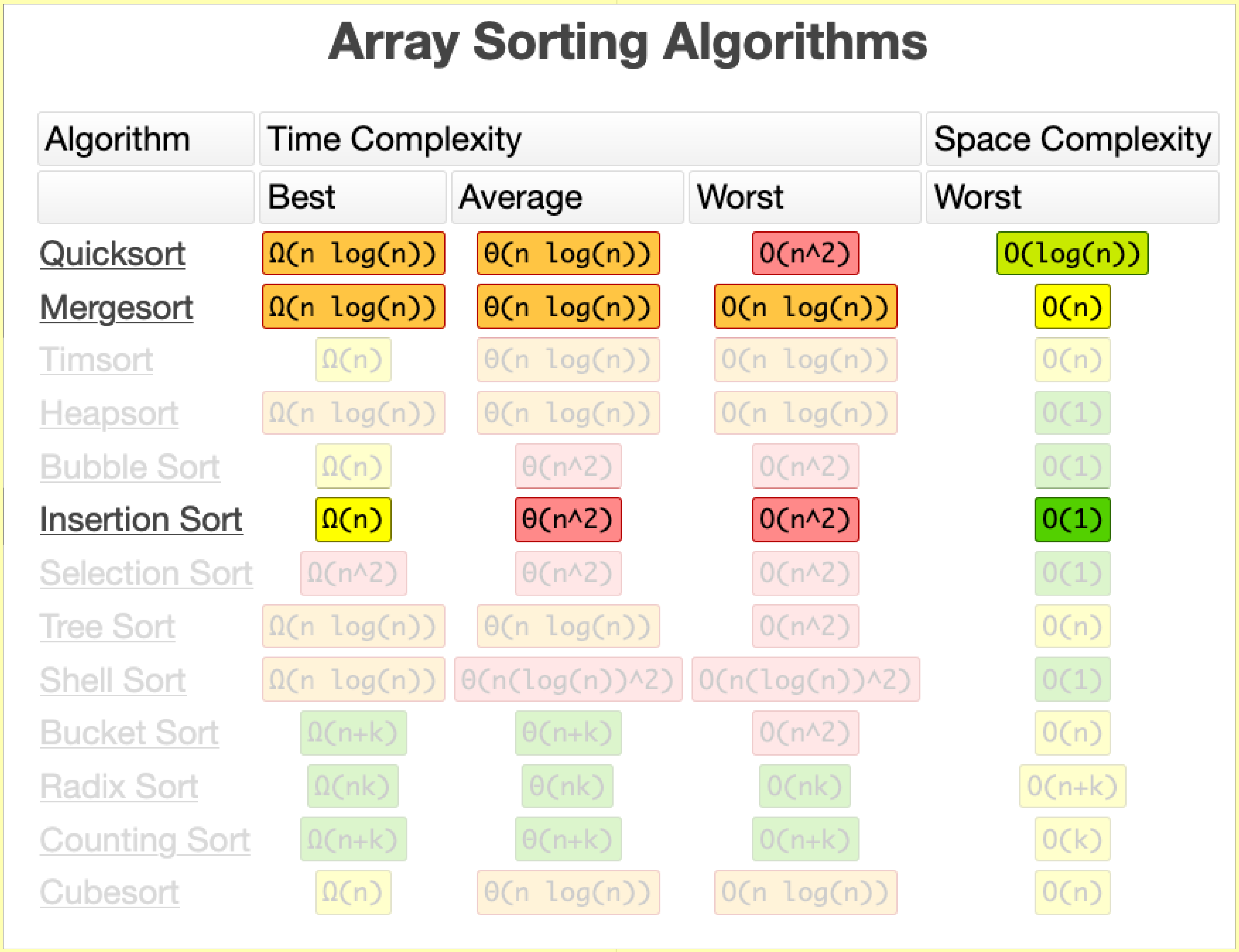

# Array Sorting Algorithms

| Algorithm | Time Complexity | | | Space Complexity |
|---|---|---|---|---|
| | Best | Average | Worst | Worst |
| Quicksort | Ω(n log(n)) | Θ(n log(n)) | O(n^2) | O(log(n)) |
| Mergesort | Ω(n log(n)) | Θ(n log(n)) | O(n log(n)) | O(n) |
| Timsort | Ω(n) | Θ(n log(n)) | O(n log(n)) | O(n) |
| Heapsort | Ω(n log(n)) | Θ(n log(n)) | O(n log(n)) | O(1) |
| Bubble Sort | Ω(n) | Θ(n^2) | O(n^2) | O(1) |
| Insertion Sort | Ω(n) | Θ(n^2) | O(n^2) | O(1) |
| Selection Sort | Ω(n^2) | Θ(n^2) | O(n^2) | O(1) |
| Tree Sort | Ω(n log(n)) | Θ(n log(n)) | O(n^2) | O(n) |
| Shell Sort | Ω(n log(n)) | Θ(n(log(n))^2) | O(n(log(n))^2) | O(1) |
| Bucket Sort | Ω(n+k) | Θ(n+k) | O(n^2) | O(n) |
| Radix Sort | Ω(nk) | Θ(nk) | O(nk) | O(n+k) |
| Counting Sort | Ω(n+k) | Θ(n+k) | O(n+k) | O(k) |
| Cubesort | Ω(n) | Θ(n log(n)) | O(n log(n)) | O(n) |

Ref: https://www.bigocheatsheet.com/

**Insertion sort:**

- Works well for small n (~ 50) or almost sorted list
- Worst case - if the list is already sorted in reverse order
- In place => space is O(1)

**Merge sort:**

- Divide and conquer
