Algorithm Analysis

Searching and Sorting

Brad Miller David Ranum

1/25/06

Outline

1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

Summation of the First *n* Integers

```python
def sumOfN(n):
    sum = 0
    for i in range(1,n+1):
        sum = sum + i

    return sum
```

Another Summation of the First *n* Integers

```
def foo(tom):
    fred = 0
    for bill in range(1,tom+1):
       barney = bill
       fred = fred + barney

 return fred
```

Timing the Summation

```
import time

def sumOfN(n):
    start = time.clock()

    sum = 0
    for i in range(1,n+1):
        sum = sum + i

    end = time.clock()

    return sum,end-start
```

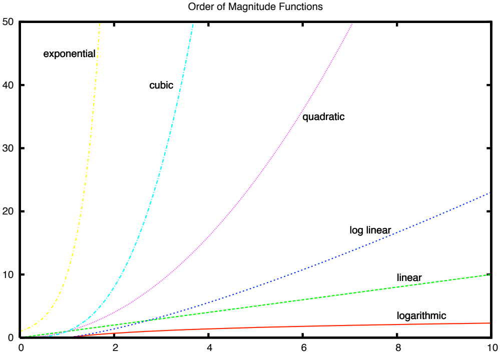
Summation Without Iteration

```python
def sumOfN3(n):
    return (n*(n+1))/2
```

Outline

1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

Plot of Common Big-O Functions



Example Python Code

```python
a=5
b=6
c=10
for i in range(n):
    for j in range(n):
        x = i * i
        y = j * j
        z = i * j
for k in range(n):
    w = a*k + 45
    v = b*b
d = 33
```

Outline

1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

Checking Off I

```
def anagramSolution1(s1,s2):
    alist = list(s2)

    pos1 = 0
    stillOK = True

    while pos1 < len(s1) and stillOK:
        pos2 = 0
        found = False
        while pos2 < len(alist) and not found:
            if s1[pos1] == alist[pos2]:
                found = True
            else:
                pos2 = pos2 + 1

```

Checking Off II

```
        if found:
            alist[pos2] = None
        else:
            stillOK = False

        pos1 = pos1 + 1

    return stillOK
```

Sort and Compare

```
def anagramSolution2(s1,s2):
    alist1 = list(s1)
    alist2 = list(s2)
    alist1.sort()
    alist2.sort()
    pos = 0
    matches = True

    while pos < len(s1) and matches:
        if alist1[pos]==alist2[pos]:
            pos = pos + 1
        else:
            matches = False
    return matches
```

Count and Compare I

```
def anagramSolution4(s1,s2):
    c1 = [0]*26
    c2 = [0]*26

    for i in range(len(s1)):
        pos = ord(s1[i])-ord(’a’)
        c1[pos] = c1[pos] + 1

    for i in range(len(s2)):
        pos = ord(s2[i])-ord(’a’)
        c2[pos] = c2[pos] + 1
```

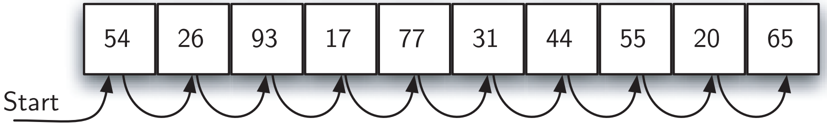
Count and Compare II

```
    j = 0
    stillOK = True
    while j<26 and stillOK:
        if c1[j]==c2[j]:
            j = j + 1
        else:
            stillOK = False

    return stillOK
```

Outline

1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

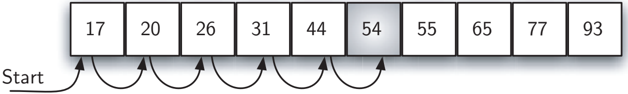
Sequential Search of a List of Integers



Sequential Search of an Unordered List

```
def sequentialSearch(alist, item):
    pos = 0
    found = False
    stop = False
    while pos < len(alist) and not found:
        if alist[pos] == item:
            found = True
        else:
            pos = pos+1

    return found
```

