Sorting Algorithm

Selection Sort:

```
def Selection_Sort(my_ary):
    ary = my_ary[:]
    n = len(ary)
    for i in range(n):
        min = i
        for j in range(i + 1, n):
            if ary[min] > ary[j]:
                min = j
        if min != i:
            ary[min], ary[i] = ary[i], ary[min]
    return ary


ary = [5, 2, 2, 4, 8, 9, 3]
```

i	min	min value	modify	ary
init	init	init	init	5, 2, 2, 4, 8, 9, 3
0	0 - > 1	5 -> 2	Yes	2, 5, 2, 4, 8, 9, 3
1	1 -> 2,	5 -> 2	Yes	2, 2, 5, 4, 8, 9, 3
2	2 -> 3 -> 6	5 -> 4 -> 3	Yes	2, 2, 3, 4, 8, 9, 5
3	3	4	No	2, 2, 3, 4, 8, 9, 5
4	4 -> 6	8 -> 5	Yes	2, 2, 3, 4, 5, 9, 8
5	5 -> 6	9 -> 8	Yes	2, 2, 3, 4, 5, 8, 9
6	6	9	No	2, 2, 3, 4, 5, 8, 9

Running time:
(n – 1) + (n – 2) + (n – 3) + … + 1
–> (n/2)(n–1)=(n^2)/2–n/2 = θ(n^2)

Bubble Sort:

```
def Bubble_Sort(my_ary):
    ary = my_ary[:]
    for i in range(len(ary)):
        for j in range(0, len(ary)-i-1):
            if ary[j] > ary[j+1]:
                ary[j], ary[j+1] = ary[j+1], ary[j]

    return ary
```

i	j	modify	ary
init	init	init	5, 2, 2, 4, 8, 9, 3
0	0	Yes	2, 5, 2, 4, 8, 9, 3
0	1	Yes	2, 2, 5, 4, 8, 9, 3
0	2	No	2, 2, 5, 4, 8, 9, 3
0	3	Yes	2, 2, 4, 5, 8, 9, 3
0	4	No	2, 2, 4, 5, 8, 9, 3
0	5	Yes	2, 2, 4, 5, 8, 3, 9
1	0	No	2, 2, 4, 5, 8, 3, 9
1	1 -> 4	Yes	2, 2, 4, 5, 3, 8, 9
2	0 -> 3	Yes	2, 2, 4, 3, 5, 8, 9
3	0 -> 2	Yes	2, 2, 3, 4, 5, 8, 9
4	0 -> 1	No	2, 2, 3, 4, 5, 8, 9
5	0 -> 0	No	2, 2, 3, 4, 5, 8, 9

Running Time:

$$\sum_{i=0}^{n-1}(n-i-1) = n^2 - \sum_{i=0}^{n-1} i - n = n^2 - n(n-1)/2 - n = n^2 - n^2/2 + n/2 - n = n^2/2 - n/2$$

θ(n^2)

Insertion Sort:

```
def Insertion_Sort(my_arr):
    ary = my_arr[:]
    for i in range(1, len(ary)):
        key = ary[i]
        j = i - 1
        while j >= 0 and key < ary[j]:
            ary[j + 1] = ary[j]
            j -= 1
        ary[j + 1] = key
    return ary
```

i	key	j	modify	ary
init	init	init	init	5, 2, 2, 4, 8, 9, 3
1	2	0	Yes	2, 5, 2, 4, 8, 9, 3
2	2	1	Yes	2, 2, 5, 4, 8, 9, 3
3	4	2	Yes	2, 2, 4, 5, 8, 9, 3
4	8	3	No	2, 2, 4, 5, 8, 9, 3
5	9	4	No	2, 2, 4, 5, 8, 9, 3
6	3	5 -> 4 -> 3 -> 2	Yes	2, 2, 3, 4, 5, 8, 9

Running time:
2 * (1 + 2 + … + n - 2 + n - 1)
-> [2(n-1)(n-1+1)]/2 = n(n-1)
-> 0(n^2)