- Height of the merge sort tree is ~ log2(n)
- Why is space O(n)??

**Quick sort:**

- Divide and conquer
- Worst case is when the list is already sorted (in order, or reverse)!
- Why is space O(log2(n))??

# Insertion Sort

```
def insertion_sort(A):
    for i in range(1, len(A)):
        k = i
        print(A, k)
        while k > 0 and A[k-1] > A[k]:
            A[k-1], A[k] = A[k], A[k-1]
            k -= 1
            print(A)
        Print('--')

A = [5, 1, 2, 4, 2, 5, 8, 9, 0]
insertion_sort(A)
A
>>
[5, 1, 2, 4, 2, 5, 8, 9, 0] 1
[1, 5, 2, 4, 2, 5, 8, 9, 0]
--
[1, 5, 2, 4, 2, 5, 8, 9, 0] 2
[1, 2, 5, 4, 2, 5, 8, 9, 0]
--
[1, 2, 5, 4, 2, 5, 8, 9, 0] 3
[1, 2, 4, 5, 2, 5, 8, 9, 0]
--
[1, 2, 4, 5, 2, 5, 8, 9, 0] 4
[1, 2, 4, 2, 5, 5, 8, 9, 0]
[1, 2, 2, 4, 5, 5, 8, 9, 0]
--
[1, 2, 2, 4, 5, 5, 8, 9, 0] 5
--
[1, 2, 2, 4, 5, 5, 8, 9, 0] 6
--
[1, 2, 2, 4, 5, 5, 8, 9, 0] 7
--
[1, 2, 2, 4, 5, 5, 8, 9, 0] 8
[1, 2, 2, 4, 5, 5, 8, 0, 9]
[1, 2, 2, 4, 5, 5, 0, 8, 9]
[1, 2, 2, 4, 5, 0, 5, 8, 9]
[1, 2, 2, 4, 0, 5, 5, 8, 9]
[1, 2, 2, 0, 4, 5, 5, 8, 9]
[1, 2, 0, 2, 4, 5, 5, 8, 9]
[1, 0, 2, 2, 4, 5, 5, 8, 9]
[0, 1, 2, 2, 4, 5, 5, 8, 9]
--

[0, 1, 2, 2, 4, 5, 5, 8, 9]
```

