**SEARCHING AND SORTING :**

**LINEAR SEARCH :**

package day17;

import java.util.Arrays;

import java.util.\*;

public class task1 {

public static void main(String args[])

{

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

int arr[] = Arrays.*stream*(s.nextLine().split(" ")).mapToInt(Integer::*parseInt*).toArray();

int tar = s.nextInt();

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

{

if(arr[i]==tar)

{

System.***out***.println("value found at index :"+i+"value :"+tar);

}

System.***out***.println("value is not present in array ");

}

}

}

BINARY SEARCH :

package day17;

import java.util.Arrays;

import java.util.\*;

public class task2 {

public static int binarySearch(int arr[],int tar)

{

int left=0 , right=arr.length-1;

while(left<=right)

{

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

if(arr[mid]==tar)

{

System.***out***.println("target found..");

return mid;

}

if(arr[mid]<tar)

{

left=mid+1;

}

else

{

right=mid-1;

}

}

return -1;

}

public static void main(String args[])

{

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

int arr[] = Arrays.*stream*(s.nextLine().split(" ")).mapToInt(Integer::*parseInt*).toArray();

int tar = s.nextInt();

int result=*binarySearch*(arr,tar);

System.***out***.println("ffh1");

if(result!=-1)

{

System.***out***.println("target index :"+result);

}

}

}

BINARY SEARCH USING STRING ARRAY

package day17;

import java.util.Arrays;

import java.util.\*;

public class task3 {

public static int binarySearch(String arr[], String tar) {

int left = 0, right = arr.length - 1;

while (left <= right) {

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

int cmp = arr[mid].compareToIgnoreCase(tar);

if (cmp == 0) {

System.***out***.println("Target found.");

return mid;

}

if (cmp < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

public static void main(String args[])

{

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

String arr[] = Arrays.*stream*(s.nextLine().split(" ")).toArray(String[]::new);

String tar = s.next();

int result=*binarySearch*(arr,tar);

System.***out***.println("ffh1");

System.***out***.println("target index :"+result);

if(result!=-1)

{

}

Square root using binary search .

package day17;

import java.util.\*;

public class task4 {

public static int squrt(int n)

{

int left=1,right=n,ans=0;

if(n==0 || n==1)

{

return n;

}

while(left<=right)

{

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

if(mid==n/mid)

{

return mid;

}

if(mid<n/mid)

{

left=mid+1;

ans=mid;

}

else {

right=mid-1;

}

}

return ans;

}

public static void main(String args[])

{

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

int n = s.nextInt();

System.***out***.println("square root of :"+n+" is :"+*squrt*(n));

System.***out***.println(25/2);

}

}

**INTER COLLETIONAL SEARCH**

This algorithm is useful when the data is uniform distributed since it estimates the likely pos of target based on the value .

This search algorithm is designed for numeric and uniformly distributed data .

To apply it to words the we first need a function that map each word to the numeric value in a way that prefer their sorted data

One common approach into a num using polynomial accumulation similar to how many hash code are computed ..

Since the numeric value of num might not a uniformly distributed so the performance gains you expert from inter colletion search might not be a pronounced as numeric data .

Pos = low + (key – arr[low] \* high-low) /arr[high]-arr[low]

Arr={10,30,40,50}

Key=35

Low=0;

High=4;

package day17;

import java.util.Arrays;

import java.util.\*;

public class task5 {

public static int intercolletional(int arr[] , int key)

{

int low = 0,high = arr.length-1;

while(low<=high && key>=arr[low] && key <=arr[high])

{

if(low==high)

{

if(arr[low]==key)

{

return low;

}

else {

return -1;

}

}

int pos = low + ((key-arr[low])\*(high-low))/ (arr[high]-arr[low]);

if(arr[pos]==key)

{

return pos;

}

if(arr[pos]<key)

{

low=pos+1;

}

else {

high=pos-1;

}

}

return -1;

}

public static void main(String args[]) {

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

int arr[] = Arrays.*stream*(s.nextLine().split(" ")).mapToInt(Integer::*parseInt*).toArray();

int key=s.nextInt();

int rs = *intercolletional*(arr,key);

if(rs!=-1)

{

System.***out***.println("element found "+rs);

}

else{System.***out***.println("not found");

}}

}

Inter collational search for string array :

package day17;

import java.util.\*;

public class task6 {

public static long StringToNum(String s) {

long num = 0;

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

num = num \* 31 + (int) s.charAt(i);

}

return num;

