Practice Problems

1 Problems

Answers for these problems are appended at the end. We **strongly recommend** you to try these problems yourself first before looking at the answers.

1.1 H-Index

The h-index is a number based on the set of a scientists most cited papers. It is defined as follows : A scientist has index h if h of his N papers have at least h citations each, and the other $(N - h)$ papers have at most h citations each.
Albert Einstein, for example, published 319 papers in scientific journals and has an h-index equal to 46. It means 46 of his papers have received 46 or more citations each, and all of his remaining 273 papers have 46 citations or less each.
The h-index was suggested in 2005 by Jorge E. Hirsch and is sometimes called the Hirsch index or Hirsch number.
Given the information of how many citations each paper from a given researcher has received, fill in the following function to return the h-index of the researcher.

```
//  The function takes an array A as the argument, where A[i]
//  is the number of citations of the researcher's ith
//  paper. The function also takes n as the argument, where n
//  is the number of papers (== length of array A )

int compute_h_index ( int A [] , int n ) {
    int i, h_index;

    sort(A, n); // assume the sort function sorts the array A in increasing order

    for ( i = ___________________; i >= 0; i--) {
        if (___________________)
        ___________________;
    }

    h_index =___________________;
    return h_index ;
}
```

1.2 Von Neumann Ordinal

The von Neumann integer i is defined as follows: for $i = 0$, it is the empty set; for $i > 0$, it is the set containing the von Neumann integers 0 to $i - 1$.

```
0 = {}          = {}
1 = {0}         = {{}}
2 = {0, 1}      = {{}, {{}}}
3 = {0, 1, 2}   = {{}, {{}}, {{}, {{}}}}
```

Write a recursive function vonNeumann() that takes a non-negative integer N and prints a string representation of the von Neumann integer N.

For example, call to vonNeumann(3) prints

```
{{}, {{}}, {{}, {{}}}}
```

1.3 A variant of binary search

The following program contains an incomplete function rmbsearch. It is a modified "Binary Search" that computes the index of the right-most occurrence of a key in a sorted array a of size elements.
Complete the function. The function must run in time proportional to log of the size of the array.

```
/* Returns the rightmost occurrence of the key in an array .
* If the key is not found , it returns -1.
* IDEA : The key is in the region a [left...right] */

int rmbsearch (int a[], int size, int key) {
    int left = 0 , right = size - 1 , mid ;
    while (___________________) {
        mid = ___________________;

        if (a[mid] == key)
            ___________________;
        else if (a[mid] < key)
            left = ___________________;
        else
          right = ___________________;
    }

    if (a[left] == key)
        return ___________________;
    else
        return -1;
}
```

1.4 McCarthy 91 function

Consider the following snippet of code and find the value of **fun(99)**. Also try to analyse what this function intuitively does.

```
int fun (int n) {
    if (n > 100)
      return n - 10;
    return fun(fun(n+11));
}
```

1.5 Problems in Byteland

In Byteland they have a very strange monetary system. Each Bytelandian gold coin has an integer number written on it. A coin n can be exchanged in a bank into three coins: n/2, n/3 and n/4. But these numbers are all rounded down (the banks have to make a profit). You can also sell Bytelandian coins for American dollars. The exchange rate is 1:1. But you can not buy Bytelandian coins. You have one gold coin of value 'n'.

Write a program that takes as input integer 'n' and outputs the maximum amount of American dollars you can get for it. (Hint: Maximum value is not $max(n, n/2 + n/3 + n/4)$)

Input: 12
Output: 13

Input: 2
Output: 2

Input: 24
Output: 27

1.6 The Whatsapp Puzzle

Cost of one chocolate is Re.1 and as a special promotion scheme you get 1 chocolate for three wrappers. Write a function to find the maximum number of chocolates you can get in rupees 'n'?

1.7 Playing with matrices

Consider a two-dimensional square matrix B of size NxN (N rows, N columns) where each element of the matrix is either character X or character Z. Moreover, all the rows in the array are of the form Xs followed by Zs, i.e., the first part of the row consists of Xs and then the rest contains only Zs. Of course, in a particular row, there may be only Xs or only Zs. This matrix can be alternatively stored in the form of a single dimensional array A of size N where A[i] stores the index of the first Z in the ith row of the matrix B. If there is no Z in the row, A[i] stores N.

For example, if the matrix B is a 5X5 matrix, having values:

```
Z Z Z Z Z
X X Z Z Z
X Z Z Z Z
X X Z Z Z
X X X X X
```

then the corresponding single dimensional array A is **[0, 2, 1, 2, 5]**.

