**practical 1**

**Aim: To implement searching techniques using JAVA**

1. Linear search

2. Binary Search

3. Recursive binary search

**Objectives:**

1. To know what searching is and  understand algorithm for Linear search and Binary search
2. To understand the basic principles of recursive definitions and functions and be able to write simple recursive function

**Theory:**

Searching: It is process of checking and finding an element from list of elements.

1. Linear search
2. Binary search

**Pseudocode for Linear search:**

Algorithm linear (a, n, key)

// key is data to be searched in array a of size length

Pre:  Unsorted list of length n.

Post:    If found, return position of key in array a. If key not  present in list, return negative value

1. for i = 0 to (n - 1)  do

 if  (key == a[i])

return i

2.  return -1

**Pseudocode for Binary search:**

Algorithm binary\_search (a, n, key)

// key - data to be searched in array a of size n

Pre: Sorted list of length n.

Post: If present, return position of key in array a; Else return -1

1. low =0
2. high = n-1
3. while (low <= high)
   1. mid = (low + high)/2
   2. if (key  == a[mid])

return mid

3. if ( key  <  a[mid])

high = mid -1

4        else

low = mid + 1

  4. return -1

**Pseudocode for Recursive binary search:**

 BinarySearch(a,  key,  low,  high)

1. if (low > high)

         return -1                          // not found

     2. mid = (low + high) / 2

     3. if (key < a[mid] )

         return BinarySearch ( a, key, low,  mid-1)

      4.  else if (key > a[mid])

         return BinarySearch ( a, key, mid+1,  high)

     5. else

         return mid                       // found

**Program:**

**JAVA implementation for Linear search:**

**package LinearSearch;**

**import java.util.Scanner;**

**public class Linearstatic {**

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

**Scanner sc = new Scanner(System.*in*);**

**int i,c=0,key,f=0;**

**int [] a = {10,20,30,40,50};**

**key = 50;**

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

**{**

**if(a[i]==key)**

**{**

**f=1;**

**break;**

**}**

**c++;**

**}**

**if(f==1)**

**System.*out*.println("Element found"+(c+1));**

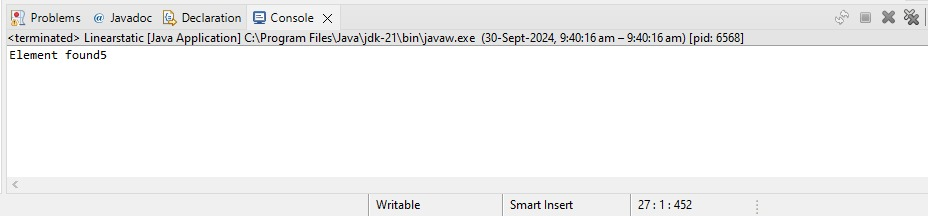
**else**

**System.*out*.println("Element not found");**

**}**

**}**

**Output:**

****

**JAVA implementation for Binary search:**

**package BinarySearch;**

**import java.util.Scanner;**

**public class BinarySearch {**

**int binarySearch(int array[], int element, int low, int high) {**

**while (low <= high) {**

**int mid = low + (high - low) / 2;**

**if (array[mid] == element)**

**return mid;**

**if (element > array[mid])**

**low = mid + 1;**

**else**

**high = mid - 1;**

**}**

**return -1;**

**}**

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

**BinarySearch obj = new BinarySearch();**

**int[] array = { 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160};**

**int n = array.length;**

**Scanner input = new Scanner(System.*in*);**

**System.*out*.println("My Roll No. is:");**

**int element = input.nextInt();**

**input.close();**

**int result = obj.binarySearch(array, element, 0, n - 1);**

**if (result == -1)**

**System.*out*.println("Not found");**

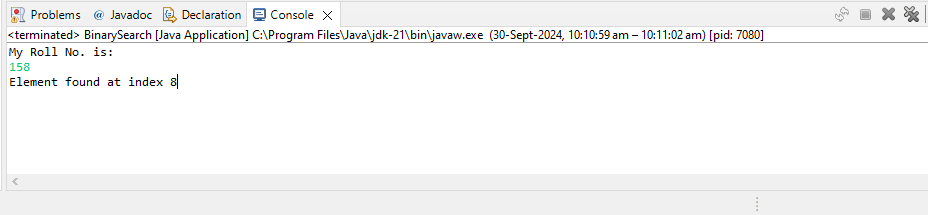
**else**

**System.*out*.println("Element found at index " + result);**

**}**

**}**

**Output:**

****

**JAVA  implementation for Recursive Binary search:**

**package** BinaryRecursiveSearch;

**import** java.util.Arrays;

**import** java.util.Scanner;

**public** **class** RecursiveSearch {

**int** rec\_bin\_search(**int** my\_arr[], **int** left, **int** right, **int** x) {

**if** (right >= left) {

**int** mid = left + (right - left) / 2;

**if** (my\_arr[mid] == x)

**return** mid;

**if** (my\_arr[mid] > x)

**return** rec\_bin\_search(my\_arr, left, mid - 1, x);

**return** rec\_bin\_search(my\_arr, mid + 1, right, x);

       }

**return** -1;

   }

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

       Scanner scanner = **new** Scanner(System.***in***);

       System.***out***.print("Enter the number of elements in the array: ");

**int** n = scanner.nextInt();

**int**[] my\_arr = **new** **int**[n];

       System.***out***.println("Enter the elements of the array:");

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

           my\_arr[i] = scanner.nextInt();

       }

       System.***out***.print("Enter the element to search for: ");

**int** x = scanner.nextInt();

       // Sort the array before binary search

       Arrays.*sort*(my\_arr);

       RecursiveSearch my\_object = **new** RecursiveSearch();

**int** len = my\_arr.length;

**int** result = my\_object.rec\_bin\_search(my\_arr, 0, len - 1, x);

**if** (result == -1)

           System.***out***.println("The element is not present in the array");

**else**

           System.***out***.println("The element has been found at index " + result + " (in the sorted array)");

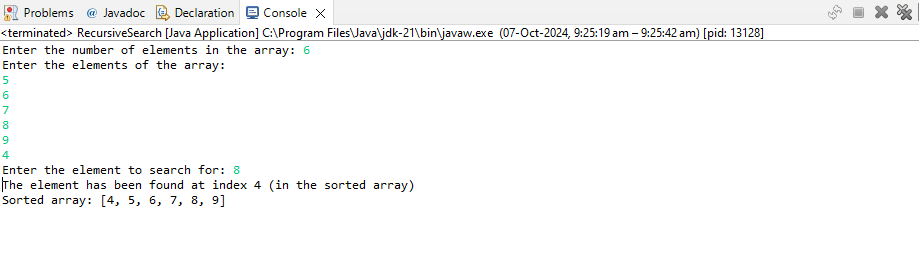
       System.***out***.println("Sorted array: " + Arrays.*toString*(my\_arr));

       scanner.close();

   }

}

**Output:**

****

**Conclusion**:

Linear search is used for an unsorted small list of elements. It has a time complexity of **O(n)**,

Binary Search is used to search through large sorted arrays. It has a time complexity of **O(log n)**

**Practical 2**

**Aim:** To implement sorting  techniques

1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Shell Sort
5. Quick Sort
6. Merge Sort

**Objectives:**

1.   To learn and understand the concept of sorting using different sorting techniques.

**Theory:**

**Pseudo code for Bubble Sort:**

Algorithm bubble (a, n)

Pre: Unsorted array a  of length n.

