**חיפוש בינארי Binary Search**

הפונקציה מקבלת מערך ממוין (מהקטן לגדול) ומפתח, ומחזירה אמת/שקר לגבי המצאות המפתח במערך. במקרה שהמפתח נמצא במערך, הפונקציה מחזירה את המיקום במערך.

public static int binarySearch(int[] arr, int key) {

int ans = -1;

boolean found = false;

int low = 0;

int high = arr.length - 1;

while (!found && low <= high) {

int middle = (low + high)/2;

if (arr[middle] == key){

found = true;

ans = middle;

}

else if (key < arr[middle])

high = middle - 1;

else

low = middle + 1;

}//while

return ans;

}//fuction binarySearch

**Linear Search חיפוש לינארי**

הפונקציה מקבלת מערך (לא צריך להיות ממוין) ומפתח, ומחזירה אמת/שקר לגבי המצאות המפתח במערך.

public static int linearSearch(int[] arr, int key) {

int ans = -1; // default (not found) value

for (int i=0; i<arr.length && ans == -1; i=i+1) {

if (key==arr[i])

ans = i;

}

return ans;

}

**Number of digits in a number מספר הספרות במספר**

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

int m;

System.out.print("Enter an integer number:");

m = sc.nextInt();

int count = 1;

while (m >= 10) {

count = count+1;

m = m / 10;

}

System.out.println("The given number has " + count + " digits");

}

**Greatest Common Divisor (GCD) המחלק המשותף הגדול ביותר**

public class Gcd

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

int m,n;

System.out.print("Enter first number:");

m = sc.nextInt();

System.out.print("Enter second number:");

n = sc.nextInt();

int r = m % n;

while (r != 0)

{

m = n;

n = r;

r = m % n;

}

System.out.println("The GCD is " + n);

}

}

**Max Use of a Lecture Roomשימוש מקסימלי של חדר הרצאות**

public static void main(String[] args) {

int[][] segArr = {{8,10}, {9, 12}, {11,13}, {14,18}, {12,14}, {15,19}, {16,18}};

System.out.println("Maximum use in hours is:" + maxUse(segArr));

}

public static int maxUse(int[][] segArr) {

// memo is an array for keeping already calculated results

// memo[k] = maximal use for lectures 0, 1, 2, ..., k

// initialization: memo[k] = 0 for all k

int[] memo = new int[segArr.length];

// sort time slots by end time

sortByEndPoints(segArr);

return maxUse(segArr,segArr.length-1,memo);

}

public static int maxUse(int[][] segArr, int i, int[] memo) {

int result = 0;

if (i >= 0) {

//there this one lecture or more

if (memo[i]==0) {

int len=segArr[i][1]-segArr[i][0]; // duration of lecture i

int prev=findNextSeg(segArr, i);

int with=maxUse(segArr,prev,memo)+len;  
// schedule lecture i and calculate the maximal use of all lectures that do not overlap with lecture i and end before i

int without=maxUse(segArr,i-1,memo);  
// do not schedule lecture i, calculate the maximal use of lectures 0, 1, ..., i-1

memo[i]=Math.max(with,without);  
// save the calculated result

}

result=memo[i];

}

return result;

}//maxUse

// find the maximal index of lecture that ends before start time of lecture i

public static int findNextSeg(int[][] segments, int i){

int endPoint = segments[i][0];

while ((i >= 0) && (endPoint < segments[i][1]) )

i = i -1;

return i;

}

//Sorts the time slots according to the end time, uses Insertion Sort algorithm

public static void sortByEndPoints (int[][] arr) {

if(arr == null)

throw new NullPointerException();

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

// insert arr[i] into a sorted sequence arr[0],..., arr[i-1]

int[] timeSlot = arr[i];

int j = i;

// shift larger values to the right

while (j > 0 && arr[j-1][1] > timeSlot[1]){

arr[j] = arr[j-1];

j--;

}

// insert key to its right j

arr[j] = timeSlot;

}

}

**Merge Sort מיון מיזוג**

public static void mergeSort(int[] arr, int low, int high){

if (low < high){

int mid = (low + high)/2;

mergeSort(arr, low, mid); // recursively sort

mergeSort(arr, mid+1, high); // recursively sort

merge(arr,low,mid,high); // merge two sorted subarrays

}

}

public static void merge(int[] arr, int low, int mid, int high){

int n = high-low+1;

int[] temp = new int[n];

int i = low, j = mid+1;

int k = 0;

while(i <= mid && j <= high) {

if (arr[j] < arr[i]) {

temp[k] = arr[j];

j = j+1;

k = k+1;

}

else {

temp[k] = arr[i];

i = i+1;

k = k+1;

}

}

while(i <= mid){

temp[k] = arr[i];

i = i+1;

k = k+1;

}

while(j <= high){

temp[k] = arr[j];

j = j+1;

k = k+1;

}

for (k = 0; k < n; k++)

arr[low + k] = temp[k];

}

}

**Selection Sort מיון בחירה**

public static void selectionSort (int[] arr){

if(arr == null)

throw new NullPointerException();

int minIndex;

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

minIndex = i;

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

if (arr[j] < arr[minIndex])

minIndex = j;

// Swap the values

swap(arr, minIndex, i);

} //for i

}//selectionSort

// Swaps the elements arr[i] and arr[j] in array arr

public static void swap(int [] arr, int i, int j){

int temp;

