**EXPERIMENT 1A**

**AIM:**

To Solve problems by using sequential search

**ALGORITHM:**

* Step 1: Traverse the array
* Step 2: Match the key element with array element
* Step 3: If key element is found, return the index position of the array element
* Step 4: If key element is not found, return -1

**PROGRAM:**

class GFG {

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;

}

public static void main(String args[])

{

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

int x = 10;

int result = search(arr, x);

if (result == -1)

System.out.print(

"Element is not present in array");

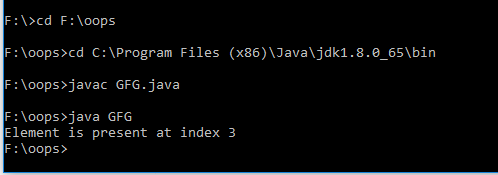
else

System.out.print("Element is present" + " at index " + result);

}

}

**OUTPUT:**



**RESULT:**

**Thus the program for sequential search was executed successfully**

**1.B.**

**AIM:**

**To Solve problems by using binary search**

**ALGORITHM**:

1. **Calculate the mid element of the collection.**
2. **Compare the search value X with the mid element.**
3. **If X = middle element, then we return the mid index position for the key found.**
4. **Else If X > mid element, then the key lies in the right half of the collection. Thus repeat steps 1 to 3 on the lower (right) half of the collection.**
5. **Else X < mid element, then the key is in the upper half of the collection. Hence you need to repeat the binary search in the upper half**.
6. **Print the index of the search value present in the collection**

**PROGRAM:**

**class BinarySearch {**

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

**{**

**if (r >= l) {**

**int mid = l + (r - l) / 2;**

**if (arr[mid] == x)**

**return mid;**

**if (arr[mid] > x)**

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

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

**}**

**return -1;**

**}**

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

**{**

**BinarySearch ob = new BinarySearch();**

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

**int n = arr.length;**

**int x = 3;**

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

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

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

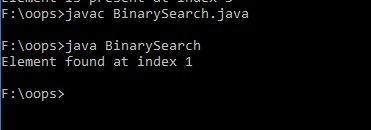
**else**

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

**}**

**}**

**OUTPUT:**



**RESULT:**

**Thus the program for Binary search was executed successfully**

**1.C: AIM:**

**To Solve problems by using quadratic sorting algorithms (selection)**

**ALGORITHM:**

1. *Initialize minimum value(min\_idx) to location 0*
2. *Traverse the array to find the minimum element in the array*
3. *While traversing if any element smaller than****min\_idx****is found then swap both the values.*
4. *Then, increment min\_idx to point to next element*
5. *Repeat until array is sorted*

**PROGRAM:**

**// Java program for implementation of Selection Sort**

**class SelectionSort**

**{**

**void sort(int arr[])**

**{**

**int n = arr.length;**

**// One by one move boundary of unsorted subarray**

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

**{**

**// Find the minimum element in unsorted array**

**int min\_idx = i;**

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

**if (arr[j] < arr[min\_idx])**

**min\_idx = j;**

**// Swap the found minimum element with the first**

**// element**

**int temp = arr[min\_idx];**

**arr[min\_idx] = arr[i];**

**arr[i] = temp;**

**}**

**}**

**// Prints the array**

**void printArray(int arr[])**

**{**

**int n = arr.length;**

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

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

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

**}**

**// Driver code to test above**

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

**{**

**SelectionSort ob = new SelectionSort();**

**int arr[] = {64,25,12,22,11};**

**ob.sort(arr);**

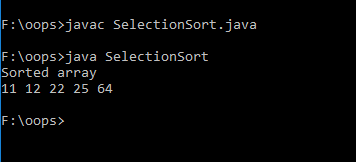
**System.out.println("Sorted array");**

**ob.printArray(arr);**

**}**

**}**

**OUTPUT:**



**RESULT:**

**Thus the Program for quadratic sorting algorithms (selection) was executed successfully**

**1.D. AIM:**

**To Solve problems by using quadratic sorting algorithms (insertion)**

**ALGORITHM:**

1. **To sort an array of size N in ascending order:**
2. **Iterate from arr[1] to arr[N] over the array.**
3. **Compare the current element (key) to its predecessor.**
4. **If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.**

**PROGRAM:**

**// Java program for implementation of Insertion Sort**

**class InsertionSort {**

**/\*Function to sort array using insertion sort\*/**

**void sort(int arr[])**

**{**

**int n = arr.length;**

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

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

**int j = i - 1;**

**/\* Move elements of arr[0..i-1], that are**

**greater than key, to one position ahead**

**of their current position \*/**

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

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

**j = j - 1;**

**}**

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

**}**

**}**

**/\* A utility function to print array of size n\*/**

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

**{**

**int n = arr.length;**

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

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

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

**}**

**// Driver method**

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

**{**

**int arr[] = { 12, 11, 13, 5, 6 };**

**InsertionSort ob = new InsertionSort();**

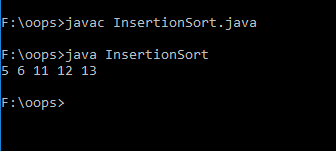
**ob.sort(arr);**

**printArray(arr);**

**}**

**}**

**OUTPUT:**



**RESULT:**

**Thus the Program for quadratic sorting algorithms (insertion) was executed successfully**