(a) Complete the following function fetchB() that takes this single dimension array A and two indices i and j as input and returns the value of B[i][j] (i.e., the [i][j]th entry of the original two-dimensional matrix).

```
char fetchB ( int A [] , int i , int j ) {}
```

(b) Now complete the following function convert that takes the matrix B and the size (N) as input, and converts B to the single dimensional array A. A and B have the relation as described above.

```
// Convert 2 D matrix B of size NxN to
// 1 D matrix A of size N

___________________ convert ( char **B, int N) {
    ___________________ A;
    A = ___________________;

    int i, j;
    for (i = 0; ___________________ ; i++) {
        for (j = 0; j < N ; j++) {
            if ( ___________________ ) {
                A[ ___________________ ] = ___________________;
                ___________________;
            }
        }
        if ( j == N )
            ___________________;
    }
  return A;
}
```

1.8 Finding missing integer

Given integer n, you are given a array containing $n - 1$ distinct integers in the range of 1 to n. As can be seen from the size of array, one of the integers from this range is missing in the array. Write an **efficient** code for the following function to find the missing integer.

```
int getMissing(int a[], int n);
```

For example:
For arguments : a[] = [1, 2, 4, 6, 3, 7, 8], n = 8,
Return value should be : 5

For arguments : a[] = [1, 2, 3], n = 4,
Return Value : 4

Note : By efficient, here we mean that you should only make a single pass over the array, ie. you iterate over the elements of the array only once.

1.9 Rotating a matrix

Given an N x N 2D matrix, **complete** the following function which rotates it by 90 degrees (anti-clockwise).

For example, before rotation :

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
```

After rotation :

```
 4  8 12 16
 3  7 11 15
 2  6 10 14
 1  5  9 13
```

```
void rotateMatrix(int mat[][N])
{
    for (int x = 0; x < N / 2; x++)
    {
        // Consider elements in groups of 4
        for (int y = x; y < N-x-1; y++)
        {
            int temp = ___________________;

            mat[x][y] = mat[y][N-1-x];

            ___________________ = ___________________;

            mat[N-1-x][N-1-y] = ___________________;

            ___________________ = temp;
        }
    }
}
```

2 Answers

2.1 H-Index

```c
int compute_h_index (int A[], int n ) {
    int i, h_index;

    sort(A, n); // assume the sort function sorts the array A in increasing order

    for (i = n-1; i >= 0; i--) {
        if (A[i] < n-i)
            break;
    }

    h_index =  n-1-i;
    return h_index ;
}
```

2.2 Von Neumann Ordinal

```c
void vonNeumann(int N) {      // Assuming N >= 0
    printf("{");
    for (int i = 0; i<= N-1; i++) {
        vonNeumann(i);
        if (i != N-1)
            printf(", ");
    }
    printf("}");
    return;
}
```

2.3 A variant of binary search

```c
int rmbsearch ( int a[] , int size , int key ) {
    int left = 0 , right = size - 1 , mid;
    while (left < right) {
        mid = (left + right + 1) / 2;
        if (a[mid] == key)
            left = mid;
        else if (a[mid] < key)
            left = mid + 1;
        else right = mid - 1;
    }

    if (a[left] == key)
        return left;
    else
        return -1;
}
```

2.4 McCarthy 91 function

Output : 91

Return value of fun() is 91 for all integer rguments $n <= 101$, and $n - 10$ for $n > 101$. This function is known as **McCarthy 91 function**. To read more, visit https://en.wikipedia.org/wiki/McCarthy_91_function.

2.5 Problems in Byteland

```c
#include<stdio.h>

int max(int a, int b) {
if (a > b)
  return a;
else
  return b;
}

int exchange(int n) {
  if (n == 0)
    return 0;
  else {
     return max(n, exchange(n/2) + exchange(n/3) + exchange(n/4));
  }
}

int main()
{
    printf("%d\n", exchange(24));
    return 0;
}
```

2.6 The Whatsapp Puzzle

```c
int choco(int n) {
  int count = n;
  while (n>=3) {
    count += n/3;
    n = n/3 + n%3;
  }
  return count;
}
```

2.7 Playing with matrices

(a)

```c
char fetch (int A[], int i, int j) {
    if (j < A[i])
        return 'X';
    return 'Z';
}
```

(b)

```c
int * convert ( char **B, int N) {
    int * A;
    A = (int *) malloc(sizeof(int) * N);

    int i, j;
    for (i = 0; i < N ; i++) {
        for (j = 0; j < N; j++) {
            if (B[i][j] == 'Z') {
                A[i] = j;
                break;
            }
        }
        if (j == N)
            A[i] = N;    // or j
    }
    return A;
}
```

2.8 Finding missing integer

Simply find the sum of all numbers in the array a[] and subtract it from $\frac{n(n+1)}{2}$.

```
int getMissing (int a[], int n) {
    int i, total;
    total  = n*(n+1)/2;
    for ( i = 0; i< n; i++)
       total -= a[i];
    return total;
}
```

2.9 Rotating a matrix

```
void rotateMatrix(int mat[][N])
{
    // Consider all squares one by one
    for (int x = 0; x < N / 2; x++)
    {
        // Consider elements in group of 4
        for (int y = x; y < N-x-1; y++)
        {
            // store current cell in temp variable
            int temp = mat[x][y];

            // move values from right to top
            mat[x][y] = mat[y][N-1-x];

            // move values from bottom to right
            mat[y][N-1-x] = mat[N-1-x][N-1-y];

            // move values from left to bottom
            mat[N-1-x][N-1-y] = mat[N-1-y][x];

            // assign temp to left
            mat[N-1-y][x] = temp;
        }
    }
}
```