Sequential Search of an Ordered List of Integers



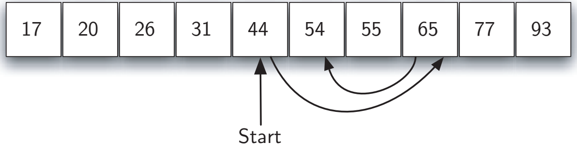
Sequential Search of an Ordered List

```
def orderedSequentialSearch(alist, item):
    pos = 0
    found = False
    stop = False
    while pos < len(alist) and not found and not stop:
        if alist[pos] == item:
            found = True
        else:
            if alist[pos] > item:
                stop = True
            else:
                pos = pos+1

    return found
```

Outline

1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

Binary search of an ordered list of integers



17	20	26	31	44	54	55	65	77	93

Binary Search of an Ordered List

```
def binarySearch(alist, item):
    first = 0
    last = len(alist)-1
    found = False
    while first<=last and not found:
        midpoint = (first + last)/2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint-1
            else:
                first = midpoint+1

    return found
```

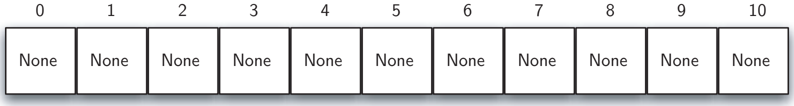
A Binary Search–Recursive Version

```
def binarySearch(alist, item):
    if len(alist) == 0:
        return False
    else:
        midpoint = len(alist)/2
        if alist[midpoint]==item:
            return True
        else:
            if item<alist[midpoint]:
              return binarySearch(alist[:midpoint],item)
            else:
              return binarySearch(alist[midpoint+1:],item)
```

Outline

1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

Hash Table with 11 Empty Slots

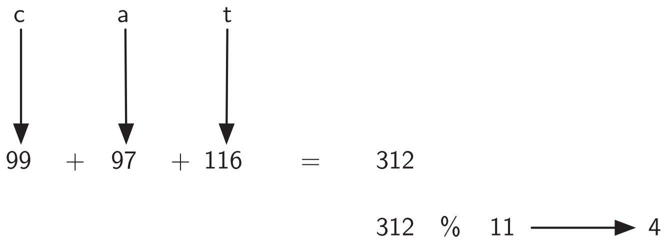


0	1	2	3	4	5	6	7	8	9	10
None	None	None	None	None	None	None	None	None	None	None

Hash Table with Six Items

0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

Hashing a String Using Ordinal Values



c a t

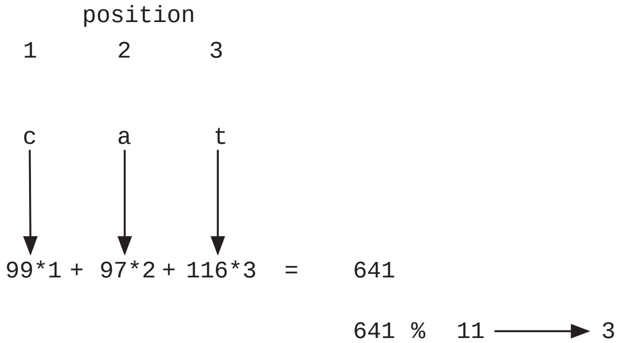
99 + 97 + 116 = 312

$312 \ \% \ 11 \longrightarrow 4$

Simple Hash Function for Strings

```python
def hash(astring, tablesize):
    sum = 0
    for pos in range(len(astring)):
        sum = sum + ord(astring[pos])

    return sum%tablesize
```

Hashing a String Using Ordinal Values with Weighting



position

1 2 3

c a t

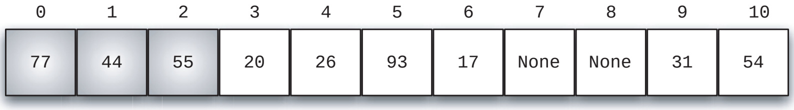
99*1 + 97*2 + 116*3 = 641

641 % 11 → 3

Collision Resolution with Linear Probing

0	1	2	3	4	5	6	7	8	9	10
77	44	55	20	26	93	17	None	None	31	54

A Cluster of Items for Slot 0



0	1	2	3	4	5	6	7	8	9	10
77	44	55	20	26	93	17	None	None	31	54

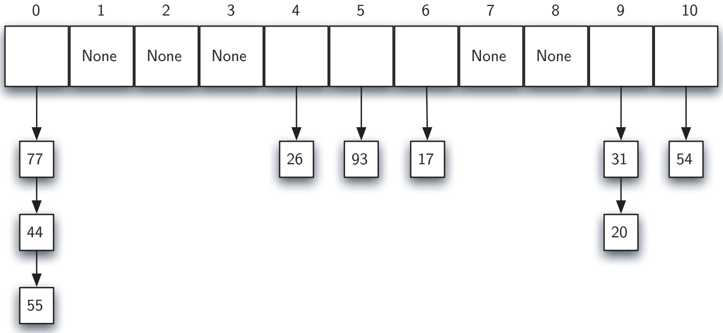
Collision Resolution Using “Plus 3”

0	1	2	3	4	5	6	7	8	9	10
77	55	None	44	26	93	17	20	None	31	54

Collision Resolution with Quadratic Probing

0	1	2	3	4	5	6	7	8	9	10
77	44	20	55	26	93	17	None	None	31	54

Collision Resolution with Chaining



- `HashTable(size)` creates a new hash table. It needs the size and returns a hash table with `size` empty slots named 0 through `size`-1.
- `store(item, data)` stores a new piece of data in the hash table using the item as the key location. It needs the item and the associated data. It returns nothing.
- `search(item)` returns the data value associated with the key item. It returns `None` if the key is not in the hash table.

HashTable Implementation in Python–Constructor

```
class HashTable:
    def __init__(self,size):
        self.slots = [None] * size
        self.data = [None] * size
```

HashTable Implementation in Python–Store Method I

```
def store(self,item,data):
  hashvalue = self.hashfunction(item,len(self.slots))

  if self.slots[hashvalue] == None:
    self.slots[hashvalue] = item
    self.data[hashvalue] = data
  else:
    nextslot = self.rehash(hashvalue,len(self.slots))
    while self.slots[nextslot] != None:
      nextslot = self.rehash(nextslot,len(self.slots))

    self.slots[nextslot]=item
    self.data[nextslot]=data
```

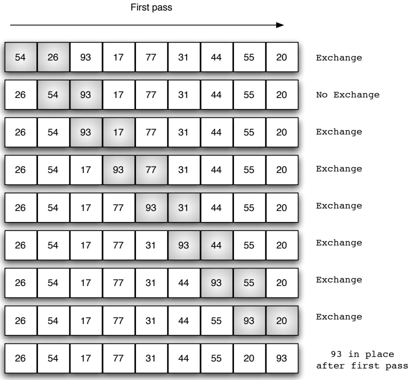
HashTable Implementation in Python–Store Method II

```
    def hashfunction(self,item,size):
     return item%size

    def rehash(self,oldhash,size):
        return (oldhash+1)%size
```

