Time Complexity

```
def foo():
    x = 1  # 1 operation
    y = 2  # 1 operation
    z = 3  # 1 operation

# Cost function: No of operation your code needs

# C(foo()) : 3 operations
# Here it is constant

# Time complexity: O(1)
```

```
def foo(n):
    x = 1  # 1 operation
    y = 2  # 1 operation
    z = 3  # 1 operation

# C(foo()) : 3 operations
# Constant

# Time complexity: O(1)
```

```
def foo(n):
    x = 1  # 1 operation
    y = 2  # 1 operation
    z = 3  # 1 operation
    print(n) # 1 operation

# C(foo()) : 4 operation
# Constant

# Time complexity: O(1)

foo(12342354)
```

```
12342354
```

```
def foo(n):
    x = 1  # 1 operation
    y = 2  # 1 operation
    z = 3  # 1 operation
```

```
    for i in range(n): # n times
        print(2) # 1 operation

# C(foo()) : 3 + n

# Time complexity: O(n)

# The complexity in above function linearly grows with value of n
```

```
def foo(n):
    x = 1  # 1 operation
    y = 2  # 1 operation
    z = 3  # 1 operation

    for i in range(n): # n times
        for j in range(n):
            print(1)  # 1 operations # This is running n^2

# C(foo()) : 3 + n^2

# Complexity is growing quadratically

# Time complexity: O(n^2)

s = 0
for i in range(0, 101):
    s = s + i
```

Sorting

```
heights = [5, 1, 2, 4, 7, 3]

for i in range(len(heights) - 1):
    for j in range(len(heights) - 1):
        # Compare values
        if heights[j] > heights[j+1]:
            count += 1
            heights[j], heights[j + 1] = heights[j + 1], heights[j]

heights

[1, 2, 3, 4, 5, 7]

def bubble_sort(heights):
    for i in range(len(heights) - 1):
```

```
        for j in range(len(heights) - 1):
            # Compare values
            if heights[j] > heights[j+1]:
                # Swap
                heights[j], heights[j + 1] = heights[j + 1],
heights[j]
    return heights, count

print(bubble_sort(heights))
```

```
([1, 2, 3, 4, 5, 7], 25)
```

```
# Optimized

def bubble_sort(heights):

    for i in range(len(heights) - 1):
        for j in range(len(heights) - 1 - i):
            # Compare values
            if heights[j] > heights[j+1]:
                # Swap
                heights[j], heights[j + 1] = heights[j + 1],
heights[j]
    return heights, count

# This is a little more optimized

# Time complexity

l = [5, 4, 3, 2, 1]

bubble_sort(l)
```

```
([1, 2, 3, 4, 5], 10)
```

```
# Hw: Write a code to get the list in descending order

# Optimize this solution

# Find number or comparison

# Hw: Count number of swaps
```