Merge Sort:

```
def merge(L, R):
    res = []
    while L and R:
        if L[0] < R[0]:
            res.append(L.pop(0))
        else:
            res.append(R.pop(0))
    while L:
        res.append(L.pop(0))
    while R:
        res.append(R.pop(0))
    return res


def merge_function(ary):
    if len(ary) > 1:
        mid = (int)(len(ary)/2)
        L = merge_function(ary[:mid])
        R = merge_function(ary[mid:])
        return merge(L,R)
    return ary
```

```
                    [5][2][2][4][8][9][3]
                           mid = 3

              [5][2][2][4]                [8][9][3]
                  mid = 1                 mid=1

          [5][2]     [2][4]                 [8][9]     [3]
           mid=0      mid=0                  mid=0

        [5]  [2]    [2]   [4]              [8]  [9]    [3]
```

[2][5] [2][4] [8][9] [3]

[2][2][4][5] [3][8][9]

[2][2][3][4][5][8][9]

Heap Sort:

```
def Heap_Sort(my_ary):
    ary = []
    for x in my_ary:
        heappush(ary, x)
    return [heappop(ary) for i in range(len(ary))]
```

Quick Sort:

```
def Quick_Sort_algrithm(ary, low, high):
    if low < high:
        pivot = Partition(ary, low ,high)
        Quick_Sort_algrithm(ary, low, pivot-1)
        Quick_Sort_algrithm(ary, pivot+1, high)


def Partition(ary, low, high):
    i = low - 1

    pivot = ary[high]
    for j in range(low, high):
        if ary[j] <= pivot:

            i += 1
            ary[i], ary[j] = ary[j], ary[i]
    ary[i+1], ary[high] = ary[high], ary[i+1]
    return (i+1)
```

ary = [5][2][2][4][8][9][3]
Quick_Sort(ary, 0, 6)
0 < 6:

pivot = Partition(ary, 0, 6)

i	pivot	j	modify	ary
init	init	init	init	5, 2, 2, 4, 8, 9, 3
-1 -> 0 -> 1	3	0 -> 1 -> … -> 6	Yes	2, 2, 3, 5, 4, 8, 9

……

```
[5][2][2][4][8][9][3]

[2][2]    3    [5][4][8][9]
[2] 2     3    [5][4][8]      9
[2] 2     3    [5][4]    8    9
[2] 2     3    4    [5]  8    9
[2][2][3][4][5][8][9]
```

Counting Sort:

```
def counting_sort(my_ary, max):
    max +=1
    ary = my_ary[:]
    count_ary = max*[0]

    for e in ary:
        count_ary[e]+=1

    print(count_ary)
    for i in range(1, max):
        count_ary[i] += count_ary[i-1]
    print(count_ary)

    for e in my_ary:
        ary[count_ary[e]-1] = e
        count_ary[e]-=1
    return ary
```

Index	0	1	2	3	4	5	6	7	8	9

Original array	5	2	2	4	8	9	3			
Counting array1	0	0	2	1	1	1	0	0	1	1
Counting array2	0	0	2->1->0	3->2	4->3	5->4	5	5	6->5	7->6
Sorted array	2	2	3	4	5	8	9			

Master theorem:

$T(n) = aT(n/b) + f(n)$ where $a \geq 1, b > 1$

n is the size of the problem

a is the number of subproblems in the recursion

n/b is the size of each subproblem. (Here it is assumed that all subproblems are essentially the same size.)

f(n) is the cost of the work done outside the recursive calls, which includes the cost of dividing the problem and the cost of merging the solutions to the subproblems.

There are 3 cases:

1. If $f(n) = O(n^{log_b a - \varepsilon})$ for some constant ε > 0, then $T(n) = \Theta(n^{log_b a})$.
2. If $f(n) = \Theta(n^{log_b a})$ with $k \geq 0$, then $T(n) = \Theta(n^{log_b a} lg\, n)$.
3. If $f(n) = \Omega(n^{log_b a + \varepsilon})$ and $af(n/b) \leq cf(n)$, for some $c < 1, n > n_0$, then $T(n) = \Theta(f(n))$

For example:

Merge Sort:

$T(n) = 2T(n/2) + \Theta(n)$

$a = 2$, $b = 2$; $n^{log_b a} = n^{log_2 2} = n = \Theta(n)$

Also $f(n) = \Theta(n)$

So it corresponds to Case 2: $T(n) = \Theta(n^{log_b a} lg n) = \Theta(n lg n)$