**INDEX:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SN | Topic | Date of Experiment | Date of submission | Signature | Remark |
| 9. | Solve various problems using recursion. |  |  |  |  |
| 10. | Perform sorting in ascending order using various algorithm. |  |  |  |  |
| 11. | Algorithm of collision resolution techniques. |  |  |  |  |
| 12. | Operations on Binary search tree. |  |  |  |  |

**EXPERIMENT: 9**

**TITLE:**

A java program to solve various problems using recursion.

**1.OBJECTIVE:**

1.1. To implement Tower of Hanoi problem.

1.2. To implement factorial.

1.3. To implement Fibonacci series.

1.4. To implement sum of first n natural numbers.

* 1. To find the power of the given number.

**2.THEORY:**

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method. Using a recursive algorithm, certain problems can be solved easily such as Tower of Hanoi(TOH). For the problems to solve recursively following two conditions must be satisfied:

2.1. Each time a function call itself there must be a rule specified that takes closer to the solution.

2.2.There must be a stopping condition or anchor condition to stop the repetition.

**3.IMPLEMENTATION:**

The following are the implementation of various java recursion programs:

**3.1. Program 1:** Implementation of factorial:

SOURCE CODE

***class factorial***

*{*

***public static void main(String[] args)***

*{*

*int n=5;*

*int result=fact(n);*

*System.out.println("The factorial of "+n+" is "+result);*

*}*

***static int fact(int n)***

*{*

*if(n==0)*

*{*

*return 1;*

*}*

*else*

*{*

*return n\*fact(n-1);*

*}*

OUTPUT:

The factorial of 5 is 120

**3.2. Program 2:** Implementation of Tower of Hanoi problem.

SOURCE CODE

***class toh***

*{*

***public static void main(String[] args)***

*{*

*toh t=new toh();*

*int n=3;*

*char a='A';*

*char b='B';*

*char c='C';*

*t.toh1(n,a,b,c);*

*}*

***void******toh1****(****int n,char a,char b,char c)***

*{*

*if(n>=1)*

*{*

*toh1(n-1,a,c,b);*

*System.out.println("Move "+n+" from "+a+" to "+c);*

*toh1(n-1,b,a,c);*

*}*

*}*

*}*

OUTPUT:

Move1 from A to C

Move2 from A to B

Move1 from C to B

Move3 from A to C

Move1 from B to A

Move2 from B to C

Move1 from A to C

**3.3. Program 3:** Implementation of Fibonacci series:

SOURCE CODE

***class fibonacci***

*{*

***public static void main(String[] args)***

*{*

*int n=5;*

*int result=fibo(n);*

*System.out.println("The "+n+"th term of fibonacci series is "+result);*

*}*

***static int fibo(int n)***

*{*

*if(n==0)*

*return 0;*

*else if(n==1)*

*return 1;*

*else*

*return fibo(n-1)+fibo(n-2);*

*}*

*}*

OUTPUT:

The 5th term of fibonacci series is 5

**3.4. Program 4:** Implementation of sum of first n natural numbers.

SOURCE CODE

***class sumofnatno***

*{*

***public static void main(String[] args)***

*{*

*sumofnatno s=new sumofnatno();*

*int n=5;*

*int result=s.sum(n);*

*System.out.println("The sum of first 5 natural numbers is"+result);*

*}*

***int sum(int n)***

*{*

*if(n>1)*

*{*

*return n+sum(n-1);*

*}*

*else*

*return 1;*

*}*

*}*

OUTPUT:

The sum of first 5 natural numbers is 15

**3.5. Program 5:** Implementation of power of a number:

SOURCE CODE

***class power***

*{*

***public static void main(String[]*** *args)*

*{*

*int base=5;*

*int exp=3;*

*int result=pow(base,exp);*

*System.out.println(base+" to the power "+exp+" is "+result);*

*}*

***static int pow(int b,int e)***

*{*

*if(e==0)*

*{*

*return 1;*

*}*

*else*

*{*

*return b\*pow(b,e-1);*

*}*

*}*

*}*

OUTPUT:

5 to the power 3 is 125

**4. OUTPUT AND DISCUSSION:**

The TOH problem, factorial, power of the number, sum of n natural numbers and Fibonacci series were implemented using recursion i.e, calling itself.