Post: Sorted array in ascending order of length n

  for i = 1 to (n – 1) do                             // n-1 passes

      for j = 1 to (n- *i* ) do

if ( a[j] > a[j+1] )

1.       temp=a[j]                        //swapping of numbers

2.       a[j]=a[j+1]

3.       a[j+1]=temp

**Pseudo code for Optimized Bubble Sort:**

 Algorithm bubble (a, n)

Pre: Unsorted array a  of length n.

Post: Sorted array in ascending order of length n

1.     for i = 1 to (n – 1) do             // n-1 passes

1.         *test = 0*

2.         for j = 1 to (n– *i* ) do

1.       if ( a[j] > a[j+1] )

1.       temp=a[j]

2.       a[j]=a[j+1]

3.       a[j+1]=temp

4.       test = 1            // exchange happened

3.         *if (test = 0)*    //no exchange - list is now sorted

1.       return

**Pseudo code for Insertion Sort:**

Algorithm insertion (a, n)

Pre: Unsorted list a of length n.

Post: Sorted list a in ascending order of length n

1.  for i = 1 to (n -1)  do               // n-1 passes

1.       temp = a[i]                                      //value to be inserted

2.       ptr = i – 1                         //pointer to move downward

3.       while ( temp < a[ptr]   and   ptr >= 0)

1.       a[ptr + 1] = a[ptr]

2.       ptr = ptr - 1

4.       a[ptr +1] = temp

**Pseudo code for Selection  Sort:**

Algorithm selection (a, n)

Pre: Unsorted array a of length n.

Post: Sorted list in ascending order of length n

1.     for i = 0 to (n -2)  do                           // n-1 passes

1.         min\_index=i

2.         for j = (i+1) to (n -1) do

1.       if ( a[min\_index] > a[j] )

1.       min\_index = j

3.         if (min\_index != i) //place smallest element at ith place

1.       temp= a[i]

2.       a[i]=a[min\_index

3.       a[min\_index]=temp

**Pseudocode for Shell Sort:**

Algorithm shell (a, n, inc, n\_inc)

// a – unsorted array, n – size of array, inc – array storing increment values, n\_inc - size of array increments

Pre: Unsorted list of length n.

Post: Sorted list in ascending order of length n

1. for(increment=0;  increment < n\_inc;  increment++)

         //span is the size of increment

         1. span = inc[increment]

         2. for(j = span; j<n ; j++)

         //inserts element a[j] into its proper position within subfile

                     // sorting

                     1. y = a[j]

                     2. for(k = j-span; (k>=0 && y<a[k]); k= k-span)

                                 1. a[k+span] = a[k]

                     3. a[k+span] = y;

**Pseudo code for Quick Sort:**

int partition (a, beg, end)

// Places pivot element piv at its proper position; elements

   before it are less than it & after it are greater than it

1. piv = a[beg]
2. up = end
3. down = beg
4. while (down < up)
   1. .while( (a[down] <= piv) && (down<end) )
      1. down=down + 1
   2. while(a[up]>piv)
      1. .up=up-1
   3. .if (down<up)
      1. swap ( a[down], a[up])
5. .swap(a[beg],  a[up])
6. .return up

**Algorithm sort (a, beg, end)**

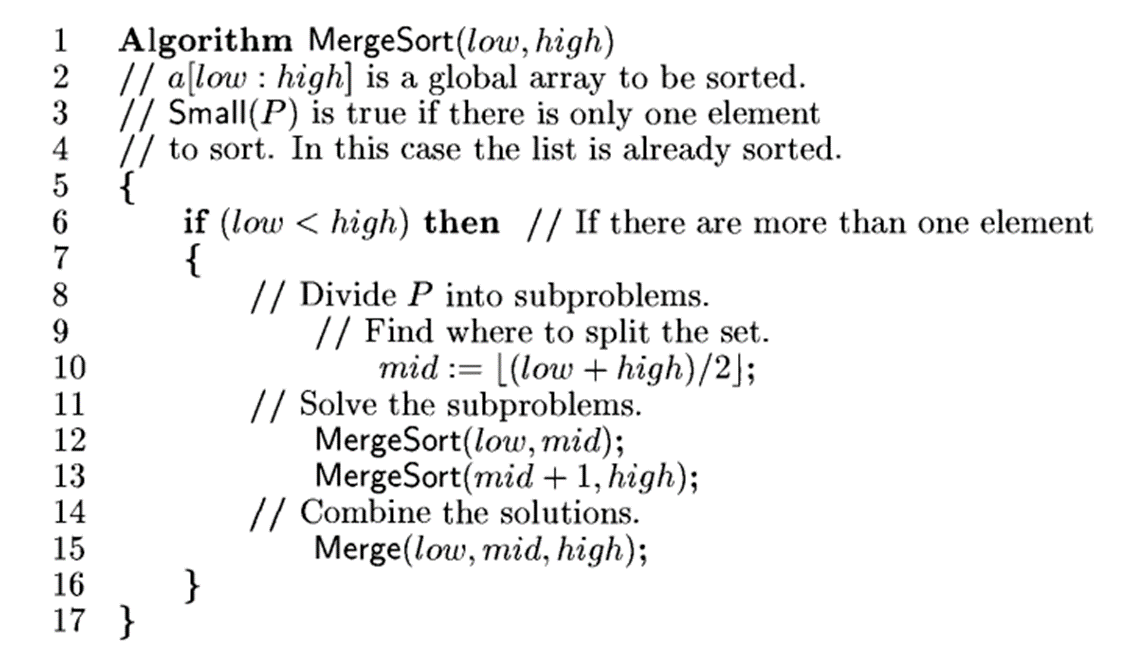
// a - array to be sorted, beg - starting index of array to be sorted, end - ending index of array to be sorted

