Sorting Algorithms

Essential sorting algorithms and their implementation in Python, for simplicity

Bubble Sort

```
def bubbleSort(arr):
    n = len(r)

    for i in range(n):
        swapped = False

        for j in range(n-i-1):

            if arr[j] > arr[j+1]:
                arr[j],arr[j+1] = arr[j+1],arr[j]
                swapped = True

        if not swapped:
            break

    return arr
```

Iterate through the array, with the index exclusive of its length. The boolean `swapped` is initialized to false. Then, iterate through the array in a range reduced by `i` and swap the current element with the next one if they aren't in ascending order. If this happens, `swapped` is set to `True`. At the end of the outer loop iteration, if no swaps occurred, the loop is concluded as it indicates there are no more swaps to be made.

Recursive bubble sort

```
def recursiveBubbleSort(arr, n):
    if n == 1:
        return arr

    for i in range(n-1):
        if arr[n] > arr[n+1]:
            arr[n],arr[n+1] = arr[n+1],arr[n]

    return recursiveBubbleSort(arr, n-1)
```

A simplified version that uses recursion: repeatedly swaps until all indexes are ruled out.

Insertion Sort

```
def insertionSort(arr):
    n = len(arr)

    for i in range(1,n):
        curr = arr[i]
        j = i-1

        while j >= 0 and arr[j] > key:
            arr[j+1] = arr[j]
            j--

        arr[j+1] = curr

    return arr
```

Iterate from index 1 to `n-1` on the outside and set the insertion target to `arr[i]`, which is stored in a variable not to be overwritten. Move all of the preceding elements (as long as they're less than `curr`) forward by one index and place `curr` where it belongs.

Recursive Insertion Sort

```
def recursiveInsertionSort(arr, n):
    if n <= 1:
        return

    recursiveInsertionSort(arr, n-1)

    curr = arr[n-1]
    j = n-2

    while j >= 0 and arr[j] > key:
        arr[j+1] = arr[j]
        j--

    arr[j+1] = curr
```

Selection Sort

```
def selectionSort(arr):
    for i in range(len(arr)):
        min = i

        for j in range(i+1,len(arr)):
            if arr[j] < min:
```

```
                min = arr[j]

        arr[i],arr[min] = arr[min],arr[i]

    return arr
```

Loop through the array `[0,length-1]` and set `min` to the loop variable's value. Inside, loop from `[i+1,length-1]` and set `min` to the smallest value found. At the end of the outside loop, swap `arr[i]` with the minimum value found.

Merge Sort

```
def mergeSort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        left = arr[:mid]
        right = arr[mid:]

        mergeSort(left)
        mergeSort(right)

        i = j = k = 0

        while j < len(left) and k < len(right):
            if left[j] < right[k]:
                arr[i] = left[j]
                j++
            else:
                arr[i] = right[k]
                k++
            i++

        while j < len(left):
            arr[i] = left[j]
            j++
            i++

        while k < len(right):
            arr[i] = right[k]
            k++
            i++
```

Divide and conquer algorithm: recursively sort each half of the original array and eventually merge them together, finally checking for leftovers.