**EXPERIMENT N0-11**

**AIM:** **Solve any problems that are based on two different algorithm paradigms.**

**1:MAX MIN USING GREEDY METHOD:**

**PROBLEM STATEMENT:**

You will be given a list of integers, arr and a single integer k You must create an array of length  k from elements of arr such that its unfairness is minimized. Call that array arr’ Unfairness of an array is calculated as

MAX(arr’) – MIN(arr’)

Where:  
- max denotes the largest integer in arr’  
- min denotes the smallest integer in  arr’

**THEORY:**

* we have given arr of size n and integer k
* we have to make possible sub array of k size
* from all the sub array we have to find the max(arr) – min(arr)
* and display the smallest among the max(arr) – min(arr)
* EG: arr = [1,4,7]

Possible sub array :

[1,4] : max=4 and min=1 : 4-1=3

[4,7] : max=7 and min=4 : 7-4=3

[1,7] : max=7 and min=1 : 7-1=6

Therefore min among above is 3

* Sort the array
* result = arr[k-1] - arr[0]
* for i in range(n-k+1):
* if arr[i+k-1] - arr[i] < result:
* result = arr[i+k-1] - arr[i]
* return result

**CODE:**

import math

import os

import random

import re

import sys

def maxMin(k, arr):

arr.sort()

result = arr[k-1] - arr[0]

for i in range(n-k+1):

if arr[i+k-1] - arr[i] < result:

result = arr[i+k-1] - arr[i]

return result

if \_\_name\_\_ == '\_\_main\_\_':

fptr = open(os.environ['OUTPUT\_PATH'], 'w')

n = int(input().strip())

k = int(input().strip())

arr = []

for \_ in range(n):

arr\_item = int(input().strip())

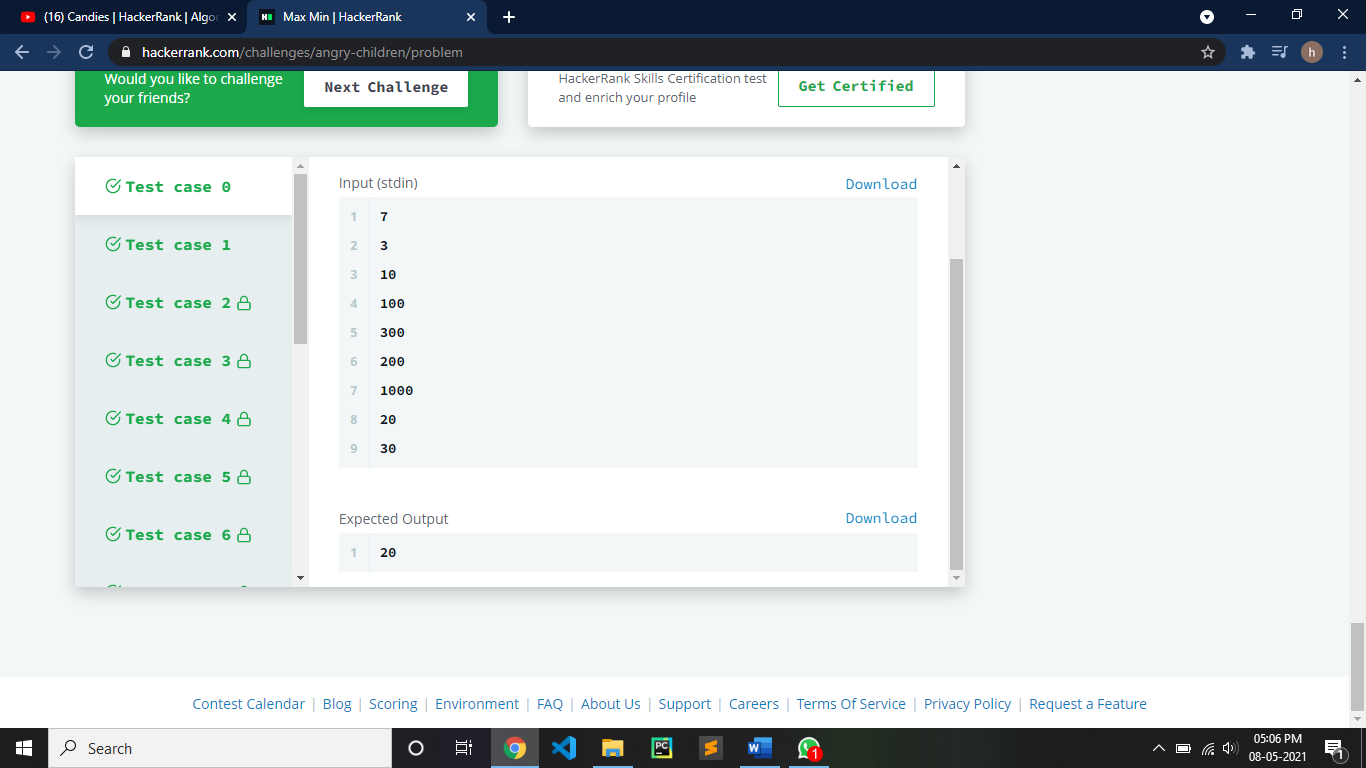
arr.append(arr\_item)

result = maxMin(k, arr)

fptr.write(str(result) + '\n')

fptr.close()

**OUTPUT:**

****

**CONCLUSION:**

* Time complexity is O(nlogn) Because we used quick sort.
* Space complexity is O(1)

**2: THE COIN CHANGE PROBLEM USING DYNAMIC PROGRAMMING**

**PROBLEM STATEMENT:-**

Given an amount and the denominations of coins available, determine how many ways change can be made for amount. There is a limitless supply of each coin type.