**5. CONCLUSION:**

A Java program to implement TOH problem, factorial, Fibonacci series, sum of n natural numbers and power of a number is successfully run.

**EXPERIMENT: 10**

**TITLE:**

A java program to sort the arrays in ascending order.

**1.OBJECTIVE:**

1.1. To implement Quick Sort.

1.2. To implement Merge Sort.

1.3. To implement Bubble Sort.

1.4. To implement Selection Sort.

**2.THEORY:**

A sorting algorithm is used to re-arrange a given array or list of elements according to a comparison operator on the elements. Java supports a variety of sorting algorithms. Most of the algorithm are flexible themselves and can be implemented with both recursive as well as an iterative approach.

**3.IMPLEMENTATION:**

The following are the implementation of popular sorting algorithm in java.

**3.1. Program 1:** Implementation of Insertion Sort algorithm.

SOURCE CODE

***class insertionsort***

*{*

***public void sort(int A[])***

*{*

*for(int j=1;j<A.length; j++)*

*{*

*int key=A[j];*

*int i=j-1;*

*while(i>=0 && key<A[i])*

*{*

*A[i+1]=A[i];*

*i--;*

*}*

*A[i+1]=key;*

*}*

*}*

***public void display(int A[])****{*

*for(int i=0;i<A.length;i++)*

*{*

*System.out.print(A[i]+" ");*

*}*

*System.out.println();*

*}*

***public static void main(String args[])***

*{*

*int A[]={9,8,7,6,5,4,3,2,1};*

*insertionsort is=new insertionsort();*

*System.out.println("Array before sorting");*

*is.display(A);*

*System.out.println("Array after Insertion sort");*

*is.sort(A);*

*is.display(A);*

*}*

*}*

OUTPUT:

Array before sorting

9 8 7 6 5 4 3 2 1

Array after Insertion sort

1 2 3 4 5 6 7 8 9

**3.2. Program 2:** Implementation of Selection Sort algorithm.

SOURCE CODE

***class selectionsort***

*{*

***void sorting(int A[])***

*{*

*for(int i=0;i<A.length;i++)*

*{*

*int min=i;*

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

*{*

*if(A[j]<A[min])*

*{*

*min=j;*

*}*

*}*

*int temp=A[i];*

*A[i]=A[min];*

*A[min]=temp;*

*}*

*}*

***void display(int A[])***

*{*

*for(int i=0;i<A.length;i++)*

*{*

*System.out.print(A[i]+",");*

*}*

*}*

***public static void main(String args[])***

*{*

*int A[]={10,9,8,7,6,5,4,3,2,1};*

*selectionsort s=new selectionsort();*

*System.out.println("Array before sorting");*

*s.display(A);*

*System.out.println("Array after Selection sorting");*

*s.sorting(A);*

*s.display(A);*

*}*

OUTPUT:

Array before sorting

10,9,8,7,6,5,4,3,2,1,

Array after Selection sorting

1,2,3,4,5,6,7,8,9,10,

**3.3. Program 3:** Implementation of Quick Sort Algorithm:

SOURCE CODE

***class quicksort***

*{*

***public void quicksorting(int A[],int p,int q){***

*if(p<q){*

*int j=partition(A,p,q);*

*quicksorting(A,p,j-1);*

*quicksorting(A,j+1,q);*

*}*

*}*

***public int partition(int A[],int i,int j)***

*{*

*int L=i-1;*

*int pivot=A[j];*

*for(int k=i;k<=j;k++)*

*{*

*if(A[k]<pivot)*

*{*

*L++;*

*swap(A,L,k);*

*}*

*}*

*swap(A,L+1,j);*

*return L+1;*

*}*

***public void swap(int A[],int i,int j)***

*{*

*int temp=A[i];*

*A[i]=A[j];*

*A[j]=temp;*

*}*

***public void display(int A[])***

*{*

*for(int i=0;i<A.length;i++) {*

*System.out.print(A[i]+",");*

*}*

*System.out.println();*

*}*

*public static void main(String args[]){*

*int A[]={1,2,3,14,5,6,17,8,9,10};*

*quicksort qs=new quicksort();*

*System.out.println("Array before sorting");*

*qs.display(A);*

*System.out.println("Array after Quick Sort");*

*qs.quicksorting(A,0,A.length-1);*

*qs.display(A);*

*}*

*}*

OUTPUT:

Array before sorting

1,2,3,14,5,6,17,8,9,10,

Array after Quick Sort

1,2,3,5,6,8,9,10,14,17,

**4. OUTPUT AND DISCUSSION:**

In each of the program the arrays are sorted using various algorithm and the output resulted to be in ascending order.

**5. CONCLUSION:**

A Java program to sort the arrays using various algorithm is successfully run.

**EXPERIMENT: 11**

**TITLE:**

A java program of collision resolution techniques.

**1.OBJECTIVE:**

1.1. To solve collision through linear probing.

1.2. To solve collision using quadratic probing.

**2.THEORY:**

Hashing is an efficient searching technique in which key is placed in direct accessible address for rapid search. It uses a hashing function which maps the key with the corresponding key address or location. On the contrary, if there occurs a situation when two distinct pieces of data or key have same hash value, it is called collision.

When hash collision occurs, there must be a systematic method for placing the second data item in the hash table, which is called collision resolution. If hash function is perfect, collision never occurs. However, it is often not possible, collision resolution become a important part of hashing.

Some of the popular methods of collision resolution methods are:

**2.1. Open Addressing:**

**2.1.1. Linear Probing**: In this method, when collision occurs the item is put in the next empty location within the array space.

**2.1.2.Quadratic Probing:** In this method, when collision occurs the following function is used: (hash value+i2)/tablesize.

**2.1.3. Double Probing**.

**2.2.Rehashing.**

**2.3. Chaining.**

**2.4. Hashing using buckets.**

**3.IMPLEMENTATION:**

The following are the implementation of collision resolution techniques in java.

**3.1. Program 1:** Implementation of Linear Probing technique.

SOURCE CODE

***import java.util.Hashtable****;*

***class linearprobing***

*{*

*int hashtable[];*

*int m;*

*int n;*

*public linearprobing(int size)*

*{*

*m=size;*

*hashtable=new int[m];*

*n=0;*

*}*

***int hashfunction(int key****)*

*{*

*return key%m;*

*}*

***public void insert(int key)***

*{*

*if(n==m)*

*{*

*System.out.println("Hash table is full");*

*return;*

*}*

*int hash=hashfunction(key);*

*while(hashtable[hash]!=0)*

*{*

*System.out.println("\ncollision occured");*

*hash=(hash+1)%m;*

*}*

*hashtable[hash]=key;*

*System.out.println(key+"inserted at position"+hash);*

*n++;*

*}*

***public static void main(String args[])***

*{*

*linearprobing lp=new linearprobing(10);*

*lp.insert(5);*

*lp.insert(12);*

*lp.insert(15);*

*}*

OUTPUT:

5 inserted at position 5

12 inserted at position 2

collision occurred

15 inserted at position 6

**3.2. Program 2:** Implementation of Quadratic Probing.

SOURCE CODE

***class quadraticprobing***

*{*

*int hashtable[];*

*int m;*

*int n;*

***quadraticprobing(int size)***

*{*

*m=size;*

*hashtable=new int[m];*

*n=0;*

*}*

***int hashfunction(int key)***

*{*

*return key%m;*

*}*

***public void insert(int key)***

*{*

*if(n==m)*

*{*

*System.out.println("Hash table is full");*

*return;*

*}*

*int hash=hashfunction(key);*

*int i=0;*

*while(hashtable[hash]!=0)*

*{*

*i++;*

*System.out.println("collision occurred");*

*hash=(hash+i\*i)%m;*

*}*

*hashtable[hash]=key;*

*System.out.println(key+"inserted at position"+hash);*

*n++;*

*}*

***public static void main(String args[])***

*{*

*quadraticprobing qp=new quadraticprobing(10);*

*qp.insert(5);*

*qp.insert(15);*

*qp.insert(25);*

*qp.insert(1);*

*}*

*}*

OUTPUT:

5 inserted at position 5

collision occurred

15 inserted at position 6

collision occurred

collision occurred

25 inserted at position 0

1 inserted at position 1

**4. OUTPUT AND DISCUSSION:**

The first program solves the collision by using (hashvalue + i)%m function whereas the quadratic probing used (hashvalue+i2)%m.

