Lecture 23: More Algorithms for Sorting

CS 1110
Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
Announcements

Next Tuesday:
- Lecture is a review session.
- There will be no post-lecture office hours.

Course Staff also hosting additional review sessions (possibly during study days).
Announcements forthcoming.
Search Algorithms

Recall from last lecture:

• Searching for data is a common task
  ▪ Linear search: on the order of $n$
    • input doubles? $\rightarrow$ work doubles!
  ▪ Binary search: on the order of $\log_2 n$
    • input doubles? $\rightarrow$ work increases by just 1 unit!
    • BUT data needs to be sorted...

• **Sorting** data now suddenly interesting...
Sorting Algorithms

• Sorting data is a common task
  ▪ **Insertion sort**: on the order of \( n^2 \)
    • input doubles? \( \rightarrow \) work **quadruples**! (yikes)

• Today's topic:
  ▪ **Merge sort**: *can we do better than Insertion Sort?*
Which algorithm does Python’s `sort` use?

- Recursive algorithm that runs much faster than insertion sort for the same size list (when the size is big)!
- A variant of an algorithm called “merge sort”
- Based on the idea that sorting is hard, but “merging” two already sorted lists is easy.
Merge sort: Motivation

Since merging is easier than sorting, if I had two helpers, I’d…

• Give each helper half the array to sort
• Then I get back their sorted subarrays and **merge** them.

What if those two helpers each had two sub-helpers?

And the sub-helpers each had two sub-sub-helpers? And…
Subdivide the sorting task
Subdivide again
And again
And one last time
Now merge
And merge again
And again
And one last time

A B C D E F G H J K L M N P Q R

A B E G H K M Q  C D F J L N P R
Done!

A B C D E F G H J K L M N P Q R
def mergeSort(li):
    """Sort list li using Merge Sort"""
    if len(li) > 1:
        # Divide into two parts
        mid = len(li)//2
        left = li[:mid]
        right = li[mid:]

        # Recursive calls
        mergeSort(left)
        mergeSort(right)

        # Merge left & right back to li
    
    # base case does nothing!
    # a list with len 0 or 1 is sorted!
The central sub-problem is the **merging** of two sorted lists into one single sorted list.

**Approach:**
keep comparing the smallest element of first list with smallest element of second list.
How to Merge

as long as both x and y have unprocessed elements

x[i] <= y[j] ?
Yes!

i = 0
j = 0
k = 0
How to Merge

as long as both x and y have unprocessed elements

x[i] <= y[j] ?

Yes!
copy x[i] to z
How to Merge

as long as both x and y have unprocessed elements

x[i] <= y[j]?

No!
How to Merge

as long as both x and y have unprocessed elements

\[ x[i] \leq y[j] \text{?} \]

No!

copy \( y[j] \) to \( z \)

---

\( i \) | 1
\( j \) | 0
\( k \) | 1
How to Merge

as long as both $x$ and $y$
have unprocessed elements

$x[i] \leq y[j]$ ?

Yes!
How to Merge

as long as both x and y have unprocessed elements

x[i] <= y[j]?

Yes!

copy x[i] to z
How to Merge

as long as both x and y have unprocessed elements

\[ x[i] \leq y[j] \, ? \]

Yes!
How to Merge

as long as both x and y have unprocessed elements

x[i] <= y[j] ?

Yes!

copy x[i] to z
How to Merge

as long as both x and y have unprocessed elements

x[i] \leq y[j]?

No!

i

j

k
How to Merge

as long as both x and y have unprocessed elements

x[i] <= y[j]?

No!

copy y[j] to z

\[ i \]
\[ j \]
\[ k \]
How to Merge

as long as both x and y have unprocessed elements

x[i] <= y[j] ?

Yes!

i

j

k
How to Merge

as long as both x and y have unprocessed elements

x[i] <= y[j] ?

