DAY 2

INTERVIEW BIT PROBLEMS :

1. Min Steps in Infinite Grid

You are in an infinite 2D grid where you can move in any of the 8 directions :

(x,y) to

(x+1, y),
(x - 1, y),
(x, y+1),
(x, y-1),
(x-1, y-1),
(x+1,y+1),
(x-1,y+1),
(x+1,y-1)

You are given a sequence of points and the order in which you need to cover the points. Give the minimum number of steps in which you can achieve it. You start from the first point.

Input :
Given two integer arrays A and B, where A[i] is x coordinate and B[i] is y coordinate of ith point respectively.

Output :
Return an Integer, i.e minimum number of steps.

Example :

Input : [(0, 0), (1, 1), (1, 2)]
Output : 2
It takes 1 step to move from (0, 0) to (1, 1). It takes one more step to move from (1, 1) to (1, 2).

CODE :

C

```
/**
 * @input A : Integer array
 * @input n1 : Integer array's ( A ) length
 * @input B : Integer array
 * @input n2 : Integer array's ( B ) length
 *
 * @Output Integer
 */
int coverPoints(int* A, int n1, int* B, int n2) {
    int min_steps=0;
    int x_c=0;
    int y_c=0;
    int i;
    for(i=0;i<n1-1;i++){
        x_c=abs(A[i]-A[i+1]);
        y_c=abs(B[i]-B[i+1]);
        if(x_c>y_c){
            min_steps=min_steps+x_c;
        }
        else{
            min_steps=min_steps+y_c;
        }
    }
    return min_steps;
}
```

PYTHON:

```
class Solution:
    # @param A : list of integers
    # @param B : list of integers
    # @return an integer
```

```
def coverPoints(self, A, B):
    min_steps=0
    x_c=0
    y_c=0
    for i in range(len(A)-1):
        x_c=abs(A[i]-A[i+1])
        y_c=abs(B[i]-B[i+1])
        min_steps=min_steps+max(x_c,y_c)
    return min_steps
```

2. Add One To Number

Given a non-negative number represented as an array of digits, add 1 to the number (increment the number represented by the digits).
The digits are stored such that the most significant digit is at the head of the list.

Example:
If the vector has [1, 2, 3]
the returned vector should be [1, 2, 4]
as 123 + 1 = 124.

NOTE: Certain things are intentionally left unclear in this question which you should practice asking the interviewer.
For example, for this problem, following are some good questions to ask :

Q : Can the input have 0's before the most significant digit. Or in other words, is 0 1 2 3 a valid input?

A : For the purpose of this question, YES

Q : Can the output have 0's before the most significant digit? Or in other words, is 0 1 2 4 a valid output?

A : For the purpose of this question, NO. Even if the input has zeroes before the most significant digit.

CODE:

C++

```
vector<int> Solution::plusOne(vector<int> &A) {
    int n=A.size()-1;
    int sum=A[n]+1;
    A[n]=sum%10;
    int carry=sum/10;
    for(int i=n-1;i>=0;i--)
    {
        int sum=A[i]+carry;
        A[i]=sum%10;
        carry=sum/10;
    }
    vector<int> B;
    if(carry==1)
    {
        B.push_back(1);
        for(int i=0;i<A.size();i++)
            B.push_back(A[i]);
        return B;
    }
    else
    {
        bool done=false;
        for(int i=0;i<A.size();i++)
        {
            if(A[i]!=0)
                done=true;
            if(done)
                B.push_back(A[i]);
        }
    }
    return B;
}
```

PYTHON

```
class Solution:
    # @param A : list of integers
    # @return a list of integers
    def plusOne(self, A):
        n=len(A)
        A[n-1]+=1
        carry=A[n-1]//10
        A[n-1]=A[n-1]%10
        for i in range(n-2,-1,-1):
            if carry==1:
                A[i]+=1
                carry=A[i]//10
                A[i]=A[i]%10
        if carry==1:
            A.insert(0,1)
        for i in range(len(A)):
            if A[i]==0:
                continue
            else:
                return A[i:]
        return A
```

3. Max Sum Contiguous Subarray

Find the contiguous subarray within an array, A of
length N which has the largest sum.

Input Format:

The first and the only argument contains an integer array, A.

Output Format:
Return an integer representing the maximum possible sum of the contiguous subarray.
Constraints:
1 <= N <= 1e6
-1000 <= A[i] <= 1000
For example:
Input 1:
A = [1, 2, 3, 4, -10]

Output 1:
10

Explanation 1:
The subarray [1, 2, 3, 4] has the maximum possible sum of 10.