Outline

1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

bubbleSort: The First Pass

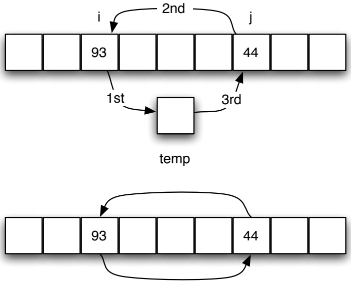


First pass

54	26	93	17	77	31	44	55	20	Exchange
26	54	93	17	77	31	44	55	20	No Exchange
26	54	93	17	77	31	44	55	20	Exchange
26	54	17	93	77	31	44	55	20	Exchange
26	54	17	77	93	31	44	55	20	Exchange
26	54	17	77	31	93	44	55	20	Exchange
26	54	17	77	31	44	93	55	20	Exchange
26	54	17	77	31	44	55	93	20	Exchange
26	54	17	77	31	44	55	20	93	93 in place after first pass

Exchanging Two Values in Python

Most programming languages require a 3-step process with an extra storage location.



In Python, exchange can be done as two simultaneous assignments.

A Bubble Sort

```
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                alist[i],alist[i+1]=alist[i+1],alist[i]
```

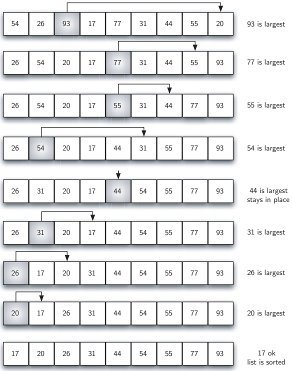
A Modified Bubble Sort

```
def shortBubbleSort(alist):
    exchanges = True
    passnum = len(alist)-1
    while passnum > 0 and exchanges:
        exchanges = False
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                exchanges = True
                alist[i],alist[i+1]=alist[i+1],alist[i]
        passnum = passnum-1
```

Outline

1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

selectionSort



54 26 93 17 77 31 44 55 20 — 93 is largest

26 54 20 17 77 31 44 55 93 — 77 is largest

26 54 20 17 55 31 44 77 93 — 55 is largest

26 54 20 17 44 31 55 77 93 — 54 is largest

26 31 20 17 44 54 55 77 93 — 44 is largest stays in place

26 31 20 17 44 54 55 77 93 — 31 is largest

26 17 20 31 44 54 55 77 93 — 26 is largest

20 17 26 31 44 54 55 77 93 — 20 is largest

17 20 26 31 44 54 55 77 93 — 17 ok list is sorted

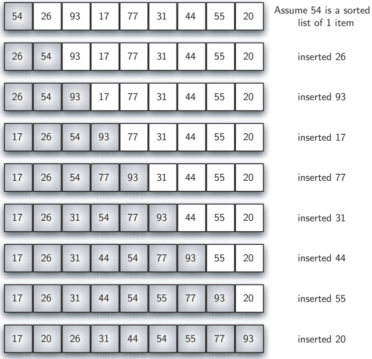
A Selection Sort

```
def selectionSort(alist):
    for fillslot in range(len(alist)-1,0,-1):
        positionOfMax=0
        for location in range(1,fillslot+1):
            if alist[location]>alist[positionOfMax]:
                positionOfMax = location

        alist[positionOfMax],alist[fillslot] = \
                           alist[fillslot],alist[positionOfMax]
```

Outline

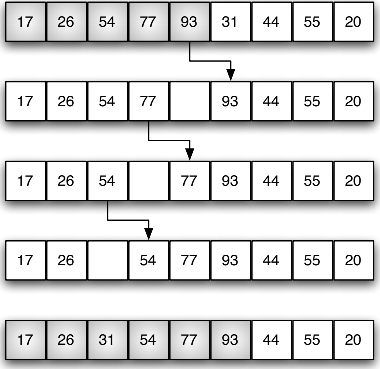
1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

insertionSort



54	26	93	17	77	31	44	55	20	Assume 54 is a sorted list of 1 item
26	54	93	17	77	31	44	55	20	inserted 26
26	54	93	17	77	31	44	55	20	inserted 93
17	26	54	93	77	31	44	55	20	inserted 17
17	26	54	77	93	31	44	55	20	inserted 77
17	26	31	54	77	93	44	55	20	inserted 31
17	26	31	44	54	77	93	55	20	inserted 44
17	26	31	44	54	55	77	93	20	inserted 55
17	20	26	31	44	54	55	77	93	inserted 20