Yes!

copy x[i] to z

z

12 15 33 35 42 45

0 1 2 3 4 5 6 7 8

i j k
How to Merge

as long as both x and y have unprocessed elements

x

<table>
<thead>
<tr>
<th>12</th>
<th>33</th>
<th>35</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

y

<table>
<thead>
<tr>
<th>15</th>
<th>42</th>
<th>55</th>
<th>65</th>
<th>75</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

z

<table>
<thead>
<tr>
<th>12</th>
<th>15</th>
<th>33</th>
<th>35</th>
<th>42</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

i

j

k

i 4

j 2

k 6
How to Merge

as long as y has unprocessed elements

copy y[j] to z

x: 12 33 35 45
0 1 2 3

y: 15 42 55 65 75
0 1 2 3 4

z: 12 15 33 35 42 45 55
0 1 2 3 4 5 6

i: 4
j: 2
k: 6
How to Merge

as long as \( y \) has unprocessed elements
How to Merge

as long as y has unprocessed elements

copy y[j] to z
How to Merge

as long as y has unprocessed elements

```plaintext
How to Merge

x: 12 33 35 45
y: 15 42 55 65 75
z: 12 15 33 35 42 45 55 65

i: 4
j: 4
k: 8
```

The diagram shows how to merge lists `x`, `y`, and `z` by comparing elements at indices `i`, `j`, and `k` respectively.
How to Merge

as long as y has unprocessed elements

copy y[j] to z

x

\[12\ 33\ 35\ 45\]

0 1 2 3

y

\[15\ 42\ 55\ 65\ 75\]

0 1 2 3 4

z

[12 15 33 35 42 45 55 65 75]

0 1 2 3 4 5 6 7 8

i 4

j 4

k 8
How to Merge

as long as y has unprocessed elements

How to Merge

as long as y has unprocessed elements

How to Merge

as long as y has unprocessed elements
def merge(x, y, z):
    # Given: sorted lists x and y
    # list z, has the combined length of x and y...
    nx = len(x); ny = len(y)

    i = 0; j = 0; k = 0;
    while i<nx and j<ny:

        # Deal with remaining values in x or y
def merge(x, y, z):
    # Given: sorted lists x and y
    # list z, has the combined length of x and y...
    nx = len(x); ny = len(y)

    i = 0; j = 0; k = 0;
    while i<nx and j<ny:
        if x[i] <= y[j]:
            z[k]= x[i];  i=i+1
        else:
            z[k]= y[j];  j=j+1
            k=k+1

    # Deal with remaining values in x or y
def merge(x, y, z):
    # Given: sorted lists x and y
    # list z, has the combined length of x and y...
    nx = len(x); ny = len(y)

    i = 0; j = 0; k = 0;
    while i<nx and j<ny:
        if x[i] <= y[j]:
            z[k]= x[i];  i=i+1
        else:
            z[k]= y[j];  j=j+1
        k=k+1
    # Deal with remaining values in x or y
    while i<nx:  # copy any remaining x-values
        z[k]= x[i];  i=i+1;  k=k+1

    while j<ny:  # copy any remaining y-values
        z[k]= y[j];  j=j+1;  k=k+1
def mergeSort(li):
    """Sort list li using Merge Sort"""

    if len(li) > 1:
        # Divide into two parts
        mid = len(li)/2
        left = li[:mid]
        right = li[mid:]

        # Recursive calls
        mergeSort(left)
        mergeSort(right)

        # Merge left & right back to li
        merge(left, right, li)
Sorting data is a common task
  - **Insertion sort**: on the order of \( n^2 \)
    - input doubles? \( \rightarrow \) work **quadruples!** (yikes)

**Today's topic:**
  - **Merge sort**: *did we do better than Insertion Sort?*

work = one comparison

*How many comparisons do we make?*
Merge sort:
\[ \sim \log_2(n) \text{ “levels”} \times \sim n \text{ comparisons each level} \]
• Sorting data is a common task
  ▪ **Insertion sort**: on the order of $n^2$
    • input doubles? → work quadruples! (yikes)
  ▪ **Merge sort**: on the order of $n \cdot \log_2(n)$

**Should we always use merge sort then?**

*Python's sort actually combines merge and insertion sort!*

For fun, check out the visualizations:
[https://www.youtube.com/watch?v=xxcpvCGrCBc](https://www.youtube.com/watch?v=xxcpvCGrCBc)
[https://www.youtube.com/watch?v=ZRPoEKHXTJg](https://www.youtube.com/watch?v=ZRPoEKHXTJg)