**5. CONCLUSION:**

A Java program to solve the occurrence of collision is successfully run.

**EXPERIMENT: 12**

**TITLE:**

A java program to run operations of binary search tree.

**1.OBJECTIVE:**

1.1. To search the node in binary search tree.

1.2.To insert the nodes in binary search tree.

1.3. To delete the node from the binary search tree.

1.4. To traverse through the binary search tree.

**2.THEORY:**

A binary tree is Binary Search Tree that has following properties:

2.1. All the keys at left subtree are smaller than value at root.

2.2. All the values of right subtree are greater than the value at the root.

2.3. Both left and right subtree are BST recursively.

**3.IMPLEMENTATION:**

The following are the implementation of various operations of binary search tree such as insertion, deletion, search and traverse.

**3.1. Program 1:** Implementation of operations of binary search tree.

SOURCE CODE

***class binarysearchtree***

*{*

***class node***

*{*

*node left;*

*node right;*

*int data;*

*node(int data)*

*{*

*this.data=data;*

*this.left=null;*

*this.right=null;*

*}*

*}*

*public node root=null;*

***public void insert(int data)***

*{*

*root=insertNode(root,data);*

*}*

***public void delete(int data)***

*{*

*root=deleteNode(root,data);*

*}*

*public boolean search(int data)*

*{*

*return searchNode(root,data);*

*}*

***node insertNode(node root,int data)***

*{*

*if(root==null)*

*{*

*return new node(data);*

*}*

*else if(data<root.data)*

*{*

*root.left=insertNode(root.left,data);*

*}*

*else if(data>root.data)*

*{*

*root.right=insertNode(root.right,data);*

*}*

*return root;*

*}*

***node deleteNode(node root,int data****)*

*{*

*if(root==null)*

*{*

*return null;*

*}*

*else if(data<root.data)*

*{*

*root.left=deleteNode(root.left,data);*

*}*

*else if(data>root.data)*

*{*

*root.right=deleteNode(root.right,data);*

*}*

*else*

*{*

*if(root.left==null)*

*{*

*return root.right;*

*}*

*else if(root.right==null)*

*{*

*return root.left;*

*}*

*node temp=minNode(root);*

*root.data=temp.data;*

*root.right=deleteNode(root.right,root.data);*

*}*

*return root;*

*}*

***private node minNode(node root****)*

*{*

*if(root.left!=null)*

*{*

*return minNode(root.left);*

*}*

*else*

*{*

*return root;*

*}*

*}*

***private boolean searchNode(node root,int data)***

*{*

*if(root==null)*

*return false;*

*else if(root.data==data)*

*return true;*

*else if(data<root.data)*

*return searchNode(root.left,data);*

*else*

*return searchNode(root.right,data);*

*}*

***public void inorder()***

*{*

*inorderTraverse(root);*

*}*

***private void inorderTraverse(node root)***

*{*

*if(root!=null)*

*{*

*inorderTraverse(root.left);*

*System.out.print(root.data+" ");*

*inorderTraverse(root.right);*

*}*

*}*

***public static void main(String args[])***

*{*

*binarysearchtree bst=new binarysearchtree();*

*bst.insert(11);*

*bst.insert(21);*

*bst.insert(31);*

*System.out.println("Inorder traverse of binary tree");*

*bst.inorder();*

*System.out.println();*

*int searchValue=31;*

*boolean isFound=bst.search(searchValue);*

*if(isFound)*

*System.out.println(searchValue+"is found");*

*else*

*System.out.println(searchValue+"not found");*

*int deleteValue=21;*

*bst.delete(deleteValue);*

*System.out.println("After deleting"+deleteValue+"the inorder traverse:");*

*bst.inorder();*

*}*

*}*

OUTPUT:

Inorder traverse of binary tree

11 21 31

31 is found

After deleting 21

the inorder traverse:

11 31

**4. OUTPUT AND DISCUSSION:**

The above program run the various operations on binary search tree such as insertion, searching, traverse, deletion of the node and finding the min node from the binary search tree.

**5. CONCLUSION:**

A Java program of various operations of BST is successfully run.