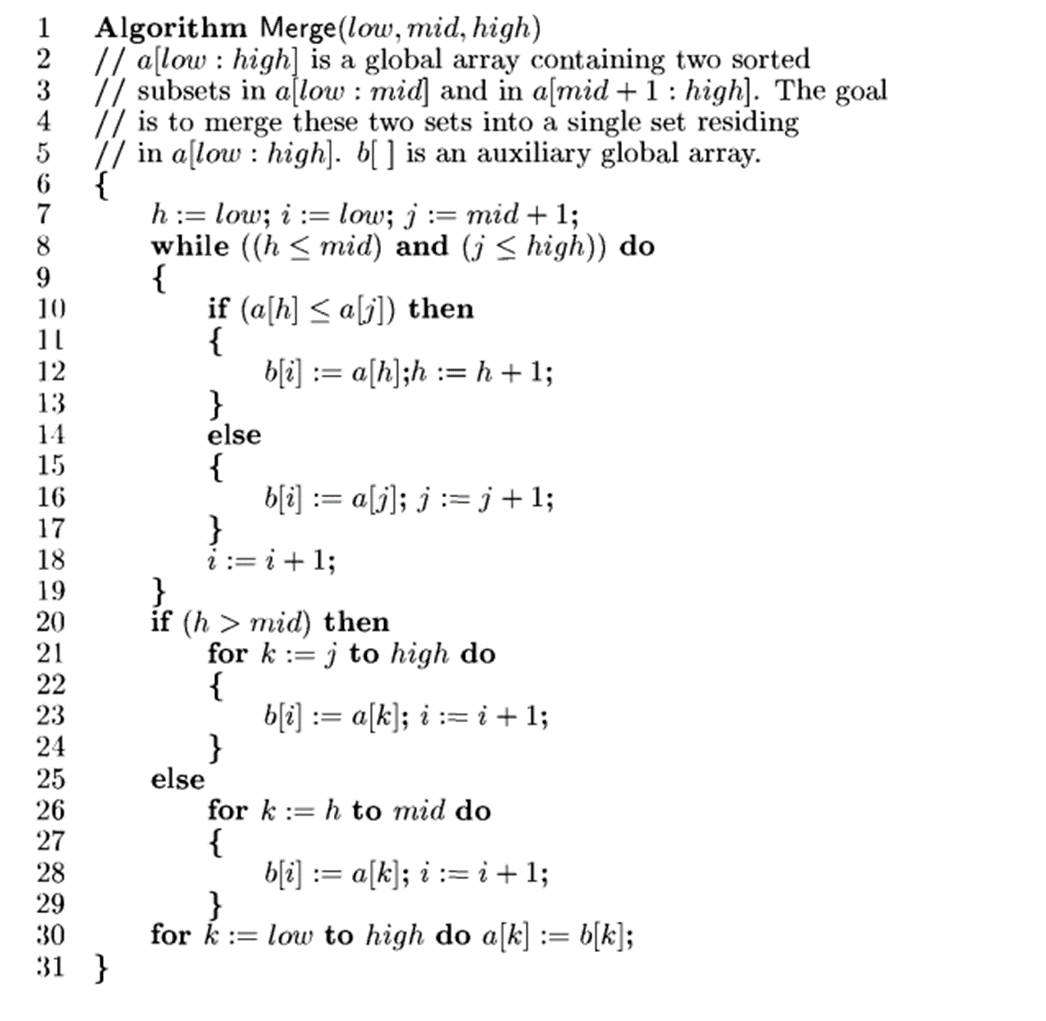
Pre: Unsorted list a of length n.

Post: Sorted list in ascending order of length n

1. if (beg < end)
   1. j = partition(a, beg, end)
   2. sort(a, beg, j-1)
   3. sort (a, j+1, end)
2. Else
   1. return