insertionSort: Fifth Pass of the Sort



17	26	54	77	93	31	44	55	20	Need to insert 31 back into the sorted list
17	26	54	77		93	44	55	20	93>31 so shift it to the right
17	26	54		77	93	44	55	20	77>31 so shift it to the right
17	26		54	77	93	44	55	20	54>31 so shift it to the right
17	26	31	54	77	93	44	55	20	26<31 so insert 31 in this position

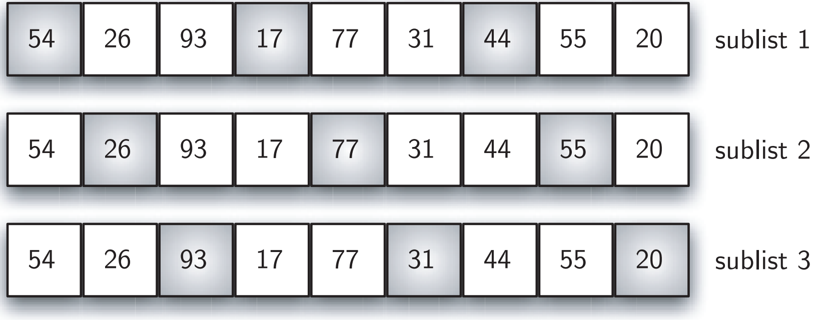
insertionSort

```
def insertionSort(alist):
   for index in range(1,len(alist)):

     currentvalue = alist[index]
     position = index

     while position>0 and alist[position-1]>currentvalue:
         alist[position]=alist[position-1]
         position = position-1

     alist[position]=currentvalue
```

Outline

1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

A Shell Sort with Increments of Three

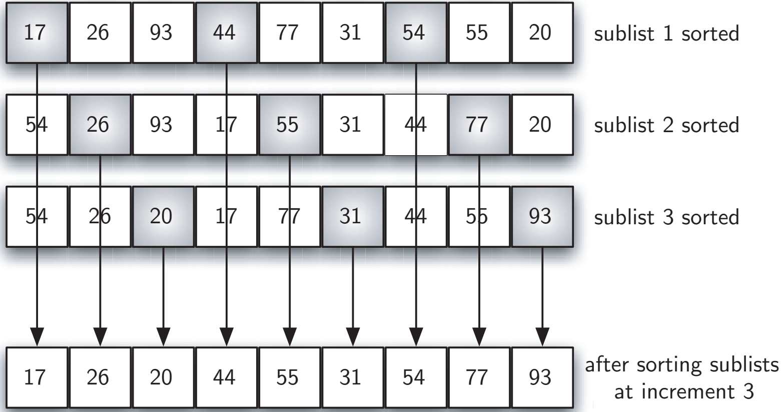


54 26 93 17 77 31 44 55 20 sublist 1

54 26 93 17 77 31 44 55 20 sublist 2

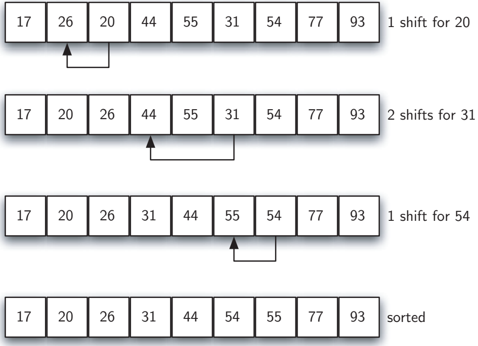
54 26 93 17 77 31 44 55 20 sublist 3

A Shell Sort after Sorting Each Sublist



17	26	93	44	77	31	54	55	20	sublist 1 sorted
54	26	93	17	55	31	44	77	20	sublist 2 sorted
54	26	20	17	77	31	44	55	93	sublist 3 sorted
17	26	20	44	55	31	54	77	93	after sorting sublists at increment 3

