The Art of Data Structures

Sorting
Agenda

• To be able to explain and implement various sorting algorithms
  • Bubble
  • Selection
  • Insertion
  • Shell
  • Merge
  • Quick
Sorting
Sorting

- This is the process of organizing data in some particular order
  - Numbers, increasing order
  - Words, alphabetically
  - etc
- Some algorithms benefit with pre-sorted data, e.g. binary search
Sorting

• Sorting is an important area of computer science
• Many sorting algorithms have been developed, and analyzed
• Sorting can take significant time, and is related to the number of items to process
Sorting

• Sorting requires two main operations:
  • **Comparisons** of items to see if they are out of order; comparisons will be an important metric
  • **Exchange** of items can be a costly operation, and also an important metric
Bubble Sort
Bubble Sort

bubble_sort: 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 93 77 31 44 55 20

Exchange

26 54 17 93 77 31 44 55 20

Exchange

26 54 17 93 77 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
Bubble Sort

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.
def bubble_sort(alist):
    for passnum in range(len(alist)-1, 0, -1):
        for i in range(passnum):
            if alist[i] > alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
def bubble_sorted(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
                temp = alist[i]
                alist[i] = alist[i + 1]
                alist[i + 1] = temp
        passnum = passnum - 1
Selection Sort
Selection Sort
def selection_sort(alist):
    for fillslot in range(len(alist)-1, 0, -1):
        position_of_max = 0
        for location in range(1, fillslot+1):
            if alist[location] > alist[position_of_max]:
                position_of_max = location
        temp = alist[fillslot]
        alist[fillslot] = alist[position_of_max]
        alist[position_of_max] = temp
Insertion Sort
## Insertion Sort

<table>
<thead>
<tr>
<th>54</th>
<th>26</th>
<th>93</th>
<th>17</th>
<th>77</th>
<th>31</th>
<th>44</th>
<th>55</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>54</td>
<td>93</td>
<td>17</td>
<td>77</td>
<td>31</td>
<td>44</td>
<td>55</td>
<td>20</td>
</tr>
<tr>
<td>26</td>
<td>54</td>
<td>93</td>
<td>17</td>
<td>77</td>
<td>31</td>
<td>44</td>
<td>55</td>
<td>20</td>
</tr>
<tr>
<td>17</td>
<td>26</td>
<td>54</td>
<td>93</td>
<td>77</td>
<td>31</td>
<td>44</td>
<td>55</td>
<td>20</td>
</tr>
<tr>
<td>17</td>
<td>26</td>
<td>54</td>
<td>77</td>
<td>93</td>
<td>31</td>
<td>44</td>
<td>55</td>
<td>20</td>
</tr>
<tr>
<td>17</td>
<td>26</td>
<td>31</td>
<td>54</td>
<td>77</td>
<td>93</td>
<td>44</td>
<td>55</td>
<td>20</td>
</tr>
<tr>
<td>17</td>
<td>26</td>
<td>31</td>
<td>44</td>
<td>54</td>
<td>77</td>
<td>93</td>
<td>55</td>
<td>20</td>
</tr>
<tr>
<td>17</td>
<td>26</td>
<td>31</td>
<td>44</td>
<td>54</td>
<td>55</td>
<td>77</td>
<td>93</td>
<td>20</td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>26</td>
<td>31</td>
<td>44</td>
<td>54</td>
<td>55</td>
<td>77</td>
<td>93</td>
</tr>
</tbody>
</table>

Assume 54 is a sorted list of 1 item

- inserted 26
- inserted 93
- inserted 17
- inserted 77
- inserted 31
- inserted 44
- inserted 55
- inserted 20
Insertion Sort

insertion_sort: The Fifth Pass

17 26 54 77 93 31 44 55 20

Need to insert 31 back into the sorted list

93 > 31 so shift it to the right

77 > 31 so shift it to the right

54 > 31 so shift it to the right

26 < 31 so insert 31 in this position
def insertion_sort(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
Shell Sort

With Increments of Three

sublist 1

sublist 2

sublist 3
Shell Sort

After Sorting Each Sublist

sublist 1 sorted
sublist 2 sorted
sublist 3 sorted
after sorting sublists at increment 3
Shell Sort

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
Shell Sort

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
```
Shell Sort

Implementation

def shell_sort(alist):
    sublistcount = len(alist)//2
    while sublistcount > 0:
        for startposition in range(sublistcount):
            gap_insertion_sort(alist, startposition, sublistcount)

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

        sublistcount = sublistcount // 2
Shell Sort

Implementation

def gap_insertion_sort(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
Merge Sort
Merge Sort
Splitting and Merging
Merge Sort

Splitting and Merging

Diagram of Merge Sort process.
Merge Sort

Implementation

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

        merge_sort(lefthalf)
        merge_sort(righthalf)

        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)
Quick Sort
Quick Sort
The First Pivot Value

54 26 93 17 77 31 44 55 20

54 will be the first pivot value
Quick Sort

Finding the Split Point for 54

leftmark and rightmark will converge on split point

26<54 move to right
93>54 stop

now rightmark
20<54 stop

exchange 20 and 93

now continue moving leftmark and rightmark

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

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

until they cross
Quick Sort
Completing the Partition Process to Find the Split Point for 54
Quick Sort

Implementation

def quick_sort(alist):
    quick_sort_helper(alist, 0, len(alist)-1)

def quick_sort_helper(alist, first, last):
    if first<last:
        splitpoint = partition(alist,first,last)
        quick_sort_helper(alist, first, splitpoint-1)
        quick_sort_helper(alist, splitpoint+1, last)
Quick Sort

Implementation (cont.)

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

    # Continued on next slide...
if rightmark < leftmark:
    done = True
else:
    temp = alist[leftmark]
    alist[leftmark] = alist[rightmark]
    alist[rightmark] = temp

temp = alist[first]
alist[first] = alist[rightmark]
alist[rightmark] = temp

return rightmark
Analysis
Analysis

- A bubble sort, a selection sort, and an insertion sort are $O(n^2)$ algorithms.
- A shell sort improves on the insertion sort by sorting incremental sub-lists.
  - It falls between $O(n)$ and $O(n^2)$.
- A merge sort is $O(n \log n)$, but requires additional space for the merging process.
Analysis

• A quick sort is $O(n \log n)$, but may degrade to $O(n^2)$ if the split points are not near the middle of the list
• It does not require additional space
Questions?