# Algorithmic Design and Techniques – example exam

## Q1. (18%)

Order the following functions by increasing growth order.

(6) =

## Q2. (10%)

What is time complexity of fun()?

|  |
| --- |
| int fun(int n)  {    int count = 0;    for (int i = n; i > 0; i /= 2)       for (int j = 0; j < i; j++)          count += 1;    return count;  } |

## Q3. (22%)

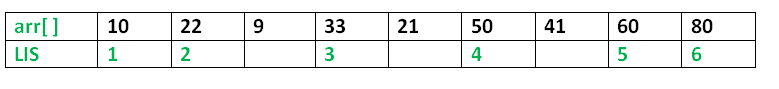
The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order.

For example, the length of LISes are:

{10, 22, 9, 33, 21, 50, 41, 60, 80}

is 6 and the LIS is

{10, 22, 33, 50, 60, 80}.



More Examples:

Input : arr[] = {3, 10, 2, 1, 20}

Output : Length of LIS = 3, The longest increasing subsequence is 3, 10, 20

Input : arr[] = {3, 2}

Output : Length of LIS = 1, The longest increasing subsequences are {3} and {2}

Input : arr[] = {50, 3, 10, 7, 40, 80}

Output : Length of LIS = 4, The longest increasing subsequence is {3, 7, 40, 80}

Write a C function, that calculates the length of the longest subsequence of a given sequence, using dynamic programming.

The header of the function is:

int lis( int arr[], int n )

where n is the number of elements in the array.

## Q4. (20%)

A problem from the operating system world:

We have n servers in our computer system. Each server i, is currently processing a(i) number of requests.

There is another array b in which b(i) represents the number of incoming requests that are scheduled to server i.

Each b(i) can be rescheduled to only one server, it may not be split between 2 or more servers.

Reschedule the incoming requests in such a way that each server i holds an **equal** number of requests after rescheduling.

An incoming request to server i (in other words - b(i)), can be rescheduled only to server i-1, i, i+1.

If there is no such rescheduling possible then output -1, else print the number of requests held by each server after rescheduling.

Examples:

Input :

a = {6, 14, 21, 1}

b = {15, 7, 10, 10}

Output : 21

b(0) scheduled to a(0) --> a(0) = 21

b(1) scheduled to a(1) --> a(1) = 21

b(2) scheduled to a(3) --> a(3) = 21

b(3) scheduled to a(3) --> a(3) = 21

a(2) remains unchanged --> a(2) = 21

Input : a = {1, 2, 3}

b = {1, 100, 3}

Output : -1

No rescheduling will result in equal requests.

Write the requested function. The functions’ signature is:

int solve(int a[], int b[], int n)

Where, n is the number of servers.

Prove that the choice your greedy algorithm takes, is a safe choice.

## Q5. (10%)

Given an array of **n** integers. The task is to find the largest number which is a perfect square. Print -1 if there is no number that is perfect square.

You may assume that the sqrt function is O(1).

**Examples**:

**Input** : arr[] = {16, 20, 25, 2, 3, 10}

**Output** : 25

**Explanation**: 25 is the largest number

that is a perfect square.

**Input** : arr[] = {36, 64, 10, 16, 29, 25|

**Output** : 64

A **Simple Solution** is to sort the elements and sort the n numbers and start checking from the end for a perfect square number using the sqrt() function. The first number from the end which is a perfect square number is our answer. What is the asymptotic complexity of this algorithm?

There is a more efficient solution (when judging by the criteria of the worst-case complexity) – please suggest such a solution, and state what is its worst-case complexity?

## Q6. (20%)

Given a circularly sorted array of integers, find the number of times the array is rotated.

Assume there are no duplicates in the array and the rotation transfers the last integer to the first place in the array.

For example;

Input: arr = [8, 9,10,2, 5,6]

Output: The array is rotated 3 times

Input: arr = [2, 5, 6, 8,9, 10]

Output: The array is rotated 0 times

Write an efficient solution to this problem

The functions’ signature is:

int findRotationCount(int arr[], int n);

(where n is the number of elements in the array).

Good – Luck!!!