# Merge Sort

```
def merge(A1, A2, A):
    n1, n2, n = len(A1), len(A2), len(A)
    assert n == n1 + n2
    i = j = 0
    while i+j < n:
        if j==n2 or (i < n1 and A1[i] < A2[j]):
            A[i+j] = A1[i]
            i += 1
        else:
            A[i+j] = A2[j]
            j += 1

def merge_sort(A):
    # exit
    n = len(A)
    if n < 2: return

    # divide
    mid = n//2
    A1 = A[0:mid]
    A2 = A[mid:n]
    print(A1, A2, A)

    # conquer
    merge_sort(A1)
    merge_sort(A2)

    # merge results
    print('before -', A1, A2, A)
    merge(A1, A2, A)
    print('after -', A1, A2, A)
    print('----\n')

A = [5, 1, 2, 4, 2, 5, 8, 9, 0]
merge_sort(A)
A
>>
[5, 1, 2, 4] [2, 5, 8, 9, 0] [5, 1, 2, 4, 2, 5, 8, 9, 0]
[5, 1] [2, 4] [5, 1, 2, 4]
[5] [1] [5, 1]
before - [5] [1] [5, 1]
after - [5] [1] [1, 5]
----

[2] [4] [2, 4]
before - [2] [4] [2, 4]
after - [2] [4] [2, 4]
----

before - [1, 5] [2, 4] [5, 1, 2, 4]
after - [1, 5] [2, 4] [1, 2, 4, 5]
----

[2, 5] [8, 9, 0] [2, 5, 8, 9, 0]
[2] [5] [2, 5]
before - [2] [5] [2, 5]
after - [2] [5] [2, 5]
----

[8] [9, 0] [8, 9, 0]
[9] [0] [9, 0]
before - [9] [0] [9, 0]
after - [9] [0] [0, 9]
----

before - [8] [0, 9] [8, 9, 0]
after - [8] [0, 9] [0, 8, 9]
----

before - [2, 5] [0, 8, 9] [2, 5, 8, 9, 0]
after - [2, 5] [0, 8, 9] [0, 2, 5, 8, 9]
----

before - [1, 2, 4, 5] [0, 2, 5, 8, 9] [5, 1, 2, 4, 2, 5, 8, 9, 0]
after - [1, 2, 4, 5] [0, 2, 5, 8, 9] [0, 1, 2, 2, 4, 5, 5, 8, 9]
----

[0, 1, 2, 2, 4, 5, 5, 8, 9]
```

# Quick Sort

```
def quick_sort(A):
    # exit
    if len(A) < 2: return

    # divide
    L = []
    R = []
    pivot = A.pop()
    while A:
        item = A.pop()
        L.append(item) if item <= pivot else R.append(item)
    print(pivot, L, R)

    # conquer
    quick_sort(L)
    quick_sort(R)

    # concatenate
    while len(L) > 0: A.append(L.pop(0)) # → inefficient, use a queue instead!
    A.append(pivot)
    while len(R) > 0: A.append(R.pop(0)) # → inefficient, use a queue instead!
    print(A)
    print('\n')

A = [5, 1, 2, 4, 2, 5, 8, 9, 0]
merge_sort(A)
A
>>
0 [] [9, 8, 5, 2, 4, 2, 1, 5]
5 [1, 2, 4, 2, 5] [8, 9]
5 [2, 4, 2, 1] []
1 [] [2, 4, 2]
2 [2] [4]
[2, 2, 4]


[1, 2, 2, 4]


[1, 2, 2, 4, 5]


9 [8] []
[8, 9]


[1, 2, 2, 4, 5, 5, 8, 9]


[0, 1, 2, 2, 4, 5, 5, 8, 9]


[0, 1, 2, 2, 4, 5, 5, 8, 9]
```