Sorting
Simple Sorting Algorithm (Recap)

```python
for i in range(len(A)):
    k = position of min. element between A[i] and A[N-1]
    Swap A[i] and A[k]
```

Selection Sort

Courtesy Prof P R Panda CSE, IIT Delhi
Simple Sorting Algorithm (Recap)

for \( i \) in range(len(A)):
    \( k \) = position of min. element between \( A[i] \) and \( A[N-1] \)
    Swap \( A[i] \) and \( A[k] \)

Selection Sort

for \( j \) in range(i+1, len(A)):
    if \( A[\text{min_index}] > A[j] \):
        \( \text{min_index} = j \)
Simple Sorting Algorithm (Recap)

Find Min for first time n elements: n-1 comparisions
Next time: n-2
.
.
up to 1

Total time = (n-1)+(n-2)+…+1=(n*(n-1))/2
O(n^2)

Selection Sort

Courtesy Prof P R Panda CSE, IIT Delhi
Merge Sort

Divide and conquer

Merge Sort

Divide and conquer

Merge Sort

Divide and conquer

Merge Sort

Divide and conquer

Merge Sort

Divide and conquer

Merge Sort

Divide and conquer

```python
def mergesort(L):
    if (len(L) < 2):
        return L[:]
    else:
        mid = len(L)//2
        left=mergesort(L[:mid])
        right=mergesort(L[mid:])
        return merge(left,right)
```
Merge Sort

Divide and conquer

Merge Sort

Running Time (Time Complexity as O)

Recurrence Relation:

\[ T(1) = 1 \text{ if } n=1 \]
\[ T(n) = 2T(n/2) + cn \]

Solution \( O(n \log_2 n) \)
Quick Sort

Based on partitioning in two parts such that first part is less than equal to x and right part is greater than x. If x is an element of the array then it gets located at the right place if the sequence is sorted.

A[0] <=x x >x pivot A[N-1]
def partition(arr, first, last):
    x = arr[first]  # pivot is the first element
    i = first + 1   # leftcounter
    j = last        # rightcounter

    while i < j and arr[i] <= x:
        i += 1
    while i < j and arr[j] > x:
        j -= 1
    if arr[j] > x:
        j -= 1
    while i < j:
        temp = arr[i]
        arr[i] = arr[j]
        arr[j] = temp
        i += 1
        j -= 1
    while arr[i] <= x: i += 1
    while arr[j] > x: j -= 1

    temp = arr[first]  # put the pivot at the right place at j
    arr[first] = arr[j]
    arr[j] = temp

    return j          # returns the pivot index
def partition(arr, first, last):
    x = arr[first]  # pivot is the first element
    i = first + 1   # left counter
    j = last        # right counter
    while i < j and arr[i] <= x:
        i += 1
    while i < j and arr[j] > x:
        j -= 1
    if arr[j] > x:
        j -= 1
    while i < j:
        temp = arr[i]
        arr[i] = arr[j]
        arr[j] = temp
        i += 1
        j -= 1
    while arr[i] <= x: i += 1
    while arr[j] > x: j -= 1

    temp = arr[first]  # put the pivot at the right place at j
    arr[first] = arr[j]
    arr[j] = temp

    return j  # returns the pivot index

def quicksort(L, left, right):
    if (left < right):
        pindex = partition(L, left, right)
        quicksort(L, left, pindex - 1)
        quicksort(L, pindex + 1, right)

a = [28, 26, 25, 11, 16, 12, 24, 29, 6, 10]
quicksort(a, 0, len(a) - 1)
print(a)
Quick Sort

Choice of pivot decides the performance of the algorithm. If the partitioning happens in two almost equal parts, it is an ideal case.

Time Complexity

Best Case:
T(1)=1
T(n)=2T(n/2)+cn where cn is the partitioning time

Complexity O(nlog₂n)
Quick Sort

Time Complexity

Worst Case:
\[ T(1) = 1 \]
\[ T(n) = T(n-1) + T(1) + cn \]
where \( cn \) is the partitioning time

Complexity \( O(n^2) \)

Average Case: \( O(n \log_2 n) \)
Insertion Sort

Basic idea is to insert the current element at the right place.

This may require shifting the elements.
Insertion Sort

https://www.geeksforgeeks.org/insertion-sort/
**Insertion Sort**

```python
def insertion_sort(alist):
    for i in range(1, len(alist)):
        temp = alist[i]
        j = i - 1
        while (j >= 0 and temp < alist[j]):
            alist[j + 1] = alist[j]
            j = j - 1
        alist[j + 1] = temp
```

Time Complexity
Worst Case: $O(n^2)$ when elements are sorted in the reverse order
Best Case: $O(n)$ when elements are already sorted
Bubble Sort

First Pass

Second Pass

Third Pass

Bubble Sort

def bubble_sort(alist):
    for i in range(1, len(alist)):
        for j in range(0, len(alist) - i):
            if (alist[j] > alist[j+1]):
                temp = alist[j]
                alist[j] = alist[j+1]
                alist[j+1] = temp

Time Complexity: $O(n^2)$