}

public static int intercolle(String arr[], String s) {

long key = *StringToNum*(s);

int low = 0, high = arr.length - 1;

while (low <= high) {

long lowValue = *StringToNum*(arr[low]);

long highValue = *StringToNum*(arr[high]);

if (key < lowValue || key > highValue) {

return -1;

}

if (low == high) {

return arr[low].equals(s) ? low : -1;

}

int pos = low + (int) ((key - lowValue) \* (high - low) / (highValue - lowValue));

if (pos < low || pos > high) {

return -1;

}

int compare = arr[pos].compareTo(s);

if (compare == 0) {

return pos;

} else if (compare < 0) {

low = pos + 1;

} else {

high = pos - 1;

}

}

return -1;

}

public static void main(String args[]) {

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

String arr[] = ss.nextLine().split(" ");

System.***out***.println("Enter the string to search:");

String key = ss.next();

int index = *intercolle*(arr, key);

if (index != -1) {

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

} else {

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

}

}

}

Jump search algo for sorted array that works by check fewer word then linear by jumping the head of fixed no of step .

This algorithm is efficient linear search but less efficient than binary search for large data set .

package day17;

import java.util.Arrays;

import java.util.\*;

public class task7 {

public static int jumpSearch(int arr[],int key)

{

int n= arr.length;

int step = (int)Math.*floor*(Math.*sqrt*(n));

int prev = 0 ;

while (arr[Math.*min*(step, n) - 1] < key) {

prev = step;

step += (int) Math.*floor*(Math.*sqrt*(n));

// If we've reached the end of the array,

//the element is not present.

if (prev >= n) {

return -1;

}

}

// Perform linear search within the identified block

while (arr[prev] < key) {

prev++;

// If we reach the end of the block or array

//without finding the key

if (prev == Math.*min*(step, n)) {

return -1;

}

}

// Check if the element at index 'prev' is

//the target element.

if (arr[prev] == key) {

return prev;

}

return -1;

}

public static void main(String args[])

{

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

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

int tar = 4;

int result=*jumpSearch*(arr,tar);

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

**Sorting :**

Sorting algorithms helps to arrange and sort data values in a container data type .

**Bubble sort :**

package day17;

import java.util.Arrays;

public class task8 {

public static void bubbleSort(int arr[])

{

int n=arr.length;

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

{

boolean swap = false;

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

{

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

{

int temp = arr[j];

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

arr[j+1]=temp;

swap=true;

}

}

if(!swap) break;

}

System.***out***.println(Arrays.*toString*(arr));

}

public static void main(String args[])

{

int arr[]= {12,55,2,554,78,1,8};

*bubbleSort*(arr);

}

Output :

[1, 2, 8, 12, 55, 78, 554]

**SELECTION SORT :**

package day17;

import java.util.Arrays;

public class task9 {

public static void selectionSort(int arr[])

{

int n= arr.length;

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

{

int min\_index = i;

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

{

if(arr[j]<arr[min\_index])

{

min\_index=j;

}

}

int temp = arr[min\_index];

arr[min\_index]=arr[i];

arr[i]=temp;

}

System.***out***.println(Arrays.*toString*(arr));

}

public static void main(String args[])

{

int arr[]= {12,5,3,27,32,8,3,1};

*selectionSort*(arr);

}

}

[1, 3, 3, 5, 8, 12, 27, 32]

Mergesort

package day17;

import java.util.Arrays;

public class task10 {

public static void mergeSort(int[] arr) {

if (arr.length > 1) {

int mid = arr.length / 2;

int[] left = new int[mid];

int[] right = new int[arr.length - mid];

System.*arraycopy*(arr, 0, left, 0, mid);

System.*arraycopy*(arr, mid, right, 0, arr.length - mid);

*mergeSort*(left);

*mergeSort*(right);

*merge*(arr, left, right);

}

}

public static void merge(int[] arr, int[] left, int[] right) {

int i = 0, j = 0, k = 0;

while (i < left.length && j < right.length) {

if (left[i] <= right[j]) {

arr[k++] = left[i++];

} else {

arr[k++] = right[j++];

}

}

while (i < left.length) {

arr[k++] = left[i++];

}

while (j < right.length) {

arr[k++] = right[j++];

}

}

public static void main(String args[]) {

int[] arr = {12, 23, 4, 54, 56, 75, 645};

*mergeSort*(arr);

System.***out***.println(Arrays.*toString*(arr));

}

}

[4, 12, 23, 54, 56, 75, 645]