**Pseudo code for Merge Sort:**





**1)Bubble Sort:**

**Program:**

**package BubbleSort;**

**import java.util.Scanner;**

**public class BubbleSort {**

**static void bubbleSort(int[] a,int n) {**

**int temp = 0,i;**

**for(i=0;i<n;i++) {**

**for(int j = 1;j<(n-i);j++) {**

**if(a[j-1]>a[j]) {**

**temp = a[j-1];**

**a[j-1]=a[j];**

**a[j] = temp;**

**}**

**}**

**}**

**System.*out*.println("Array After Bubble sort");**

**for(i = 0;i<n;i++) {**

**System.*out*.println(a[i]+" ");**

**}**

**}**

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

**int[] a = new int[10];**

**int n,i;**

**Scanner sc = new Scanner(System.*in*);**

**System.*out*.println("ROLL NO: 158");**

**System.*out*.println("Enter Array size");**

**n = sc.nextInt();**

**System.*out*.println("Enter Array element");**

**for(i = 0;i<n;i++) {**

**a[i] = sc.nextInt();**

**}**

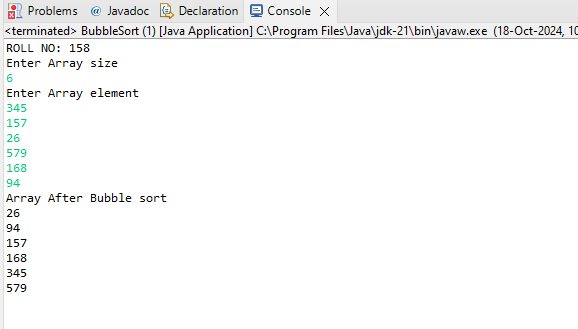
***bubbleSort*(a,n);**

**sc.close();**

**}**

**}**

**Output:**

****

**2)Selection Sort:**

**Code:**

**package sortingAlgos;**

**import java.util.Scanner;**

**public class selectionSort {**

**static void selectionSort(int[] a, int n) {**

**int temp, i;**

**for (i = 0; i < n - 1; i++) {**

**int min\_index = i;**

**for (int j = i + 1; j < n; j++) {**

**if (a[j] < a[min\_index]) {**

**min\_index = j;**

**}**

**}**

**if (min\_index != i) {**

**// Swap elements**

**temp = a[min\_index];**

**a[min\_index] = a[i];**

**a[i] = temp;**

**}**

**}**

**System.*out*.println("Array After Selection Sort:");**

**for (i = 0; i < n; i++) {**

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

**}**

**System.*out*.println();**

**}**

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

**int[] a;**

**int n, i;**

**Scanner sc = new Scanner(System.*in*);**

**System.*out*.println("ROLL NO: 158");**

**System.*out*.print("Enter Array size: ");**

**n = sc.nextInt();**

**a = new int[n];**

**System.*out*.println("Enter Array elements:");**

**for (i = 0; i < n; i++) {**

**a[i] = sc.nextInt();**

**}**

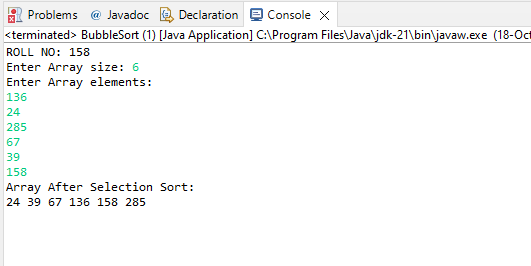
***selectionSort*(a, n);**

**sc.close();**

**}**

**}**

**Output:**

****

**3)Insertion Sort:**

**Code:**

**package sortingAlgos;**

**import java.util.Scanner;**

**public class InsertionSort {**

**public static void InsertionSort(int[] a) {**

**for (int i = 1; i < a.length; i++) {**

**int key = a[i];**

**int j = i - 1;**

**while (j >= 0 && a[j] > key) {**

**a[j + 1] = a[j];**

**j--;**

**}**

**a[j + 1] = key;**

**}**

**System.*out*.println("Array After Insertion Sort:");**

**int i;**

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

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

**}**

**System.*out*.println();**

**}**

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

**int[] a;**

**int n, i;**

**Scanner sc = new Scanner(System.*in*);**

**System.*out*.println("ROLL NO: 158");**

**System.*out*.print("Enter Array size: ");**

**n = sc.nextInt();**

**a = new int[n];**

**System.*out*.println("Enter Array elements:");**

**for (i = 0; i < n; i++) {**

**a[i] = sc.nextInt();**

**}**

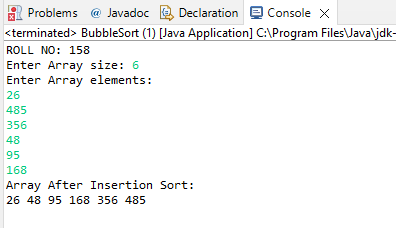
***InsertionSort*(a);**

**sc.close();**

**}**

**}**

**Output:**

****

**4)Shell Sort:**

**Code:**

**package sortingAlgos;**

**import java.util.Scanner;**

**public class shellSort {**

**public static void shellSort(int[] a, int n, int[] inc, int n\_inc) {**

**for (int increment = 0; increment < n\_inc; increment++) {**

**int span = inc[increment];**

**for (int j = span; j < n; j++) {**

**int y = a[j];**

**int k;**

**for (k = j - span; (k >= 0 && y < a[k]); k -= span) {**

**a[k + span] = a[k];**

**}**

**a[k + span] = y;**

**}**

**}**

**}**

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

**int[] a;**

**int n, i;**

**Scanner sc = new Scanner(System.*in*);**

**System.*out*.println("ROLL NO: 158");**

**System.*out*.print("Enter Array size: ");**

**n = sc.nextInt();**

**a = new int[n];**

**System.*out*.println("Enter Array elements:");**

**for (i = 0; i < n; i++) {**

**a[i] = sc.nextInt();**

**}**

**// Define an array of increments for Shell Sort**

**int[] increments = {5, 3, 1};**

**// Call the shellSort method with correct arguments**

***shellSort*(a, a.length, increments, increments.length);**

**// Display the sorted array**

**System.*out*.println("Sorted Array:");**

**for (i = 0; i < n; i++) {**

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

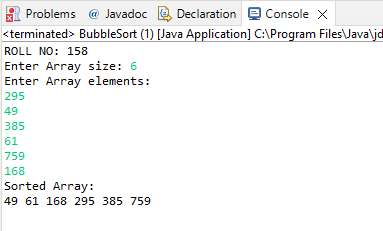
**}**

**sc.close();**

**}**

**}**

**Output:**

****

**5)Quick Sort**

**Code:**

**public static int quickPartition(int[] a, int beg, int end) {**

**int pvt = a[beg];**

**int down = beg;**

**int up = end;**

**int temp;**

**while (down < up) {**

**while (a[down] <= pvt && down < end) {**

**down++;**

**}**

**while (a[up] > pvt) {**

**up--;**

**}**

**if (down < up) {**

**temp = a[down];**

**a[down] = a[up];**

**a[up] = temp;**

**}**

**}**

**temp = a[beg];**

**a[beg] = a[up];**

**a[up] = temp;**

**return up;**

**}**

**public static void sort(int[] a, int beg, int end) {**

**if (beg < end) {**

**int j = *quickPartition*(a, beg, end);**

***sort*(a, beg, j - 1);**

***sort*(a, j + 1, end);**

**}**

**}**

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

**int[] a;**

**int n, i;**

**Scanner sc = new Scanner(System.*in*);**

**System.*out*.println("ROLL NO: 158");**

**System.*out*.print("Enter Array size: ");**

**n = sc.nextInt();**

**a = new int[n];**

**System.*out*.println("Enter Array elements:");**

**for (i = 0; i < n; i++) {**

**a[i] = sc.nextInt();**

**}**

***sort*(a, 0, a.length - 1);**

**System.*out*.println("Sorted array:");**

**for (i = 0; i < n; i++) {**

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

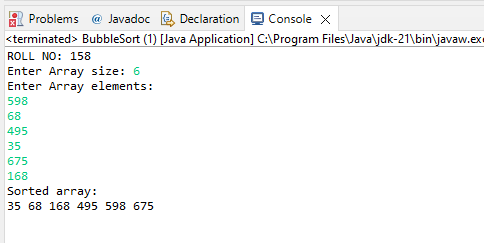
**}**

**sc.close();**

**}**

**}**

**Output:**

****

**6)Merge Sort:**

**Code:**

**package sortingAlgos;**

**import java.util.Scanner;**

**public class mergeSort {**

**private static void merge(int[] arr, int low, int middle, int high) {**

**int n1 = middle - low + 1;**

**int n2 = high - middle;**

**int[] leftArray = new int[n1];**

**int[] rightArray = new int[n2];**

**for (int i = 0; i < n1; i++) {**

**leftArray[i] = arr[low + i];**

**}**

**for (int j = 0; j < n2; j++) {**

**rightArray[j] = arr[middle + 1 + j];**

**}**

**int i = 0, j = 0;**

**int k = low;**

**while (i < n1 && j < n2) {**

**if (leftArray[i] <= rightArray[j]) {**

**arr[k] = leftArray[i];**

**i++;**

**} else {**

**arr[k] = rightArray[j];**

**j++;**

**}**

**k++;**

**}**

**while (i < n1) {**

**arr[k] = leftArray[i];**

**i++;**

**k++;**

**}**

**while (j < n2) {**

**arr[k] = rightArray[j];**

**j++;**

**k++;**

**}**

**}**

**private static void sort(int[] arr, int low, int high) {**

**if (low < high) {**

**int middle = (low + high) / 2;**

***sort*(arr, low, middle);**

***sort*(arr, middle + 1, high);**

***merge*(arr, low, middle, high);**

**}**

**}**

**private static void printArray(int[] arr) {**

**for (int num : arr) {**

**System.*out*.print(num + " ");**

**}**

**System.*out*.println();**

**}**

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

**int[] a;**

**int n, i;**

**Scanner sc = new Scanner(System.*in*);**

**System.*out*.println("ROLL NO:158");**

**System.*out*.println("Enter Array size");**

**n = sc.nextInt();**

**// Initialize the array with the given size**

**a = new int[n];**

**System.*out*.println("Enter Array elements");**

**for(i = 0; i < n; i++) {**

**a[i] = sc.nextInt();**

**}**

***sort*(a, 0, a.length - 1);**

**System.*out*.println("\nSorted Array:");**

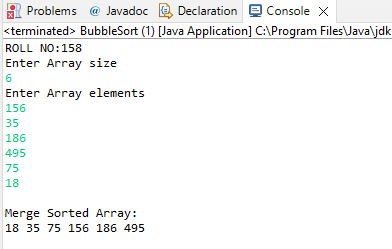
***printArray*(a);**

**sc.close();**

**}**

**}**

**Output:**

****

**Conclusion:**

* Thus we have studied and implemented bubble sort, insertion sort , selection sort,

 Shell and radix sort.

* The Time complexity for Bubble, Insertion and selection sort are

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Time complexity** | | |
|  | Best | Average | worst |
| Bubble sort | O(n) | O(n2) | O(n2) |
| Insertion Sort | O(n) | O(n2) | O(n2) |
| Selection Sort | O(n2) | O(n2) | O(n2) |

* Shell Sort is a comparison based sorting. Time complexity of Shell Sort depends on gap sequence.
* Radix Sort is an efficient non-comparison based sorting algorithm

**Practical 3 - To implement hashing methods and collision resolution techniques**

**Aim: To implement hashing methods and collision resolution techniques**

1.    Modulo Division

2.  Digit Extraction

3.  Fold shift

4.  Fold Boundary

5.  Linear Probe for Collision Resolution

**Objectives:**

1. Learn how to map a large amount of data to a smaller table using a “hash function”
2. Learn to how to solve collision using Linear Probing

**Theory:**

**Modulo – Division Method:**

Address = key  MODULO  list size  + 1

**Digit Extraction:**

Selected digits are extracted from the key and used as the address.