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

**Insertion Sort מיון הכנסה**

public static void insertionSort (int[] arr) {

if(arr == null)

throw new NullPointerException();

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

// insert arr[i] into a sorted sequence arr[0],..., arr[i-1]

int key = arr[i];

int j = i;

// shift larger values to the right

while (j > 0 && arr[j-1] > key){

arr[j] = arr[j-1];

j--;

}

// insert key to its right j

arr[j] = key;

}

}

**Bubble Sort מיון בועות**

public static void bubbleSort(int[] array) {

boolean isSorted = false;

int tmp;

for (int bbl=0; !isSorted && bbl<array.length-1; bbl=bbl+1) {

isSorted = true;

for (int index=0; index<array.length-1-bbl; index=index+1) {

if (array[index] > array[index+1]) {

tmp = array[index];

array[index] = array[index+1];

array[index+1] = tmp;

isSorted = false;

}

}

}

}

**Print Prime Numbersהדפסת המספר הראשוניים**

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

System.out.print("Enter an integer number:");

int n = sc.nextInt();

int m = 2; // first prime number

while (m <= n) {

boolean isPrime = true;

int i = 2;

while (i < m && isPrime) {

if (m % i == 0)

isPrime = false;

i = i + 1;

} // while

if(isPrime)

System.out.println(m);

m = m + 1;

} // while m <= n

}//main

**IsSubString חיפוש תת-מחרוזת במחרוזת**

public static boolean isSubstring(String str,String sub){

boolean isFound = false;

int lastInd = str.length()- sub.length();

for (int i=0; i<=lastInd && !isFound; i=i+1) {

String strSub = str.substring(i, i+sub.length());

if (strSub.equals(sub)) {

isFound = true;

}

}

return isFound;

}

**\*Recursionרקורסיה \***

**Factorial**

public static int fact(int n) {

int ans;

if(n==0)

ans = 1;

else

ans = n \* fact(n-1);

return ans;

}

**Fibonacci**

public static int fib(int n) {

int ans;

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

ans = 1;

else

ans = fib(n-1)+fib(n-2);

return ans;

}

**Tail Recursion: Print in Binary**

public static void printBin(int n){ //wrapper

printBin(n, "");

}

public static void printBin(int n, String acc){

if(n < 2)

System.out.println(n+acc);

else printBin(n/2, n%2+acc); // prints the last digit

}

**Reverse String**

public static String reverse(String s) {

String answer;

if (s.length()==0)

answer = s;

else

answer = reverse(s.substring(1)) + s.charAt(0);

return answer;

}

**Hanoi Towers**

public static void hanoi(int n, char source, char destination, char extra) {

i

hanoi(n-1,source,extra,destination);

System.out.println("Move disk from "+source+ " to "+destination);

hanoi(n-1,extra,destination,source);

}

**Substrings of Length K from String**

public static void substrs(String s,int k) { //Wrapper

substrs(s, k, "");

}

public static void substrs(String s, int k, String acc) {

if (s.length() == k)

System.out.println(acc+s);

else if(k==0)

System.out.println(acc);

else {

substrs(s.substring(1), k-1, acc + s.charAt(0));

substrs(s.substring(1), k, acc);

}

}

**Palindrome Text**

public static boolean isPalindrome(String pal) {

boolean isPal = false;

int length = pal.length();

if (length == 0 || length == 1)

// can be “if (length <= 1)” instead

isPal = true;

else {

isPal = (pal.charAt(0)==pal.charAt)length-1)) && (isPalindrome(pal.substring(1,length-1)));

}

return isPal;

}

**Subset Sum**

בהינתן מערך של משקולות ומשקל נוסף, נרצה לבדוק האם ניתן להרכיב מהמשקולות משקל השווה למשקל הנתון.

public static boolean calcWeights(int[] weights, int sum) { //wrapper

return calcWeights(weights, 0, sum);

}

public static boolean calcWeights(int[] weights, int i, int sum) {

boolean res = false;

if (sum == 0)

res = true;

else if (i >= weights.length)

res = false;

else

res = (calcWeights(weights,i+1,sum-weights[i]) || calcWeights(weights, i+1, sum) );

return res;

}

**Maze**

public static boolean solve(int[][] grid, int row, int col) {

boolean done = false;

if (valid(grid, row, col)) {

grid[row][col] = 3; // mark visited

if ((row == grid.length-1) && (col == grid[0].length-1))

done = true; // maze is solved

else {

done=( solve(grid, row + 1, col) || // try down

solve(grid, row, col+1) || // try right

solve(grid, row-1, col) || // try up

solve(grid, row, col-1) ); // try left

}

if (done)

grid[row][col] = 7; // mark as part of the path

}

return done;

}

**Slope מטריצה משופעת**

מטריצה תקרא משופעת אם כל איברי האלכסון הראשי שווים לאפס, כל האיברים שנמצאים מתחת לאלכסון הראשי הם שליליים וכל האיברים שנמצאים מעל לאלכסון הראשי הם חיוביים.

public static boolean slope(int[][] data) {

return slope(data, 0);

}

public static boolean slope(int[][] data, int index) {

boolean isSlope = false;

//end of array – last cell, if it’s 0 then it’s OK

if (index == data.length - 1)

isSlope = (data[index][index] == 0);

else

isSlope = (check(data,index) &&

slope(data, index+1));

return isSlope;

}