Algorithms and Data Structures for Data Science
Recursion and Sorting

CS 277
Brad Solomon

September 29, 2021
mp_train due today!

Optional extra credit will be worth extra credit, currently 0 points

Can request a 24-hour extension policy

https://forms.gle/1KreTgcfJb1fXQeQ6

Request one-on-one office hours by emailing with a list of times
mp_sort released today

Due October 13th

A ‘toy’ version of a complete data science project
Informal Early Feedback released today

Form available on Moodle (along with lab surveys)

Feedback form open until October 13th

If >50% class participates, everyone gets 10 bonus points
Reminder: Sign up for exam 1!

Sign up at: [https://cbtf.engr.illinois.edu/sched](https://cbtf.engr.illinois.edu/sched)

Email DRES accommodations: [http://cbtf@illinois.edu/](http://cbtf@illinois.edu/)

Exam Locations: 057 Grainger Library; L520 Digital Comp Lab
Programming questions on exam:

Score: 0/2 (0%)

Test Results

[X] [0/1] Testing provided examples

Max points: 1

Earned points: 0

Message:

'Peak count for [1]: ' is inaccurate
'Peak count for [1, 2, 3, 2]: ' is inaccurate
'Peak count for [2, 1, 0, 1]: ' looks good
'Peak count for [2, 2]: ' is inaccurate

[X] [0/1] Testing other examples...
Learning Objectives

Review sorting fundamentals

Introduce recursion

Analyze recursive strategies to list sorting
The Sorting Problem

Given a collection of objects, $C$, with comparable values, order the objects such that $\forall x \in C$, $x_i \leq x_{i+1}$

Input: 8 4 3 1 2 5 6 9 0 7

Output: 0 1 2 3 4 5 6 7 8 9
Optimal Sorting

**Claim:** Any deterministic comparison-based sorting algorithm must perform $O(n \log n)$ comparisons to sort $n$ objects.
Selection vs Insertion Sort

**Selection Sort**
```python
def selectionSort(inList):
    n = len(inList)
    for i in range(n):
        mindex = i
        for j in range(i+1, n):
            if inList[j] < inList[mindex]:
                mindex = j
        inList[i], inList[mindex] = inList[mindex], inList[i]
```

**Insertion Sort**
```python
def insertionSort(inList):
    n = len(inList)
    for i in range(1, n):
        val = inList[i]
        j = i - 1
        while j >= 0 and val < inList[j]:
            inList[j+1]=inList[j]
            j -= 1
        inList[j+1]=val
```

**Time:**

**Space:**
Divide and Conquer Algorithms

Recursively break a problem into sub-problems until the problems become simple enough to solve directly.
Recursion

*The process by which a function calls itself directly or indirectly is called recursion.*
Recursive For Loop

```python
for i in range(5):
    print(i)
```

```python
def recursiveFor(n):
    if n == 0:
        print(n)
        return
    recursiveFor(n-1)
    print(n)
```
Recursion

**Base Case:** What is the smallest sub-problem? What is the trivial solution?

**Recursive Step:** How can I systematically reduce my problem to an easier one?

**Combining:** How can I build my solution from recursive pieces?
Recursive findMax

Base Case:

Recursive Step:

Combining:
Recursive Fibonacci

\[ \text{Fib}(n) = \text{Fib}(n - 1) + \text{Fib}(n - 2), \quad n > 1 \]

Base Case:

Recursive Step:

Combining:
Recursive Array Sorting

Base Case:

Recursive Step:

Combining:
Merging two sorted lists

Merging two sorted arrays can be done in $O(n + m)$ time
Merging two sorted lists

Merging two sorted arrays can be done in $O(n + m)$ time

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Merging two sorted lists

Let $i = \text{index for } L_1$, $j = \text{index for } L_2$, $k = \text{index for } L_m$

Case 1: Both lists contain unmerged values
Merging two sorted lists

Let $i = \text{index for } L_1$, $j = \text{index for } L_2$, $k = \text{index for } L_m$

**Case 2:** All values in $L_1$ have been merged
Merging two sorted lists

Let $i = \text{index for } L_1$, $j = \text{index for } L_2$, $k = \text{index for } L_m$

**Case 3:** All values in $L_2$ have been merged
Conceptual MergeSort

1) Input list recursively split to “sorted” base case

```
3 2 1 4
```

2) Sorted lists are merged back together

```
3 2 1 4
```

```
1 2 3 4
```
MergeSort

4 3 6 7 1

4 3 6

4 3

4 3

6

7 1

7 1

7 1
MergeSort Efficiency

1) Input list recursively split to “sorted” base case
MergeSort Efficiency

1) Input list recursively split to “sorted” base case

![Diagram showing a list split into two parts, then merging them back together.]

Time:

Space:
MergeSort Efficiency

2) Sorted lists are merged back together

1 3 8 5 0 7 6 2

1 3 5 8 0 7 2 6

1 3 5 8 0 2 6 7

0 1 2 3 4 5 6 7
MergeSort Efficiency

2) Sorted lists are merged back together

Time:

Space:
<table>
<thead>
<tr>
<th>Best Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Space:</strong></td>
<td></td>
</tr>
</tbody>
</table>