**Fold Shift :** key is divided into number of parts say k1,k2,…..,kn where each parts has the same number of digits except the last part , which can have lesser digits.Add all these parts  and ignore last carry.

For eg. If  Key = 123456789

**123  
 +  456  
         789  
         1368**                      Discard  1  so the address is **368**

**Fold Boundary:** left and right numbers are folded on a fixed boundary between them and the center number. These results in two outside values are being reversed.

         For eg. If Key = 123-456-789

**321  
         456  
 +      987  
           1764**    Discard 1   so the address is : **764**

**Code:**

**package adsa\_hashing;**

**import java.util.\*;**

**import java.util.Scanner;**

**public class New\_hashing {**

**long[] givenkeys = new long[5];**

**long[] list = new long[100];**

**// Method to accept keys from the user**

**public void accept() {**

**make\_null();**

**Scanner sc = new Scanner(System.*in*);**

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

**System.*out*.println("Enter key:");**

**givenkeys[i] = sc.nextLong();**

**}**

**}**

**// Method to initialize the list to 0**

**public void make\_null() {**

**for (int i = 0; i < 100; i++) {**

**list[i] = 0;**

**}**

**}**

**// Method to print the list array**

**public void print() {**

**for (int i = 0; i < 100; i++) {**

**if (list[i] != 0) {**

**System.*out*.println("List[" + i + "]=" + list[i]);**

**}**

**}**

**}**

**// Modular Division Hashing Method**

**public void modular\_division() {**

**System.*out*.println("Enter 4 digit keys");**

**accept();**

**for (int i = 0; i < 4; i++) {**

**int index = (int) ((givenkeys[i] % 100) + 1);**

**if (list[index] == 0) {**

**list[index] = givenkeys[i];**

**} else {**

**int temp = index;**

**while (list[temp] != 0) {**

**temp++;**

**}**

**list[temp] = givenkeys[i];**

**}**

**}**

**print();**

**}**

**// Digit Extraction Hashing Method**

**public void digit\_extraction() {**

**System.*out*.println("Enter 3-digit keys");**

**accept();**

**Scanner sc = new Scanner(System.*in*);**

**int pos1, pos2;**

**System.*out*.println("Enter 2 positions:");**

**pos1 = sc.nextInt();**

**pos2 = sc.nextInt();**

**for (int i = 0; i < 4; i++) {**

**int add = 0;**

**int d1 = (int) (givenkeys[i] / 100);**

**int d2 = (int) ((givenkeys[i] / 10) % 10);**

**int d3 = (int) (givenkeys[i] % 10);**

**// Switch case for position 1**

**switch (pos1) {**

**case 1:**

**add = add + (d1 \* 10);**

**break;**

**case 2:**

**add = add + (d2 \* 10);**

**break;**

**case 3:**

**add = add + (d3 \* 10);**

**break;**

**}**

**// Switch case for position 2**

**switch (pos2) {**

**case 1:**

**add = add + d1;**

**break;**

**case 2:**

**add = add + d2;**

**break;**

**case 3:**

**add = add + d3;**

**break;**

**}**

**if (list[add] == 0) {**

**list[add] = givenkeys[i];**

**} else {**

**int temp = add;**

**while (list[temp] != 0) {**

**temp++;**

**}**

**list[temp] = givenkeys[i];**

**}**

**}**

**print();**

**}**

**public void fold\_shift() {**

**System.*out*.println("Enter 4-digit values");**

**accept();**

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

**int p1 =(int)(givenkeys[i]/100);**

**int p2 =(int)(givenkeys[i]/100);**

**int index = (p1 + p2)%100;**

**if(list[index]==0)**

**list[index]=givenkeys[i];**

**else {**

**int temp = p1 + p2;**

**while(list[temp]!=0)**

**temp++;**

**list[temp] = givenkeys[i];**

**}**

**}**

**print();**

**}**

**public void fold\_boundary() {**

**make\_null();**

**System.*out*.println("Enter 4-Digit Keys");**

**accept();**

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

**int n1 = (int)(givenkeys[i]/100);**

**n1 = (n1%10)\* 10 + (n1/10);**

**int n2 = (int)(givenkeys[i]/100);**

**n2 = (n2%10)\* 10 + (n2/10);**

**int index = (n1 + n2)% 100;**

**if(list[index]==0)**

**list[index] = givenkeys[i];**

**else {**

**int temp = index;**

**while(list[temp]!=0)**

**temp = (temp +1)% 100;**

**list[temp] = givenkeys[i];**

**}**

**}**

**print();**

**}**

**// Main method with menu-driven interface**

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

**System.*out*.println("ROLL NO: 158");**

**Scanner sc = new Scanner(System.*in*);**

**New\_hashing h = new New\_hashing();**

**int ch;**

**do {**

**System.*out*.println("\n1. Modular Division");**

**System.*out*.println("2. Digit Extraction");**

**System.*out*.println("3. Fold Shift");**

**System.*out*.println("4. Fold Boundary");**

**System.*out*.println("5. Exit");**

**System.*out*.println("Enter your choice:");**

**ch = sc.nextInt();**

**switch (ch) {**

**case 1:**

**h.modular\_division();**

**break;**

**case 2:**

**h.digit\_extraction();**

**break;**

**case 3:**

**h.fold\_shift();**

**break;**

**case 4:**

**h.fold\_boundary();**

**break;**

**case 5:**

**System.*out*.println("Exiting program...");**

**sc.close();  // Ensure Scanner is closed before exiting**

**System.*exit*(0);**

**}**

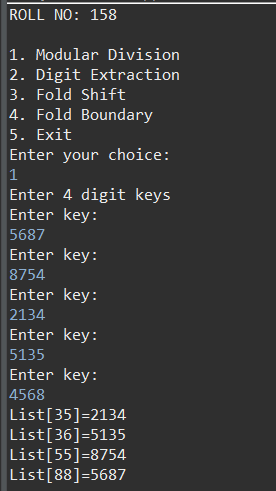
**} while (ch != 3);**

**sc.close();**

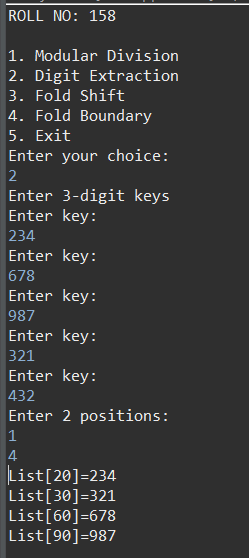
**}**

**}**

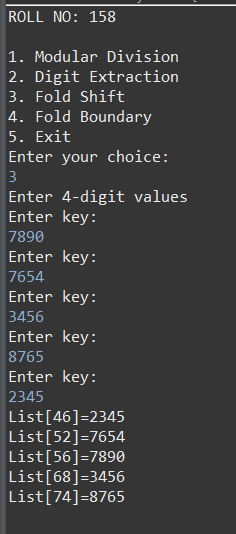
**1)Modulo:**

****

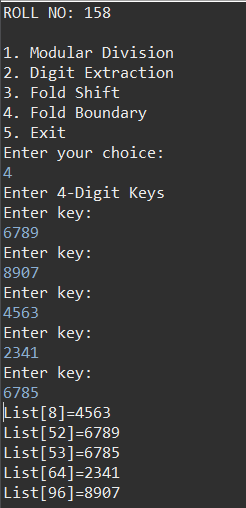
**2) Digit Extraction:**

****

**3) Fold\_Shift:**

****

**4) Fold\_Boundary:**

****

**Conclusion**:  Map a large amount of data to a smaller table using a “hash function” and learn to how to solve collision using Linear Probing

**Practical 4 - To implement different operation on stack using array**

**Aim: To implement different operation on stack using array**

1.    Push()

2.  Pop()

3.  isfull()

4.   isempty()

5.  count()

6. display()

**Objectives:**

1. Learn how to implement different operation on stack using array

**Theory:**

  A stack is an ordered collection of items into which new items may be inserted and items may be deleted at one end called TOP of the stack.

   A stack is a homogeneous collection of items of any one type.

  Data can be added or removed from only the top.

                 Last In, First Out (LIFO)

**Operations perform on stack:**

* **Push  :** Place an item onto the stack. If there is no place for   new item, stack is in overflow state.

**Algorithm:**

**Algorithm :push( S, TOP, X ):** This algorithm insert element x to the top of the stack which is represented by array S containing  N elements with pointer TOP denoting the top most element in the stack.

1.  [ check for stack overflow]

               if  TOP >= N-1

            write[ stack overflow]

             return

2.  [ Increment TOP ]

          TOP = TOP+1

3.  [ Insert element ]

         S[TOP]=X

4.  [finished]

                    return

* **Pop :**Return the item at the top of the stack and then remove it. If pop is called when stack is empty, it is in an underflow state.

**Algorithm :pop( S, TOP ):** This algorithm remove top most  element from top of the stack which is represented by array S containing  N elements with pointer TOP denoting the top most element in the stack.

1.  [ check for stack underflow]

                     if  TOP = -1

            write[ stack underflow on POP]

             return

2.  [ Decrement  TOP Pointer  ]

          TOP = TOP-1

3.  [ return top  element from stack ]

           return(S[TOP+1]

* **isfull:** Tells if the stack is full or not.

**Algorithm :isfull( S, TOP, N ):** This algorithm   check whether stack is full . stack S containing  N elements with pointer TOP denoting the top most element in the stack.

1.  [ check for stack overflow]

               if  TOP > =N-1

               write[ stack full]

             return

* **isEmpty** : Tells if the stack is empty or not.

**Algorithm :isempty l( S, TOP, N ):** This algorithm   check whether stack is full . stack S containing  N elements with pointer TOP denoting the top most element in the stack.

1.  [ check for stack overflow]

                     if  TOP ==-1

            write[ stack empty]

             return

* **Count:** The number of items in the stack.

**Algorithm :count( S, TOP ):** This algorithm count number of elements present in stack which is represented by array S containing  N elements with pointer TOP denoting the top most element in the stack.

1.  [check for  empty stack ]

          if  TOP==-1

            count=0

             return count

2.  For i=0 to top

            Count=count+1

3.  Return count

* **Change():**

**Algorithm :change( S, TOP,X i ):** This algorithm  display elements present in stack from top of the stack S containing  N elements with pointer TOP denoting the top most element in the stack.

1.  [check for  empty stack ]

          if  TOP==-1

            display(“stack is  empty”)

2.  For i=top to 0

          Display ( s[i])

3.  [ finish ]

 return

**Code:**

**package Linear\_DS;**

**import java.util.Scanner;**

**public class Stack\_DS {**

**int TOP = -1;  // Pointer to the top of the stack, initially -1 (empty stack)**

**int N;  // Maximum size of the stack**

**int[] stack;  // Stack array**

**// Constructor to initialize the stack**

**public Stack\_DS(int size) {**

**N = size;**

**stack = new int[N];**

**}**

**// Push operation with overflow check**

**public void push(int X) {**

**if (isFull()) {**

**System.*out*.println("Stack Overflow. Cannot push " + X);**

**return;**

**}**

**TOP = TOP + 1;**

**stack[TOP] = X;**

**System.*out*.println(X + " pushed to stack.");**

**}**

**// Pop operation with underflow check**

**public void pop() {**

**if (isEmpty()) {**

**System.*out*.println("Stack Underflow on POP. The stack is empty.");**

**return;**

**}**

**int poppedElement = stack[TOP];**

**TOP = TOP - 1;**

**System.*out*.println("Popped element: " + poppedElement);**

**}**

**// Check if the stack is full**

**public boolean isFull() {**

**return TOP >= N - 1;**

**}**

**// Check if the stack is empty**

**public boolean isEmpty() {**

**return TOP == -1;**

**}**

**// Count the number of items in the stack**

**public int count() {**

**return TOP + 1; // Because TOP is zero-indexed**

**}**

**// Change (Display) all elements from top to bottom**

**public void change() {**

**if (isEmpty()) {**

**System.*out*.println("Stack is empty.");**

**} else {**

**System.*out*.println("Stack elements from top to bottom:");**

**for (int i = TOP; i >= 0; i--) {**

**System.*out*.println(stack[i]);**

**}**

**}**

**}**

**// Print the current state of the stack**

**public void printStack() {**

**if (isEmpty()) {**

**System.*out*.println("Stack is empty.");**

**} else {**

**System.*out*.println("Stack elements: ");**

**for (int i = 0; i <= TOP; i++) {**

**System.*out*.println(stack[i]);**

**}**

**}**

**}**

**// Main method with infinite loop and switch cases**

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

**Scanner sc = new Scanner(System.*in*);**

**System.*out*.println("Enter the size of the stack:");**

**int size = sc.nextInt();**

**Stack\_DS stackOps = new Stack\_DS(size);**

**while (true) {**

**System.*out*.println("\nChoose an operation:");**

**System.*out*.println("1. Push");**

**System.*out*.println("2. Pop");**

**System.*out*.println("3. Check if Full");**

**System.*out*.println("4. Check if Empty");**

**System.*out*.println("5. Count items in Stack");**

**System.*out*.println("6. Change (Display all elements)");**

**System.*out*.println("7. Print Stack");**

**System.*out*.println("8. Exit");**

**System.*out*.print("Enter your choice: ");**

**int choice = sc.nextInt();**

**switch (choice) {**

**case 1:**

**System.*out*.print("Enter value to push: ");**

**int value = sc.nextInt();**

**stackOps.push(value);**

**break;**

**case 2:**

**stackOps.pop();**

**break;**

**case 3:**

**if (stackOps.isFull()) {**

**System.*out*.println("The stack is full.");**

**} else {**

**System.*out*.println("The stack is not full.");**

**}**

**break;**

**case 4:**

**if (stackOps.isEmpty()) {**

**System.*out*.println("The stack is empty.");**

**} else {**

**System.*out*.println("The stack is not empty.");**

**}**

**break;**

**case 5:**

**System.*out*.println("The number of elements in the stack: " + stackOps.count());**

**break;**

**case 6:**

**stackOps.change();**

**break;**

**case 7:**

**stackOps.printStack();**

**break;**

**case 8:**

**System.*out*.println("Exiting...");**

**sc.close();**

**System.*exit*(0);**

**default:**

**System.*out*.println("Invalid choice. Please try again.");**

**}**

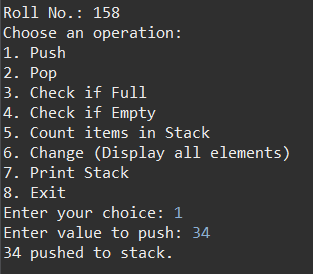
**}**

**}**

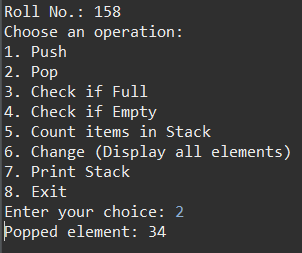
**}**

**Output:**

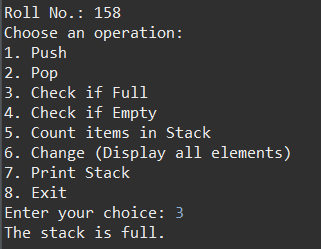
**1)Push**

****

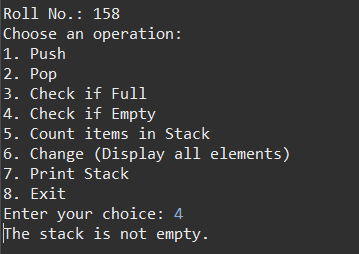
**2) Pop**

****

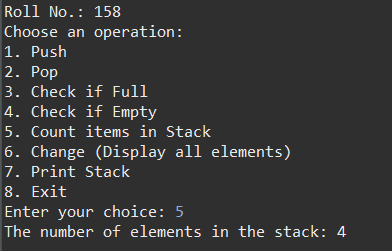
**3) Check if Full**

****

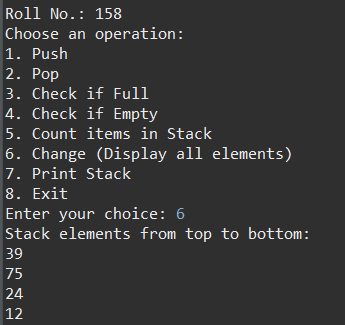
**4) Check if Empty**

****

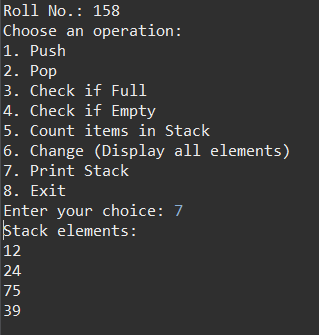
**5) Count items in Stack**

****

**6) Change**

****

**7) Print**

****

**Conclusion**:  Stack is a linear data structure which works on Last In First Out (LIFO) principle.

**Practical 5**

**Aim: Evaluation of postfix expression and balancing of parenthesis**

To implement applications of stack:

1. Evaluation of postfix expressions

2. Balancing of parenthesis

**Objectives:**

1. Learn how to apply stack logic to evaluate postfix expression and checking parenthesis are balanced in given expression

**Theory:** Stack using Array Implementation

**1. Postfix Expressions Evaluation**

The expression of the form “a b operator” (ab+) i.e., when a pair of operands is followed by an operator. Iterate the expression from left to right and keep on storing the operands into a stack. Once an operator is received, pop the two topmost elements and evaluate them and push the result in the stack again.

1. Create an empty stack that will contain operands.

2. Take one by one token from the left to right.

1. If a token is an operand, push it onto the stack.

1. If token is an operator op

2. Pop the top item from the stack as operand2.

3. Pop again the top item from the stack as operand1.

4. Perform operation  operand1  op  operand2.

5. Push the result back to stack.

3. When all tokens in input expression are processed stack should contain a single item, which is

the value of expression

**2. Balancing Parenthesis**

The idea is to put all the opening brackets in the stack. Whenever you hit a closing bracket, search if the top of the stack is the opening bracket of the same nature. If this holds then pop the stack and continue the iteration. In the end if the stack is empty, it means all brackets are balanced or well-formed. Otherwise, they are not balanced.

1. Declare a character stack S.

2. Now traverse the expression string exp.

1. If the current character is a starting bracket (‘(‘ or ‘{‘ or ‘[‘) then push it to stack.

2. If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack and if

the popped character is the matching starting bracket then fine else brackets are not

balanced.

3. After complete traversal, if there is some starting bracket left in the stack then “not balanced”.

**Program:**

1. **1. Postfix Expressions Evaluation**

**package Linear\_DS;**

**import java.util.Stack;**

**public class PostFixEvaluator {**

**public static int evaluatePostfix(String expression) {**

**Stack<Integer> stack = new Stack<>();**

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

**char token = expression.charAt(i);**

**if (Character.*isDigit*(token)) {**

**stack.push(token - '0');**

**} else {**

**int operand2 = stack.pop();**

**int operand1 = stack.pop();**

**int result = 0;**

**switch (token) {**

**case '+':**

**result = operand1 + operand2;**

**break;**

**case '-':**

**result = operand1 - operand2;**

**break;**

**case '\*':**

**result = operand1 \* operand2;**

**break;**

**case '/':**

**result = operand1 / operand2;**

**break;**

**}**

**stack.push(result);**

**}**

**}**

**return stack.pop();**

**}**

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

**String expression = "231\*+9-";**

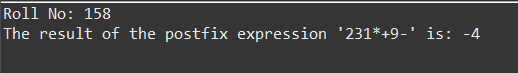
**System.*out*.println("Roll No: 158");**

**System.*out*.println("The result of the postfix expression '" + expression + "' is: " + *evaluatePostfix*(expression));**

**}**

**}**

**Output:**

****

**Program:**

**2. Balancing Parenthesis**

**package Linear\_DS;**

**import java.util.Stack;**

**public class ParenthesisBalancer {**

**public static boolean isBalanced(String expression) {**

**Stack<Character> stack = new Stack<>();**

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

**char current = expression.charAt(i);**

**if (current == '(' || current == '{' || current == '[') {**

**stack.push(current);**

**}**

**else if (current == ')' || current == '}' || current == ']') {**

**if (stack.isEmpty() || !*isMatchingPair*(stack.pop(), current)) {**

**return false;**

**}**

**}**

**}**

**return stack.isEmpty();**

**}**

**private static boolean isMatchingPair(char open, char close) {**

**return (open == '(' && close == ')') ||**

**(open == '{' && close == '}') ||**

**(open == '[' && close == ']');**

**}**

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

**String expression = "{[()]}";**

**System.*out*.println("Roll no. 158");**

**System.*out*.println("The expression '" + expression + "' is " + (*isBalanced*(expression) ? "balanced" : "not balanced"));**

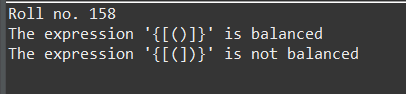
**expression = "{[(])}";**

**System.*out*.println("The expression '" + expression + "' is " + (*isBalanced*(expression) ? "balanced" : "not balanced"));**

**}**

**}**

**Output:**

****

**Conclusion:** Applied stack logic to evaluate postfix expression and checking parenthesis are balanced in given expression.

**Practical 6**

**Aim: To implement linear queue using array**

1.    enqueue

2.      dequeue

3.     count

4.   display

**Objectives:**

1. Learn how to implement different operations on linear queue using array

**Theory:**

**Algorithm : enqueue**

**QINSERT (Q,F,R,N,Y) :** Given F and R , pointers to the front and rear elements of the queue , a queue Q consisting of N elements and y is an element which is inserted by this procedure at the rear of the queue . Initially F and R are set to zero.

1. [ checking for overflow]

         If (R>=N)

                     then write ( “ overflow”)

                             return

2.            [Increment rear pointer ]

   R=R+1

3.            [Insert element]

         Q[R] = Y

4.            [ Is front pointer properly set]

         if F=0

                     Then F= 1

                     Return

**Algorithm : Dequeue**

**QDELETE (Q,F,R)** : Given F and R , pointers to the front and rear elements of the queue , This procedure deletes elements in front  of the  queue .  Y is a temporary variable .

1. [ checking for underflow]

         If ( F==0)

                     then write ( “ underflow”)

                             return

2.            [ Delete element]

         Y= Q[F]

3.            [ Queue empty]

         if F=R

         Then  F=  R = 0

         Else  F= F + 1         ( increment front pointer )

4.            [  Return element ]

                     Return( Y )

**Algorithm : count**

**QCOUNT (Q,F,R) :** Given F and R , pointers to the front and rear elements of the queue , a queue Q consisting of N elements and count is a temporary variable containing no of elements in the queue. This procedure returns 0 if the queue is empty otherwise returns count.

1. [ checking for underflow]

         If ( F==0)

                     then write ( “ Queue is empty ”)

                       count=0

                     return count

2.            else

         for i=front to rear

                     Count =count +1

return count

**Algorithm : display**

**Qdisplay (Q,F,R) :** Given F and R , pointers to the front and rear elements of the queue , a queue Q consisting of N elements and count is a temporary variable containing no of elements in the queue. This procedure returns 0 if the queue is empty otherwise returns count.

2.            [ checking for underflow]

         If ( F==0)

                     then write ( “ Queue is empty ”)

                         return

3.            else

         for i=front to rear

         display ( Q[i])

**Program:**

**package Linear\_DS;**

**import java.util.Scanner;**

**public class Queue\_linear {**

**private int[] Q;**

**private int front;**

**private int rear;**

**private int size;**

**// Constructor to initialize the queue with a given size**

**public Queue\_linear(int n) {**

**size = n;**

**Q = new int[size];**

**front = 0;**

**rear = 0;**

**}**

**// Enqueue operation: Insert element Y at the rear**

**public void enqueue(int y) {**

**if (rear >= size) {**

**System.*out*.println("Overflow: Queue is full.");**

**return;**

**}**

**Q[rear] = y;**

**rear++;**

**// Set front to 1 if this is the first element**

**if (front == 0) {**

**front = 1;**

**}**

**}**

**// Dequeue operation: Remove and return element from the front**

**public int dequeue() {**

**if (front == 0 || front > rear) {**

**System.*out*.println("Underflow: Queue is empty.");**

**return -1;**

**}**

**int y = Q[front - 1];**

**// Adjust front and rear if queue is empty after dequeue**

**if (front == rear) {**

**front = 0;**

**rear = 0;**

**} else {**

**front++;**

**}**

**return y;**

**}**

**// Check if the queue is empty**

**public boolean isEmpty() {**

**return front == 0 || front > rear;**

**}**

**// Count operation: Return the number of elements in the queue**

**public int count() {**

**if (isEmpty()) {**

**return 0;**

**}**

**return rear - front + 1;**

**}**

**// Display operation: Print all elements in the queue from front to rear**

**public void display() {**

**if (isEmpty()) {**

**System.*out*.println("Queue is empty.");**

**return;**

**}**

**System.*out*.print("Queue elements: ");**

**for (int i = front - 1; i < rear; i++) {**

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

**}**

**System.*out*.println();**

**}**

**// Main method to handle user interaction with queue operations**

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

**Scanner sc = new Scanner(System.*in*);**

**System.*out*.println("Enter the size of the queue:");**

**int size = sc.nextInt();**

**Queue\_linear queue = new Queue\_linear(size);**

**while (true) {**

**System.*out*.println("\nChoose an operation:");**

**System.*out*.println("1. Enqueue");**

**System.*out*.println("2. Dequeue");**

**System.*out*.println("3. Check if Queue is Empty");**

**System.*out*.println("4. Count items in Queue");**

**System.*out*.println("5. Display Queue");**

**System.*out*.println("6. Exit");**

**System.*out*.println("Roll No: 158");**

**System.*out*.print("Enter your choice: ");**

**int choice = sc.nextInt();**

**switch (choice) {**

**case 1:**

**System.*out*.print("Enter value to enqueue: ");**

**int value = sc.nextInt();**

**queue.enqueue(value);**

**break;**

**case 2:**

**int dequeuedValue = queue.dequeue();**

**if (dequeuedValue != -1) {**

**System.*out*.println("Dequeued element: " + dequeuedValue);**

**}**

**break;**

**case 3:**

**if (queue.isEmpty()) {**

**System.*out*.println("The queue is empty.");**

**} else {**

**System.*out*.println("The queue is not empty.");**

**}**

**break;**

**case 4:**

**System.*out*.println("The number of elements in the queue: " + queue.count());**

**break;**

**case 5:**

**queue.display();**

**break;**

**case 6:**

**System.*out*.println("Exiting...");**

**sc.close();**

**System.*exit*(0);**

**default:**

**System.*out*.println("Invalid choice. Please try again.");**

**}**

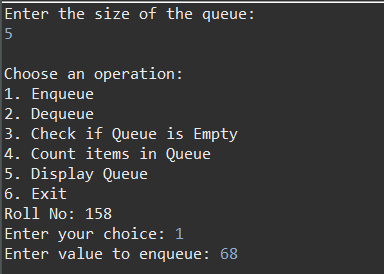
**}**

**}**

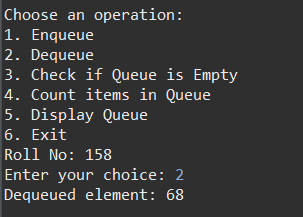
**}**

**Output:**

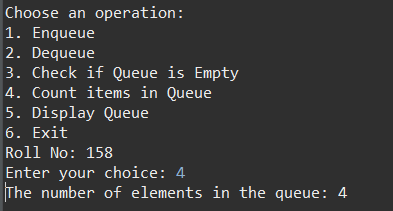
**1.    Enqueue:**

****

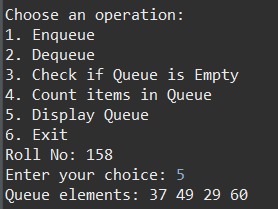
**2.      Dequeue:**

****

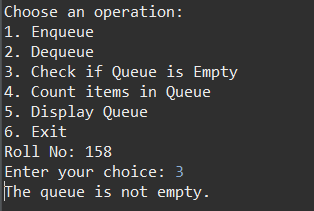
**3.     Count:**

****

**4.   Display**

****

**5.  If empty**

****

**Conclusion**:  LINEAR queue is a linear data structure which works on First In First Out principle.