**THEORY :-**

* The Coin Change Problem is considered by many to be essential to understanding the paradigm of programming known as Dynamic Programming**.**
* Ex. If we have given 4rs to change and available coin types are [1,2,3]. So, there is 4 ways to change 4rs i.e. {1,1,1,1}, {2,2}, {3,1},{2,1,1}.
* First we will enter the amount to change and number of coin types.
* Then we will enter the values of each coin type.
* After that we will check if the entered amount to change is less than 0, if true we return 0 and if the entered amount to change is equal to 0 will return 1 and if number of coins is less than or equal to 0 will return 0.
* counts = [0] \* (money+1)

counts[0] = 1

* for i in range(0, no\_of\_coins):

sum = 0

for j in range(coins[i],money+1):

counts[j] += counts[j-coins[i]]

* return counts[money]

**SOURCE CODE:**

import math

import os

import random

import re

import sys

money,no\_of\_coins = list(map(int,input().strip().split(' ')))

coins = list(map(int,input().strip().split(' ')))

count = 0

def count\_make\_change(money, coins, no\_of\_coins):

if money < 0:

return 0

if money == 0:

return 1

if no\_of\_coins <= 0:

return 0

return count\_make\_change(money-coins[no\_of\_coins-

1], coins, no\_of\_coins) + count\_make\_change(money,coins,no\_of\_coins-1)

def count\_make\_change\_bottom\_up(money, coins, no\_of\_coins):

counts = [0] \* (money+1)

counts[0] = 1

for i in range(0, no\_of\_coins):

sum = 0

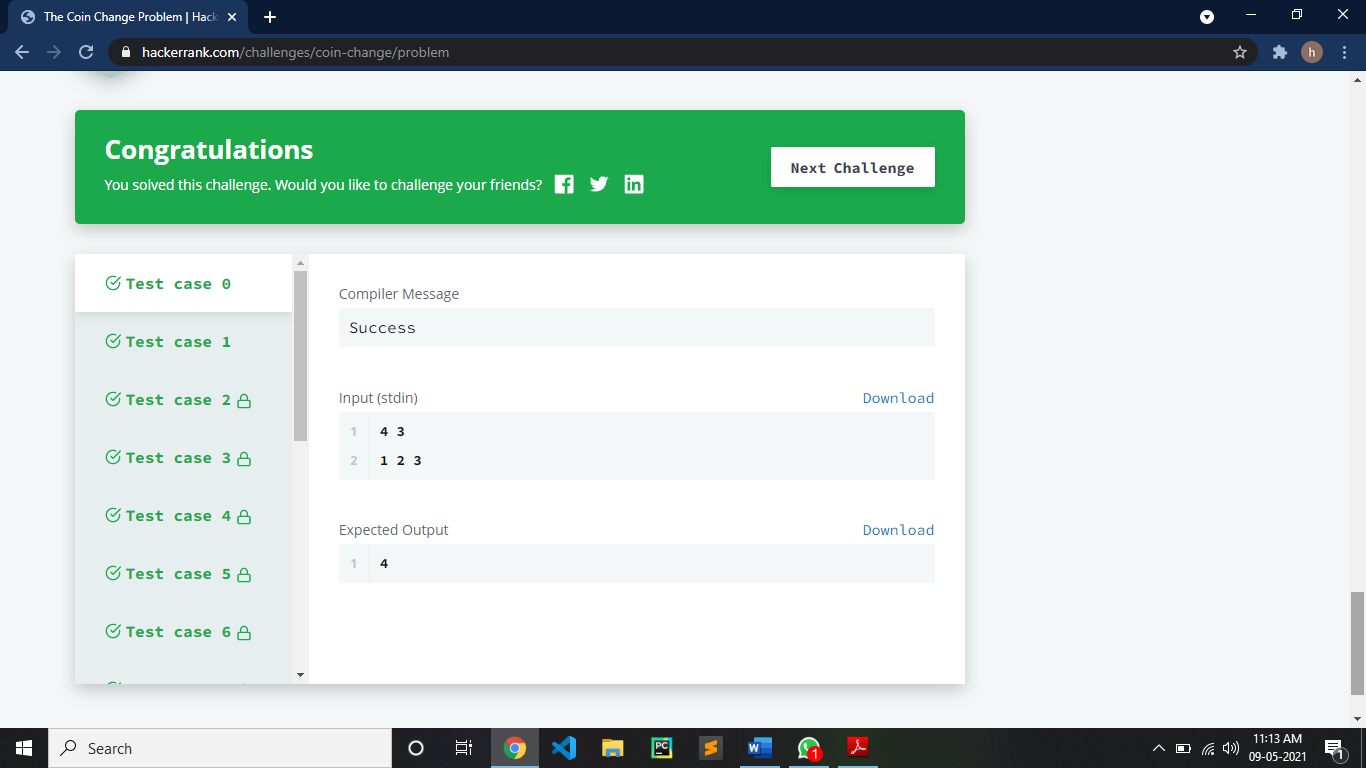
for j in range(coins[i],money+1):

counts[j] += counts[j-coins[i]]

return counts[money]

print(count\_make\_change\_bottom\_up(money,coins,no\_of\_coins))

**OUTPUT:**

****

**CONCLUSION :-**

* Time Complexity is O(N\*M)
* space complexity is O(N).