Input 2:
A = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Output 2:
6

Explanation 2:
The subarray [4,-1,2,1] has the maximum possible sum of 6.

CODE :

PYTHON

```
class Solution:
    # @param A : tuple of integers
    # @return an integer
    def maxSubArray(self, A):
        maxi=-1*pow(2,64)
        c_maxi=0
        n=len(A)
        for i in range(n):
            c_maxi=c_maxi+A[i]
            if c_maxi>maxi:
                maxi=c_maxi
            if c_maxi<0:
                c_maxi=0
        return maxi
```

C++

```
int Solution::maxSubArray(const vector<int> &A) {
    int n=A.size();
    int maxi=-100000000000;
    int c_maxi=0;
    for(int i=0;i<n;i++){
        c_maxi+=A[i];
        if(c_maxi>maxi){
            maxi=c_maxi;
        }
        if(c_maxi<0){
            c_maxi=0;
        }
    }
    return maxi;
}
```

4. Maximum Absolute Difference

You are given an array of N integers, A1, A2 ,…, AN. Return maximum value of f(i, j) for all 1 ≤ i, j ≤ N.
f(i, j) is defined as |A[i] - A[j]| + |i - j|, where |x| denotes absolute value of x.
For example,

A=[1, 3, -1]

f(1, 1) = f(2, 2) = f(3, 3) = 0
f(1, 2) = f(2, 1) = |1 - 3| + |1 - 2| = 3
f(1, 3) = f(3, 1) = |1 - (-1)| + |1 - 3| = 4
f(2, 3) = f(3, 2) = |3 - (-1)| + |2 - 3| = 5

So, we return 5.

CODE :

PYTHON

TIME COMPLEXITY : O(n^2)

```
class Solution:
    # @param A : list of integers
    # @return an integer
    def maxArr(self, A):
        n=len(A)
        abs_diff=0
        output=0
        for i in range(n):
            for j in range(i,n):
                abs_diff=abs(A[i]-A[j])+abs(i-j)
                if abs_diff>output:
                    output=abs_diff
        return output
```

TIME COMPLEXITY : O(n)

```
class Solution:
    # @param A : list of integers
    # @return an integer
```

```
def maxArr(self, A):
        n=len(A)
        ma1=-1*pow(2,64)
        mi1=1*pow(2,64)
        ma2=-1*pow(2,64)
        mi2=1*pow(2,64)

        for i in range(n):
            ma1=max(ma1,A[i]+i)
            mi1=min(mi1,A[i]+i)
            ma2=max(ma2,A[i]-i)
            mi2=min(mi2,A[i]-i)

        return max(ma1-mi1,ma2-mi2)
```

C++

```
int Solution::maxArr(vector<int> &A) {
    int n=A.size();
    int ma1=INT_MIN,ma2=INT_MIN, mi1=INT_MAX,
mi2=INT_MAX;
    for(int i=0;i<n;i++){
        ma1=max(ma1,A[i]+i);
        mi1=min(mi1,A[i]+i);
        ma2=max(ma2,A[i]-i);
        mi2=min(mi2,A[i]-i);
    }
    return max(ma1-mi1,ma2-mi2);
}
```

5. Repeat and Missing Number Array

You are given a read only array of n integers from 1 to n. Each integer appears exactly once except A which appears twice and B which is missing.

Return A and B.

Note: Your algorithm should have a linear runtime complexity.
Could you implement it without using extra memory?
Note that in your output A should precede B.

Example:

Input:[3 1 2 5 3]

Output:[3, 4]

A = 3, B = 4

CODE:

C++

```
vector<int> Solution::repeatedNumber(const vector<int> &A) {
	long long int n=A.size();
	long long int
arr_sum=(n*(n+1))/2,arr_sum_sqr=(n*(n+1)*(2*n+1))/6;
	long long int miss=0;
	long long int repeat=0;

	for(int i=0;i<n;i++){
		arr_sum-=(long long int)A[i];
		arr_sum_sqr-=(long long int)A[i] * (long long int)A[i];
	}

	miss=(arr_sum+(arr_sum_sqr/arr_sum))/2;
	repeat=miss-arr_sum;

	vector<int>result;
	result.push_back(repeat);
	result.push_back(miss);
	return result;
}
```

PYTHON

```
class Solution:
# @param A : tuple of integers
# @return a list of integers
def repeatedNumber(self, A):
    output=[]
    A=list(A)
    n=len(A)
    for i in range(n):
        if A[abs(A[i])-1]>0:
            A[abs(A[i])-1]=-A[abs(A[i])-1]
        else:
            output.append(abs(A[i]))
    for i in range(n):
        if A[i]>0:
            output.append(i+1)
            break
    return output
```