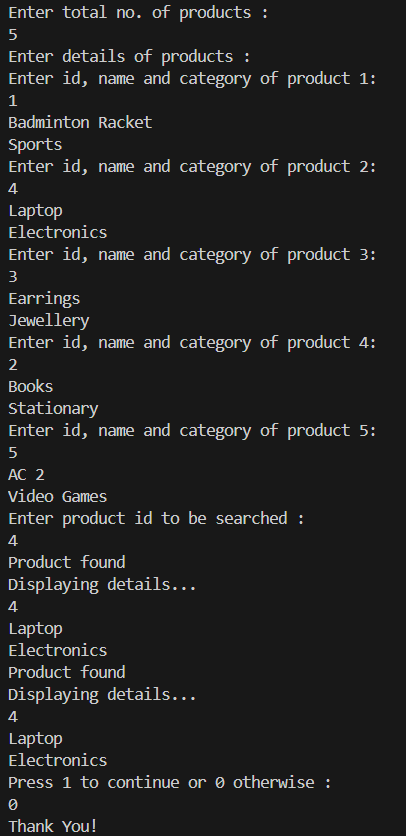
}while(ch==1);

System.out.println("Thank You!");

}

}

Output :



Q). Discuss which algorithm is suitable for your platform and why?

-> Binary Search is better than Linear Search for searching product IDs — **only if the product data is sorted**, because it is much faster (O(log n) vs O(n)).

**Exercise 7: Financial Forecasting**

Q). Explain the concept of recursion and how it can simplify certain problems.

-> The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

Recursion helps in logic building. Recursive thinking helps in solving complex problems by breaking them into smaller subproblems. The idea is to represent a problem in terms of one or more smaller subproblems, and add one or more base conditions that stop the recursion. A recursive program stops at a base condition. There can be more than one base condition. Recursion helps us in modeling solutions to complex problems using the Divide and Conquer method to design elegant solutions with clearer Code Clarity. Certain problems like the traversing tree structures or solving the Towers of Hanoi, are inherently recursive and have elegant recursive solutions.

Code :

package Algorithms.Exercise7;

class FinancialForecasting

{

public static double predictFutureValue(int currentPeriod, double initialValue, double growthRate)

{

if(currentPeriod==0)

{

return initialValue;

}

return predictFutureValue(currentPeriod-1, initialValue, growthRate)\*(1+growthRate);

}

public static void main(String[] args)

{

double initialValue=1000.0,growthRate=0.05,futureValue=0.0;

int futurePeriods=5;

futureValue=predictFutureValue(futurePeriods, initialValue, growthRate);

System.out.println("Predicted value after "+futurePeriods+" : "+futureValue);

}

}

Output :



Q). Discuss the time complexity of your recursive algorithm.

-> The recursive function predictFutureValues calls itself exactly once per recursive level, reducing currentPeriod by 1 each time,

until it reaches 0. This forms a chain of calls from N to 0.

Thus, the time complexity of the above recursive method is O(N), where N is the no. of times the function calls itself.

As regards the space complexity, each recursive call to the function adds a frame to the call stack. Since there are N recursive calls,

therefore N call stacks at once. Hence, space complexity of the above recursive function is O(N).

Q). Explain how to optimize the recursive solution to avoid excessive computation.

-> We can rewrite the recursive method iteratively as below to reduce its space complexity :

public static double predictFutureValue(int periods, double initialValue, double growthRate)

{

double value=initialValue;

for(int i=1;i<=periods;i++)

value\*=(1+growthRate);

return value;

}

The time complexity remains the same however, the space complexity reduces to O(1) as it no longer takes extra memory which is

more efficient for large values of N.