ShellSort: A Final Insertion Sort with Increment of 1



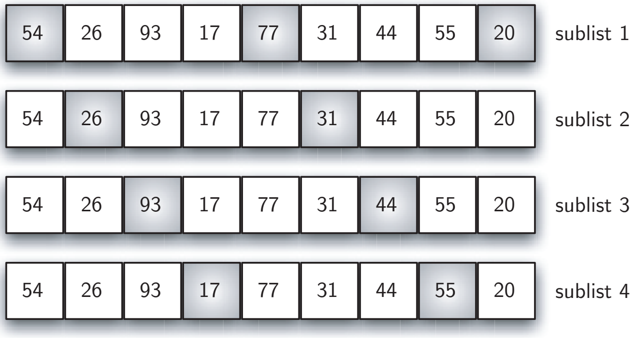
17 26 20 44 55 31 54 77 93 1 shift for 20

17 20 26 44 55 31 54 77 93 2 shifts for 31

17 20 26 31 44 55 54 77 93 1 shift for 54

17 20 26 31 44 54 55 77 93 sorted

Initial Sublists for a Shell Sort



54	26	93	17	**77**	31	44	55	**20**	sublist 1
54	**26**	93	17	77	**31**	44	55	20	sublist 2
54	26	**93**	17	77	31	**44**	55	20	sublist 3
54	26	93	**17**	77	31	44	**55**	20	sublist 4

shellSort I

```
def shellSort(alist):
    sublistcount = len(alist)/2
    while sublistcount > 0:

      for startposition in range(sublistcount):
        gapInsertionSort(alist,startposition,sublistcount)

      print "After increments of size",sublistcount,
                                   "The list is",alist

      sublistcount = sublistcount / 2
```

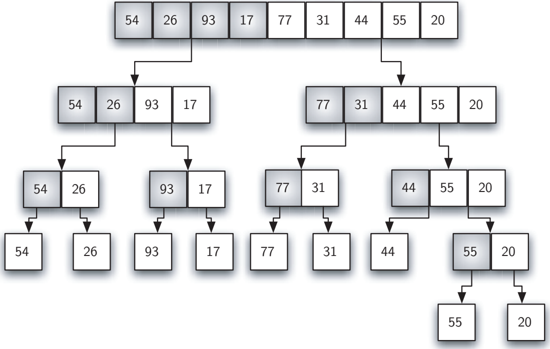
shellSort II

```
def gapInsertionSort(alist,start,gap):
    for i in range(start+gap,len(alist),gap):

        currentvalue = alist[i]
        position = i

        while position>=gap and \
                alist[position-gap]>currentvalue:
            alist[position]=alist[position-gap]
            position = position-gap

        alist[position]=currentvalue
```

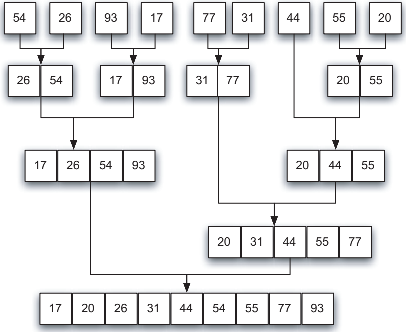
Outline

1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

Splitting and Merging in a Merge Sort



Splitting and Merging in a Merge Sort



mergeSort I

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

        mergeSort(lefthalf)
        mergeSort(righthalf)
```

mergeSort II

```
        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
```

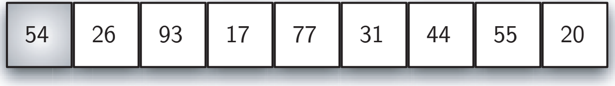
mergeSort III

