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]

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

\[\text{ABEHJKLMQ}\quad\text{CDFJLNPR}\]
\[\text{EJKLMABKO}\quad\text{DFLP}\text{ CJNR}\]

Done!

\[\text{ABCDEFLGJKLMNPR}\]

And one last time

\[\text{ABEHJKLMQ}\quad\text{CDFJLNPR}\]

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

\[\begin{array}{cccccc}
12 & 33 & 35 & 45 \\
15 & 42 & 55 & 65 & 75
\end{array}\]

Approach:
keep comparing the smallest element of first list with smallest element of second list.

\[\begin{array}{cccccc}
12 & 15 & 33 & 35 & 42 & 45 & 55 & 65 & 75
\end{array}\]

How to Merge
as long as both x and y have unprocessed elements
\[x[i] \leq y[j] \Rightarrow i, j \Rightarrow \text{Yes!}\]
How to Merge

as long as both x and y have unprocessed elements

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

Yes!

copy \( x[i] \) to z

No!

copy \( y[j] \) to z

z

0 1 2 3 4 5 6 7 8

i 0

j 0

k 0

x

12 33 35 45
0 1 2 3

y

15 42 55 65 75
0 1 2 3 4

k 0

x

12 33 35 45
0 1 2 3

y

15 42 55 65 75
0 1 2 3 4

k 1

x

12 33 35 45
0 1 2 3

y

15 42 55 65 75
0 1 2 3 4

k 2

x

12 33 35 45
0 1 2 3

y

15 42 55 65 75
0 1 2 3 4

k 3
How to Merge

as long as both \( x \) and \( y \) have unprocessed elements

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

Yes!

copy \( x[i] \) to \( z \)

No!

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

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>
<th>55</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

k

<table>
<thead>
<tr>
<th>6</th>
</tr>
</thead>
</table>

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

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>
<th>55</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

k

| 7  |

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

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>
<th>55</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

k

| 8  |

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

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>
<th>55</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

k

| 9  |
```python
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
        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 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: did we do better than Insertion Sort?

work = one comparison

How many comparisons do we make?
```

Merge sort: 
\(~ \log_2(n) \) "levels" \( \sim n \) comparisons each level

```
Sorting Algorithms

- Sorting data is a common task
  - **Insertion sort**: on the order of $n^2$
    - input doubles? $\Rightarrow$ 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=ZRPoEKHXTJg