Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. Greedy algorithms are used for optimization problems. An optimization problem can be solved using Greedy if the problem has the following property: At every step, we can make a choice that looks best at the moment, and we get the optimal solution of the complete problem.  
If a Greedy Algorithm can solve a problem, then it generally becomes the best method to solve that problem as the Greedy algorithms are in general more efficient than other techniques like Dynamic Programming. But Greedy algorithms cannot always be applied. For example, Fractional Knapsack problem can be solved using Greedy, but [0-1 Knapsack](http://www.geeksforgeeks.org/archives/18430) cannot be solved using Greedy.

Let us consider the [Activity Selection problem](http://en.wikipedia.org/wiki/Activity_selection_problem) as our first example of Greedy algorithms. Following is the problem statement.  
You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

**Example 1 :** Consider the following 3 activities sorted by

by finish time.

start[] = {10, 12, 20};

finish[] = {20, 25, 30};

A person can perform at most **two** activities. The

maximum set of activities that can be executed

is {0, 2} [ These are indexes in start[] and

finish[] ]

The greedy choice is to always pick the next activity whose finish time is least among the remaining activities and the start time is more than or equal to the finish time of previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.

Coding link: http://ide.geeksforgeeks.org/Jqy77M

But the coding is in java.

// The following implementation assumes that the activities

// are already sorted according to their finish time

import java.util.\*;

public class ActivitySelection

{

private static Scanner sc;

// Prints a maximum set of activities that can be done by a single

// person, one at a time.

// n --> Total number of activities

// s[] --> An array that contains start time of all activities

// f[] --> An array that contains finish time of all activities

public static void printMaxActivities(int s[], int f[], int n)

{

int i, j;

System.out.print("Following activities are selected : ");

// The first activity always gets selected

i = 0;

System.out.print(i+" ");

// Consider rest of the activities

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

{

// If this activity has start time greater than or

// equal to the finish time of previously selected

// activity, then select it

if (s[j] >= f[i])

{

System.out.print(j+" ");

i = j;

}

}

}

// driver program to test above function

public static void main(String[] args)

{

int k,l;

int []s = new int[6];

int []f = new int[6];

sc = new Scanner(System.in);

System.out.println("Enter the starting activities");

for(k=0;k<6;k++)

s[k]=sc.nextInt();

System.out.println("Enter the finishing activities");

for(l=0;l<6;l++)

f[l]=sc.nextInt();

int n = s.length;

printMaxActivities(s, f, n);

}

}