Approach 1

Take first element and compare it with all the remaining elements to the right. Ignore smaller elements. Consider only larger elements and update current_max variable. Repeat for all the elements. current_max is max difference.

Complexity

Time : (n-1)+(n-2)+....1 O(n^2) Space : O(1)

In [2]:

```
def findMaxDifference(arr):
    arr_len = len(arr)
    cur_max = 0
    for i in range(len(arr)-1):
        for j in range(i+1,len(arr)):
            if arr[j] < arr[i]:
                continue
            if arr[j]-arr[i] > cur_max:
                cur_max = arr[j]-arr[i]
    return cur_max
```

In [3]:

```
arr = [3,2,1,10,9,7]
findMaxDifference(arr)
#ans 9
```

Out[3]:

```
9
```

In [4]:

```
arr = [145,10,55,1,9,7]
findMaxDifference(arr)
#ans 45
```

Out[4]:

```
45
```

Approach 2 using difference array

construct difference array diff[i] = arr[i+1]-arr[i] then maximum difference in arr = maximum sum sub array in diff array computing maximum sum sub array: cur_diff = diff[0] iterate through the elements, update cur_diff by adding elements. diff[i] = diff[i] + diff[i-1] ==> take i-1 only if >0 or else ignore it. Complexity Time complexity: O(n) --> computing difference array O(n) --> finding max sum sub array Total --> O(n) Space complexity: O(n) --> for difference array We can make it order of 1 by dynamically computing diff[i]

In [5]:

```
def findDifferenceArray(arr):
    diff_arr = []
    for i in range(len(arr)-1):
        diff_arr.append(arr[i+1]-arr[i])
    return diff_arr
```

In [7]:

```
def findSumSubarray(arr):
    cur_sum = arr[0]
    for i in range(1,len(arr)):
        if arr[i-1] > 0 :
            arr[i] = arr[i]+arr[i-1]
        cur_sum = max(cur_sum,arr[i])
    return cur_sum
```

In [8]:

```
arr = [145,10,55,1,9,7]
diff_arr = findDifferenceArray(arr)
```

```
findSumSubarray(diff_arr)
#ans 45
```

Out[8]: 45

In [10]:

```
arr = [3,2,1,10,9,7]
diff_arr = findDifferenceArray(arr)
findSumSubarray(diff_arr)
#ans 9
```

Out[10]: 9

Approach 3

[,,,,b,,,,] if we have b and we want b-a to be maximum then a should be minimum of all elements to the left of b. So while scanning arr from left to right we mantain min_ele_so_far and max_diff_so_far. we update min_ele_so_far and max_diff_so_far while scanning as in the program

Complexity Time : O(n) Space : O(1)

In [17]:

```
def findMaxDifference(arr):
    min_ele_so_far = arr[0]
    max_diff_so_far = arr[1]-arr[0]
    for i in range(1,len(arr)):
        if arr[i] < min_ele_so_far:
            min_ele_so_far = arr[i]
        else:
            max_diff_so_far = max(max_diff_so_far,arr[i]-min_ele_so_far) #Try not to
    return max_diff_so_far
```

In [21]:

```
arr = [3,2,1,10,9,7]
findMaxDifference(arr)
#ans 9
```

Out[21]: 9

In [22]:

```
arr = [145,10,55,1,9,7]
findMaxDifference(arr)
#ans 45
```

Out[22]: 45

In [23]:

```
def findMaxDifference1(arr):
    min_ele_so_far = arr[0]
    max_diff_so_far = arr[1]-arr[0]
    curr_diff = arr[1]-arr[0]
    for i in range(1,len(arr)):
        if arr[i] < min_ele_so_far:
            min_ele_so_far = arr[i]
        else:
            curr_diff = arr[i]-min_ele_so_far
            if curr_diff > max_diff_so_far:
                max_diff_so_far = curr_diff
    return max_diff_so_far
```

In [ ]: