```python
def count_scores(scores):
    counts = [0] * 10
    for score in scores:
        index = score // 10
        counts[index] += 1
    for i in range(10):
        print(f"[{i*10},{i*10+9}]:
{counts[i]}")
```

```python
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}

print(set1 | set2)  # Union: {1, 2, 3, 4, 5,
6}
print(set1 & set2)  # Intersection: {3, 4}
print(set1 - set2)  # Difference: {1, 2}
print(set1 ^ set2)  # Symmetric Difference:
{1, 2, 5, 6}
print(set1 <= set2)  # Subset: False
print(set1 >= set2)  # Superset: False
print(set1.isdisjoint(set2))  # Disjoint:
False

set1.add(7)
print(set1)  # {1, 2, 3, 4, 7}

set1.remove(4)
print(set1)  # {1, 2, 3, 7}

set1.clear()
print(set1)  # {}
```

```python
def find_factors(val):
    if val <= 0:
        return
    L = []
    for i in range(1, int(val**0.5) + 1):
        if val % i == 0:
            L.append((i, val//i))
    return L
```

```python
import math

class Point2d(object):
    def __init__(self, x0=0, y0=0):
        self.x = x0
        self.y = y0
    def magnitude(self):
        return math.sqrt(self.x**2 +
self.y**2)
    def dist(self, o):
        return math.sqrt((self.x - o.x)**2 +
(self.y - o.y)**2)
    def __sub__(self,o):
        return Point2d(self.x-o.x,
self.y-o.y)
    def __mul__(self,s):
        return Point2d(s*self.x, s*self.y)
    def __eq__(self,o):
        return self.x==o.x and self.y==o.y
    def __lt__(self,o):
        """This is the less than operator"""
    def __str__(self):
        return "({},{})".format(self.x,
self.y)
```

```python
pts = [ (2,5), (12,3), (12,1), (6,5), (14,
10), (12, 10), (8,12), (5,3) ]
>>> sorted( pts, key = lambda p: p[1],
reverse=True)
[(8, 12), (14, 10), (12, 10), (2, 5), (6, 5),
(12, 3), (5, 3), (12, 1)]
```

```python
def three_way_merge(L1,L2,L3):
    L = []
    i1,i2,i3 = 0,0,0
    done1,done2,done3 = False,False,False
    while not (done1 and done2 and done3):
        if not done1 and (done2 or L1[i1]
< L2[i2]) and (done3 or L1[i1] < L3[i3]):
            L.append(L1[i1])
            i1 += 1
            done1 = i1 >= len(L1)
        elif not done2 and (done3 or
L2[i2] < L3[i3]):
            L.append(L2[i2])
            2
            i2 += 1
            done2 = i2 >= len(L2)
        else:
            L.append(L3[i3])
            i3 += 1
            done3 = i3 >= len(L3)
    return L

# N^2 / O(N^2)
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

# (N-1) + (N-2) + … + 1 =(N.(N 1)/2) /
O(N^2)
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] =
arr[min_idx], arr[i]

# merging sub lists takes O(N), worst case
O(NlogN)
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = arr[:mid]
    right = arr[mid:]
    left = merge_sort(left)
    right = merge_sort(right)
    return merge(left, right)

def merge(L1,L2):
    L = []
    i1 = 0
    i2 = 0
    done1 = False
    done2 = False
    while not (done1 and done2):
        if not done1 and (done2 or L1[i1]
< L2[i2]):
            L.append(L1[i1])
            i1 += 1
            done1 = i1 >= len(L1)
        else:
            L.append(L2[i2])
            i2 += 1
            done2 = i2 >= len(L2)
    return L
```

```python
def make_runs(L):
    if len(L) == 0:
        return(L)
    newL = []
    localL = [L[0]]
    for index in range(1, len(L)):
        if L[index] >= localL[-1]:
            localL.append(L[index])
        else:
            newL.append(localL)
            localL = [L[index]]
            newL.append(localL)
    return newL

>>> print( make_runs( [7, 5, 9, 11, 2, 6,
10, 18, 19, 17] ))
[[7], [5, 9, 11], [2, 6, 10, 18, 19],
[17]]
```

```python
a = True
b = False

print(a and b)   # False
print(a or b)    # True
print(not a)     # False
print(a ^ b)     # True
print(not (a and b))  # True (NAND)
print(not (a or b))   # False (NOR)
print(not (a ^ b))    # False (XNOR)
print(not a or b)     # False
(Implication)
```

```python
def find_neighbors(point, grid):
    y, x = point
    max_y, max_x = len(grid), len(grid[0])
    directions = [(-1, 0), (1, 0), (0,
-1), (0, 1), (-1, -1), (-1, 1), (1, -1),
(1, 1)]

    neighbors = [(y + dy, x + dx) for dy,
dx in directions if 0 <= y + dy < max_y
and 0 <= x + dx < max_x]

    return neighbors
```

```python
def depth(L, i):
    if type(L) != list:
        return i
    D = [i+1]
    for l in L:
        D.append(depth(l, i+1))
    return max(D)
```