Longest consecutive sequence

```
def lcs(nums):
    longest=0
    num-set = set(nums)
    for n in num-set:
        if n-1 not in num-set:
            length=0
            while n+length in num-set:
                length+=1
            longest = max(longest,
                          length)
    return longest
```

Graph valid tree (check if undirected graph is tree)

```
def validTree(n, edges):
    if not n:
        return True
    adj = {i:[] for i in range(n)}
    for n1,n2 in edges:
        adj[n1].append(n2)
        adj[n2].append(n1)
    visit = set()
    def dfs(i, prev):
        if i in visit:
            return False
        visit.add(i)
        for j in adj[i]:
            if j==prev:
                continue
            if not dfs(j,i):
                return False
        return True
    return dfs(0,-1) and n==len(visit)
```

Nbr. of connected components in an undirected graph

```
def countComponents(n, edges):
    graph={i:[] for i in range(n)}
    for u,v in edges:
        graph[u].append(v)
        graph[v].append(u)
    visited=set()
    def dfs(node):
        visited.add(node)
        for neigh in graph[node]:
            if neigh not in visited:
                dfs(neigh)
    nbr_comp=0
    for i in range(n):
        if i not in visited:
            num_comp += 1
            dfs(i)
    return num-comp
```

INTERVAL (INTERVAL)

Insert interval (into non-overlapping intervals sorted by start)

```
def insert(intervals, newInterval):
    result=[]
    i, n = 0, len(intervals)
    while i<n and intervals[i][1] <
          newInterval[0]:
        newInterval[0] :
        result.append(intervals[i])
        i += 1
    while i<n and intervals[i][0] <=
          newInterval[1]:
        newInterval[0]=min(newInterval[0],
                           intervals[i][0])
        newInterval[1]=max(newInterval[1],
                           intervals[i][1])
        i +=1
    result.append(newInterval)
    while i<n:
        result.append(intervals[i])
        i += 1
    return result
```

Merge Intervals (Merge Overlapping Intervals)

```
def merge(intervals):
    if not intervals:
        return None
    intervals.sort(key=lambda x: x
                                 [0])
    merged = []
    merged.append(intervals[0])
    for i in range(1, len(intervals)):
        last_merged= merged[-1]
        if intervals[i][0] <= last_merged
                                     [1]:
            last_merged[1] = max(
                last_merged[1], intervals
                                [i][1])
        else:
            merged.append(intervals[i])
    return merged
```

Non-overlapping Intervals (min. nbr. of intervals to remove to make non-overlapping)

```
def eraseOverlapIntervals(intervals):
    intervals.sort(key=lambda x: x[1])
    end= float('-inf')
    non_overlapping =0
    for i in range(len(intervals)):
        if intervals[i][0] >= end:
            non-overlapping +=1
            end= intervals[i][1]
    return len(intervals) - non-overlapping
```

Meeting Rooms (Are all intervals non-overlapping?)

```
def canAttendAllMeetings(intervals):
    intervals.sort(key = lambda x: x[0])
    for i in range(1, len(intervals)):
        i1 = intervals[i-1]
        i2 = intervals[i]
        if i1[1] > i2[0]:
            return False
    return True
```

Meeting Rooms 2 (Min. nbr of rooms needed)

```
def minMeetingRooms(intervals):
    if not intervals:
        return 0
    times=[(el[0],0) for el in intervals]
        +[(el[1],1) for el in intervals]
    times.sort(key = lambda x: x[0])
    cnt=0
    max_cnt = float('-inf')
    for t in times:
        if t[1]==0:
            cnt+=1
        else:
            cnt -=1
        max_cnt=max(cnt, max
    return max-cnt
```

ARRAY (Array)

Best time to buy & sell stock (max a[j]-a[i] where i<

```
def maxProfit(prices):
    min-price = float('inf')
    max-profit = 0
    for price in prices:
        if price < min-price:
            min-price = price
        elif price-min-price>ma
            max-profit = price-m
    return max-profit
```

Product of array except itsel

```
def productExceptSelf(nums)
    n= len(nums)
    answer = [1]* n
    left-prod = 1
    for i in range(n):
        answer[i] = left-product
        left-prod *= nums[i]
    right-prod=1
    for i in range(n-1, -1, -1):
        answer[i] *= right-prod
        right-prod *= nums[i]
    return answer
```

Maximum Sum Subarray

```
def maxSubarr(nums):
    cur_sum= nums[0]
    max_sum= nums[0]
    for num in nums[1:]:
        cur_sum=max(num, cur_
                         nu
        max_sum=max(max_sum
    return max_sum
```

Maximum Product Subarra

```
def maxProd(nums):
    if not nums:
        return 0
    cur_max = nums[0]
    cur_min = nums[0]
```