Sorting Algorithms

CPE 103

Theresa Migler-VonDollen

Sorting

Sorting

Input: A list of numbers, $a_1, a_2, a_3, \ldots a_n$.
Goal: Return a list of the same numbers sorted in increasing order.

Example:
Given: $4, 907, 34, 18, 42, 36, 71, 34, 16$
Return: $4, 16, 18, 34, 34, 36, 42, 71, 907$

Selection Sort - Review from 102

Sorting

Input: A list of numbers, $a_1, a_2, a_3, \ldots a_n$.
Goal: Return a list of the same numbers sorted in increasing order.

SelectionSort($A[0, \ldots, n-1]$)

Input: A list of unsorted numbers $A[0, \ldots, n-1]$
Output: The same list sorted in increasing order

```
for i = 0, ..., n − 1 do
    Find min of A[i, ..., n − 1].
    Suppose that the min occurs at position j.
    Swap A[i] with A[j].
```

Selection Sort - Analysis

- Is it correct?

Lemma

Upon completion of SelectionSort, *for any* $i \in \{1, \ldots, n-1\}$, $A[i-1] \leq A[i]$.

- Running Time:
  - $T(n) = T(n-1) + O(n) = O(n^2)$

Bubble Sort

Sorting

Input: A list of numbers, $a_1, a_2, a_3, \ldots a_n$.
Goal: Return a list of the same numbers sorted in increasing order.

BubbleSort($A[0, \ldots, n-1]$)

Input: A list of unsorted numbers $A[0, \ldots, n-1]$
Output: The same list sorted in increasing order

for $i = 0, \ldots, n-1$ **do**
    for $j = 0, \ldots, n-1$ **do**
        if $A[j] > A[j+1]$ **then**
            Swap $A[j]$ and $A[j+1]$

Bubble Sort - Analysis

- Is it correct?

Lemma

Upon completion of BubbleSort, *for any* $i \in \{1, \ldots, n-1\}$, $A[i-1] \leq A[i]$.

- Running Time:
  - There are two for loops, each of size n.
  - Step 4 is constant time.
  - Therefore, the running time is $O(n^2)$.

How does BubbleSort perform on already sorted lists?

Insertion Sort

Sorting

Input: A list of numbers, $a_1, a_2, a_3, \ldots a_n$.
Goal: Return a list of the same numbers sorted in increasing order.

InsertionSort($A[0, \ldots, n-1]$)

Input: A list of unsorted numbers $A[0, \ldots, n-1]$
Output: The same list sorted in increasing order

```
for i = 1, ..., n − 1 do
    j = i
    while j > 0 and A[j − 1] > A[j] do
        Swap A[j] and A[j − 1]
        j = j − 1
```

Insertion Sort - Analysis

- Is it correct?

Lemma

Upon completion of InsertionSort*, for any* $i \in \{1, \ldots, n-1\}$*,* $A[i-1] \leq A[i]$*.*

- Running Time:
  - There is one for loop of size n.
  - At most the while loop will perform n swaps.
  - Therefore, the running time is $O(n^2)$.

Selection Sort vs Bubble Sort vs Insertion Sort

The three algorithms are asymptotically equivalent.

- However, in practice InsertionSort is much faster than the others.

Which algorithms are *online* - can sort lists as they receive them?

- SelectionSort requires the whole input at the beginning.
- InsertionSort is online.
- What about BubbleSort?

Divide and Conquer

Divide and Conquer is a strategy that solves a problem by:

1. Breaking the problem into subproblems that are themselves smaller instances of the same type of problem.
2. Recursively solving these subproblems.
3. Appropriately combining their answers.

Merge Sort

Sorting

Input: A list of numbers, $a_1, a_2, a_3, \ldots a_n$.
Goal: Return a list of the same numbers sorted in increasing order.

MergeSort($A[0, \ldots, n-1]$)

Input: A list of unsorted numbers $A[0, \ldots, n-1]$
Output: The same list, sorted in increasing order

if $n \leq 1$ **then**
    return A
else
    return Merge(MergeSort($A[0, \ldots, \lfloor n/2 \rfloor]$),MergeSort($A[\lfloor n/2 \rfloor + 1, \ldots, n-1]$))

Merge Sort

Sorting

Input: A sequence of numbers, $a_1, a_2, a_3, \ldots a_n$.
Goal: Return a list of the same numbers sorted in increasing order.

Merge($x[0, \ldots, k-1]$,$y[0, \ldots, \ell-1]$)

Input: Two sorted lists, $x[0, \ldots, k-1]$ and $y[0, \ldots, \ell-1]$
Output: One sorted list that contains all elements of both lists.

if $x = \emptyset$ **then return** y
if $y = \emptyset$ **then return** x
if $x[0] \leq y[0]$ **then**
    return $x[0] \circ$Merge$(x[1, \ldots, k-1], y[0, \ldots, \ell-1])$
else
    return $y[0] \circ$Merge$(x[0, \ldots, k-1], y[1, \ldots, \ell-1])$

Merge Sort - Correctness

Theorem

Merge *correctly merges two sorted lists.*

Proof.

We will procede by induction on the total size of the lists being merged. (We will prove that Merge correctly merges two sorted lists of total size n.)

- Base Case: $(n = 1)$
  This will only occur if either x or y is empty, and the other list has exactly 1 element.
  - Merge correctly merges the empty list with any other sorted list.
- Inductive Hypothesis: Suppose that Merge correctly merges two sorted lists of total size equal to n.

Merge Sort - Correctness

Theorem

Merge *correctly merges two sorted lists.*

Proof (Cont.)

- Inductive Step: Consider two sorted lists with total size $n+1$.
  - In steps 3 and 4 of the algorithm, Merge correctly places the smallest element at the beginning of the list.
  - Merge then concatenates that element with the Merge of the remaining elements of the two lists.
    - The total size of the remaining two lists is n.
    - By the Inductive Hypothesis, Merge correctly merges the remainder.
- Conclusion: Therefore, by PMI, Merge correctly merges two sorted lists.

□

Merge Sort - Running Time

- What is the running time?
  - $T(n) = 2T(n/2) + O(n) = O(n \log n)$

Quick Sort

Sorting

Input: A list of numbers, $a_1, a_2, a_3, \ldots a_n$.
Goal: Return a list of the same numbers sorted in increasing order.

QuickSort is also a divide and conquer algorithm.
Idea:

- Pick a "pivot point".
  - Picking a good pivot point can greatly affect the running time.
- Break the list into two lists:
  - Those elements less than the pivot element.
  - Those elements greater than the pivot element.
- Recursively sort each of the smaller lists.
- Make one big list: the 'smallers' list, the pivot points, and the 'biggers' list.

Quick Sort

Sorting

Input: A list of numbers, $a_1, a_2, a_3, \dots a_n$.
Goal: Return a list of the same numbers sorted in increasing order.

QuickSort($A[0, \dots, n-1]$, *low*, *high*)

Input: A list of unsorted numbers $A[0, \dots, n-1]$, two integers *high* and *low*
Output: The same list sorted in increasing order

```
if low < high then
    pivotLocation = Partition(A, low, high)
    QuickSort(A, low, pivotLocation)
    QuickSort(A, pivotLocation + 1, high)
```

Quick Sort - Partition

Partition($A, low, high$)

Input: A list of unsorted numbers $A[0, \ldots, n-1]$, two integers *high* and *low*

Output: An integer (the pivot location) and a list partitioned about the pivot.

$pivot = A[low]$
$leftwall = low$
for $i = low + 1, \ldots, high$ **do**
    if $A[i] < pivot$ **then**
        Swap $A[i]$ and $A[leftwall]$
        $leftwall = leftwall + 1$
Swap $A[low]$ with $A[leftwall]$
    return *leftwall*

Quick Sort - Running Time

Average case analysis:

- $O(n \log n)$

Worst case analysis:

- $O(n^2)$