```

        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
```

Outline

1. What Is Algorithm Analysis?
   - Big-O Notation
   - An Anagram Example
2. Searching
   - The Sequential Search
   - The Binary Search
   - Hashing
3. Sorting
   - The Bubble Sort
   - The Selection Sort
   - The Insertion Sort
   - The Shell Sort
   - The Merge Sort
   - The Quick Sort

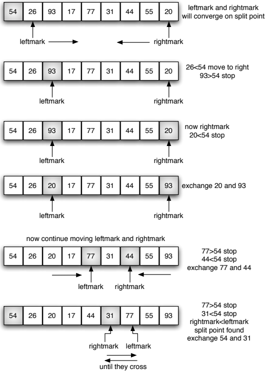
The First Pivot Value for a Quick Sort



54	26	93	17	77	31	44	55	20

54 will be the
first pivot value

Finding the Split Point for 54



54 26 93 17 77 31 44 55 20

leftmark and rightmark will converge on split point

leftmark rightmark

54 26 93 17 77 31 44 55 20

26<54 move to right
93>54 stop

leftmark rightmark

54 26 93 17 77 31 44 55 20

now rightmark
20<54 stop

leftmark rightmark

54 26 20 17 77 31 44 55 93

exchange 20 and 93

leftmark rightmark

now continue moving leftmark and rightmark

54 26 20 17 77 31 44 55 93

77>54 stop
44<54 stop
exchange 77 and 44

leftmark rightmark

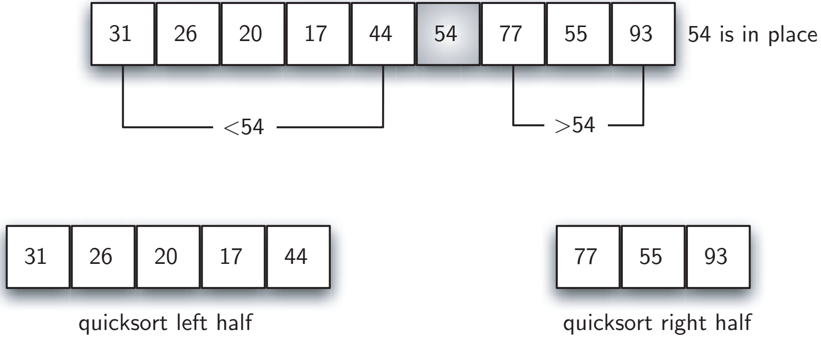
54 26 20 17 44 31 77 55 93

77>54 stop
31<54 stop
rightmark<leftmark
split point found
exchange 54 and 31

rightmark leftmark

until they cross

Completing the Partition Process to Find the Split Point for 54



31 26 20 17 44 54 77 55 93

54 is in place

<54

>54

31 26 20 17 44

quicksort left half

77 55 93

quicksort right half

A Quick Sort I

```
def quickSort(alist):
    quickSortHelper(alist,0,len(alist)-1)

def quickSortHelper(alist,first,last):
    if first<last:

        splitpoint = partition(alist,first,last)

        quickSortHelper(alist,first,splitpoint-1)
        quickSortHelper(alist,splitpoint+1,last)
```

A Quick Sort II

```
def partition(alist,first,last):
    pivotvalue = alist[first]

    leftmark = first+1
    rightmark = last

    done = False
    while not done:
        while leftmark <= rightmark and \
                alist[leftmark] < pivotvalue:
            leftmark = leftmark + 1

        while alist[rightmark] > pivotvalue and \
                rightmark >= leftmark:
            rightmark = rightmark -1
```

A Quick Sort III

```

        if rightmark < leftmark:
            done = True
        else:
            alist[leftmark],alist[rightmark]= \
                         alist[rightmark],alist[leftmark]

    alist[first],alist[rightmark]= \
                   alist[rightmark],alist[first]

    return rightmark
```