Lecture 26: More on Algorithms for Sorting

CS 1110
Introduction to Computing Using Python

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

• Discussion sections this week
  ▪ First 10 minutes dedicated to getting your started on A6
  ▪ Remaining time is office hour for your A6/Prelim 2 questions

• Final Exam on May 21\textsuperscript{st} 1:30-4pm. Your assigned exam session (in-person or online) is shown in CMS. Submit a \textit{“regrade request” in CMS by May 12} if you have a \textit{legitimate} reason for requesting a change. If you have an exceptional circumstance for switching from in-person to online, you must upload to CMS your college’s approval of your modality change.
More Announcements

- **A6** due on Friday
  - Remember academic integrity
- **Expected release dates of solutions and feedback**
  - **A5** solutions: Wed May 12
  - **A4** grades and feedback: Thurs May 13
  - **A6** solutions: Tues May 18
  - **A5** grades and feedback: Thurs May 20
  - **Final exam** grades and feedback: Tues May 25
  - **A6** grades and feedback: Fri May 28
Algorithms for Sorting

• Well known algorithms
  ▪ focus on reviewing programming constructs (\textit{while} loop) and analysis
  ▪ will not use built-in methods such as \texttt{sort}, \texttt{index}, \texttt{insert}, etc.

• Today we’ll discuss \texttt{merge sort} and compare it to \texttt{insertion sort}, which we discussed last lecture

• More on the topic in next course, CS 2110!
The Insertion Process of Insertion Sort

- Given a sorted list $x$, insert a number $y$ such that the result is sorted
- Sorted: arranged in ascending (small to big) order

We’ll call this process a “push down,” as in push a value down until it is in its sorted position
Algorithm Complexity

- Count the number of comparisons needed
- In the worst case, need $i$ comparisons to push down an element in a sorted segment with $i$ elements.
How much work is a push down?

push down a “big” value

push down a “small” value

This push down takes 2 comparisons

This push down takes 4 comparisons.

Worst case scenario: \( n \) comparisons needed to push down into a length \( n \) sorted segment.
Algorithm Complexity (Q)

def swap(b, h, k):
    :

def push_down(b, k):
    while k > 0 and b[k-1] > b[k]:
        swap(b, k-1, k)
        k = k-1

def insertion_sort(b):
    for i in range(1, len(b)):
        push_down(b, i)

Count (approximately) the number of comparisons needed to sort a list of length n

A. ~ 1 comparison
B. ~ n comparisons
C. ~ n^2 comparisons
D. ~ n^3 comparisons
E. I don’t know
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 have 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

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

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

The central sub-problem is the **merging** of two sorted lists into one single sorted list

```
12  33  35  45
15  42  55  65  75
12  15  33  35  42  45  55  65  75
```
Merge

ix<4 and iy<5 \( \Rightarrow \) \( x(ix) \leq y(iy) \) YES
Merge

\[ ix < 4 \text{ and } iy < 5 \Rightarrow x(ix) \leq y(iy) \]  NO
ix < 4 and iy < 5 \implies x(ix) \leq y(iy) \quad \text{YES}
Merge

\[
x = \begin{bmatrix} 12 & 33 & 35 & 45 \end{bmatrix}
\]

\[
y = \begin{bmatrix} 15 & 42 & 55 & 65 & 75 \end{bmatrix}
\]

\[
z = \begin{bmatrix} 12 & 15 & 33 & 35 & \ldots & \ldots & \ldots \end{bmatrix}
\]

\[
ix < 4 \text{ and } iy < 5 \Rightarrow x(ix) \leq y(iy) \quad \text{YES}
\]
$\text{Merge}$

$x$

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
12 & 33 & 35 & 45 \\
\end{array}
\]

$y$

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
15 & 42 & 55 & 65 & 75 \\
\end{array}
\]

$z$

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
12 & 15 & 33 & 35 & 42 & & & & \\
\end{array}
\]

$ix<4$ and $iy<5 \rightarrow x(ix) \leq y(iy)$ \ NO
Merge

ix < 4 and iy < 5 \implies x(i) \leq y(i) \quad \text{YES}
ix at 4 → take y(iy)
iy < 5 \rightarrow \text{take } y(iy)
Merge

\[\text{ix} \quad 4\]

\[\text{iy} \quad 4\]

\[\text{iz} \quad 8\]

\[\text{iy} < 5 \Rightarrow \text{take } y(iy)\]
# Given lists x and y and list z, which has
# the combined length of x and y...

nx = len(x); ny = len(y)

ix = 0; iy = 0; iz = 0;

while ix<nx and iy<ny:
    if x[ix] <= y[iy]:
        z[iz]= x[ix];  ix=ix+1
    else:
        z[iz]= y[iy];  iy=iy+1
    iz=iz+1

while ix<nx  # copy any remaining x-values
    z[iz]= x[ix];  ix=ix+1;  iz=iz+1

while iy<ny  # copy any remaining y-values
    z[iz]= y[iy];  iy=iy+1;  iz=iz+1
How do merge sort and insertion sort compare?

• Insertion sort: (worst case) makes $i$ comparisons to insert an element in a sorted array of $i$ elements. For an array of length $n$:

__________________________ for big $n$

• Merge sort: _____________________
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
        ...

    All the comparisons between list elements are done during merge
# Given lists x and y and list z, which has the combined length of x and y...

```
nx = len(x); ny = len(y)

ix = 0; iy = 0; iz = 0;
while ix<nx and iy<ny
    if x[ix] <= y[iy]:
        z[iz] = x[ix]; ix=ix+1
    else:
        z[iz] = y[iy]; iy=iy+1
    iz=iz+1

while ix<nx  # copy any remaining x-values
    z[iz] = x[ix]; ix=ix+1; iz=iz+1

while iy<ny  # copy any remaining y-values
    z[iz] = y[iy]; iy=iy+1; iz=iz+1
```
Merge – best case scenario

x

2 3 5 14

y

15 42 55 65

z
Merge – worst case scenario

X: 12, 23, 45, 64

Y: 15, 42, 55, 65

Z: Need to do n-1 comparisons where n is total number of elements merged.
Merge sort: about $\log_2(n)$ “levels”; about $n$ comparisons each level
How do merge sort and insertion sort compare?

• Insertion sort: (worst case) makes $i$ comparisons to insert an element in a sorted array of $i$ elements. For an array of length $n$:
  
  $1+2+\ldots+(n-1) = \frac{n(n-1)}{2}$, say $n^2$ for big $n$

• Merge sort: $n \cdot \log_2(n)$ comparisons

• Should we always use merge sort then? Python actually uses a variant that combines merge sort and insertion sort!