Asymptotic Notation and Merge Sort

Performance is important

- Algorithm might be run on a very large data set
- be efficient in terms of CPU and memory usage

1. Look at sorting algorithms of different efficiency
2. Learning how efficiency of algorithm can be determined

Ex.

- How to sort arbitrary set of numbers such as student IDs

Big O notation

F(x) = O(g(x)) for x->infinity

O(1)
O(log(log(n)))
O(log(n)
O(n)
O(n(log(n))
O(n^2)
O(2^n)
O(n!)

- Constants and lower degrees are ignored

Insertion Sort

```
def insertionSort(a[]):
    for(index in range(len(a)):
       set current value to the index
        store index
    while(position>0 and a> currentval
        swap positions
        set position to last position
    set a at position to the currentvalue
```

O(n^2)

Merge Sort

- Divide and conquer
- Recursive
- Splits it in half until it has 1 element
- Merges all lists together and sorts them

```
def mergeSort(alist):
#split
    print("Splitting ",alist)
    if len(alist)>1:
    mid = len(alist)//2
    lefthalf = alist[:mid]
    righthalf = alist[mid:]

    mergeSort(lefthalf)
    mergeSort(righthalf)
#merge
    i=0
    j=0
    k=0
    while i < len(lefthalf) and j < len(righthalf):
    if lefthalf[i] < righthalf[j]:
        alist[k]=lefthalf[i]
        i=i+1
    else:
        alist[k]=righthalf[j]
        j=j+1
    k=k+1
    while i < len(lefthalf):
        alist[k]=lefthalf[i]
        i=i+1
        k=k+1
    while j < len(righthalf):
        alist[k]=righthalf[j]
        j=j+1
        k=k+1
    print("Merging ",alist)
```