Algorithms and Data Structures for Data Science

Sorting

CS 277
Brad Solomon

September 27, 2021
Lab_hash Feedback

Average score: 82%

PL average time: 163 minutes

Material of neutral helpfulness to a sizable minority of students

Lab taught learning objectives and universally improved coding confidence

There was a problem with double_hash but it was resolved immediately
Learning Objectives

Motivate the need for sorting

Explore iterative solutions to sorting

Introduce recursion
The Sorting Problem

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

\[
\begin{array}{cccccccccccc}
8 & 4 & 3 & 1 & 2 & 5 & 6 & 9 & 0 & 7 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]
Sorting leads to better visualization
Sorting is a fundamental problem in CS

Many algorithms begin with or include a sorting step

Fundamental sorting algorithms are great for mastering concepts

Sorting algorithms are a classic introduction to algorithms
Optimizing sort is an ongoing challenge

**GraySort:** Sort rate (TBs / minute) achieved while sorting a very large amount of data (currently 100 TB minimum).

**CloudSort:** Minimum cost (Dollars) for sorting a very large amount of data on a public cloud. (currently 100 TB).

**MinuteSort:** Amount of data that can be sorted in 60 seconds or less.

**TeraByeSort:** Elapsed time to sort 1 TB of data

Competition details: [http://sortbenchmark.org/](http://sortbenchmark.org/)
SelectionSort

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]
InsertionSort

1. Divide array into two parts
2. Insert the first unsorted item into the sorted position
3. Repeat until all items are sorted
InsertionSort “Insert”

1 2 4 5 7 3 8

1 2 4 5 3 7 8

1 2 4 3 5 7 8

1 2 3 4 5 7 8
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
Selection vs InsertionSort

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Selection vs InsertionSort

5 4 3 2 1

5 4 3 2 1
Selection vs InsertionSort

```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]

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
```
**Claim:** Any deterministic comparison-based sorting algorithm must perform $O(n \log n)$ comparisons to sort $n$ objects.
Divide and Conquer Algorithms

Recursively break a problem into sub-problems until the 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(n+1):
    print(i)
```

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

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

Given a list, sum all the items in the list using recursion

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

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

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

Given a list, sum all the items in the list *using recursion*

```
8 4 2 6 5
```
Recursive findMax

| 8 | 4 | 3 | 1 | 2 | 5 | 6 | 9 | 0 | 7 |

**Base Case:**

**Recursive Step:**

**Combining:**
Recursive Fibonacci

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

Base Case:

Recursive Step:

Combining:
Recursive List Partitioning

Using all elements in a list, can we make two lists which have equal sums?

\[
\begin{array}{cccccc}
6 & 5 & 4 & 2 & 7 \\
1 & 1 & 1 & 1 & 1 & 1 \\
2 & 3 & 3 & 3 & 3 & 1 \\
\end{array}
\]
Recursive List Partitioning

Using all elements in a list, can we make two lists which have equal sums?

Base Case:
Recursive List Partitioning

Using all elements in a list, can we make two lists which have equal sums?

Recursive Step:
Recursive List Partitioning

Using all elements in a list, can we make two lists which have equal sums?

(New) Base Case:
Recursive List Partitioning

Using all elements in a list, can we make two lists which have equal sums?

**Combination Step:**
Recursive List Partitioning

Using all elements in a list, can we make two lists which have equal sums?

4 3 1
Using all elements in a list, can we make two lists which have equal sums?

Input

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4, 3, 1]</td>
<td>([], [])</td>
</tr>
<tr>
<td>[3, 1]</td>
<td>([4], [])</td>
</tr>
<tr>
<td>[1]</td>
<td>([3, 4], [])</td>
</tr>
<tr>
<td>[]</td>
<td>([1, 3, 4], [])</td>
</tr>
</tbody>
</table>
Recursive Array Sorting

Base Case:

Recursive Step:

Combining: