Lecture 03/02/16

Lecturer: Xiaodi Wu

March 2nd, 2016
# Sorting algorithms: comparison-based

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble Sort</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Merge Sort</td>
<td>$O(n \log(n))$</td>
</tr>
<tr>
<td>Quick Sort</td>
<td>$O(n \log(n))$</td>
</tr>
<tr>
<td>Heap Sort</td>
<td>$O(n \log(n))$</td>
</tr>
</tbody>
</table>
Quick & Merge Sort

Divide and Conquer

- Original problem: sort $n$-item sequence $S$. Divide the problems into sub-problems.
Quick & Merge Sort

Divide and Conquer

- Original problem: sort $n$-item sequence $S$. Divide the problems into sub-problems.
- Divide $S$ into $S_1$ and $S_2$. Sort $S_1$ and $S_2$ separately.
Quick & Merge Sort

Divide and Conquer

- Original problem: sort $n$-item sequence $S$. Divide the problems into sub-problems.
- Divide $S$ into $S_1$ and $S_2$. Sort $S_1$ and $S_2$ separately.
- Combine the sorting result of $S_1$ and $S_2$ to get the sorted list for $S$. 

Terminal case: when $|S| = 1, 2$, sort $S$ directly.
Quick & Merge Sort

Divide and Conquer

- Original problem: sort $n$-item sequence $S$. Divide the problems into sub-problems.
- Divide $S$ into $S_1$ and $S_2$. Sort $S_1$ and $S_2$ separately.
- Combine the sorting result of $S_1$ and $S_2$ to get the sorted list for $S$.
- When sort $S_1, S_2$, apply the same procedure recursively.
Quick & Merge Sort

Divide and Conquer

- Original problem: sort $n$-item sequence $S$. Divide the problems into sub-problems.
- Divide $S$ into $S_1$ and $S_2$. Sort $S_1$ and $S_2$ separately.
- Combine the sorting result of $S_1$ and $S_2$ to get the sorted list for $S$.
- When sort $S_1$, $S_2$, apply the same procedure recursively.
- Terminal case: when $|S| = 1, 2$, sort $S$ directly.
Algorithm

- Original problem: sort \( n \)-item sequence \( S \). Divide the problems into sub-problems.
- Let \( S_1 \) be the first half of \( S \) and \( S_2 \) the second half.
Merge Sort

Algorithm

- Original problem: sort $n$-item sequence $S$. Divide the problems into sub-problems.
- Let $S_1$ be the first half of $S$ and $S_2$ the second half.
- Merge two sorted $S_1$ and $S_2$ to get the sorted list for $S$. 
Merge Sort

Algorithm

- Original problem: sort $n$-item sequence $S$. Divide the problems into sub-problems.
- Let $S_1$ be the first half of $S$ and $S_2$ the second half.
- Merge two sorted $S_1$ and $S_2$ to get the sorted list for $S$.
- When sort $S_1$, $S_2$, apply the same procedure recursively.
Merge Sort

Algorithm

- Original problem: sort \( n \)-item sequence \( S \). Divide the problems into sub-problems.
- Let \( S_1 \) be the first half of \( S \) and \( S_2 \) the second half.
- Merge two sorted \( S_1 \) and \( S_2 \) to get the sorted list for \( S \).
- When sort \( S_1, S_2 \), apply the same procedure recursively.
- Terminal case: when \( |S| = 1, 2 \), sort \( S \) directly.
In assignment 3, we asked one problem to merge $k$ sorted sequences into one with $O(n \log k)$ (using heaps).
Merge two sorted sequences

- In assignment 3, we asked one problem to merge $k$ sorted sequences into one with $O(n \log k)$ (using heaps).
- This implies an $O(n)$ algorithm for merging two sorted sequences.
Merge two sorted sequences

- In assignment 3, we asked one problem to merge $k$ sorted sequences into one with $O(n \log k)$ (using heaps).
- This implies an $O(n)$ algorithm for merging two sorted sequences.

Simple solution: given sorted $S_1$ and $S_2$

- One can easily maintain the smaller one of the front of $S_1$ and $S_2$.
- Remove and insert the smaller one into $S$. Update the front of $S_1$ (or $S_2$).
Merge Sort: Divide
Merge Sort: Conquer
Running Time

- Let $T(n)$ denote the time of merge-sort on $n$ items.
Running Time

- Let $T(n)$ denote the time of merge-sort on $n$ items.
- By the divide-and-conquer design, we have

\[ T(n) = 2T(n/2) + O(n), \forall n > 2, \quad T(1) = O(1), \quad T(2) = O(1). \]
Running Time

- Let $T(n)$ denote the time of merge-sort on $n$ items.
- By the divide-and-conquer design, we have

$$T(n) = 2T(n/2) + O(n), \forall n > 2, \quad T(1) = O(1), \quad T(2) = O(1).$$

- In general, one can write down the following relations,

$$T(n/2) = 2T(n/4) + O(n/2)$$
$$T(n/4) = 2T(n/8) + O(n/4)$$
$$\ldots$$
$$T(n/2^i) = 2T(n/2^{i+1}) + O(n/2^i)$$
Thus, we have

\[ T(n) = 2^i T(n/2^i) + O(i \times n). \]

We can choose \( i \) as large as \( \log(n) \). Then

\[ T(n) = 2^{\log n} T(1) + O(n \log n) = O(n \log n). \]
Quick Sort

Algorithm

- Original problem: sort \( n \)-item sequence \( S \). Divide the problems into sub-problems.
- Choose a pivot \( x \in S \), and then let \( L = \{ y \in S | y < x \} \), \( E = \{ y \in S | y = x \} \), \( G = \{ y \in S | y > x \} \)
Quick Sort

Algorithm

- Original problem: sort $n$-item sequence $S$. Divide the problems into sub-problems.
- Choose a pivot $x \in S$, and then let $L = \{y \in S|y < x\}$, $E = \{y \in S|y = x\}$, $G = \{y \in S|y > x\}$
- Recursively apply quick sort to $L, G$. (no need for $E$).
Quick Sort

Algorithm

- Original problem: sort \( n \)-item sequence \( S \). Divide the problems into sub-problems.
- Choose a pivot \( x \in S \), and then let \( L = \{y \in S | y < x\} \), \( E = \{y \in S | y = x\} \), \( G = \{y \in S | y > x\} \)
- Recursively apply quick sort to \( L \), \( G \). (no need for \( E \)).
- Combine the sorted \( L \), \( E \), \( G \). Simply \([L, E, G]\).
Quick Sort

Pivot Choice

- Multiple choices. Could affect the final complexity.

Ideally, hope \( L \), \( G \) have equal sizes. Then choose the median as the pivot.

Find the median: \( O(n) \). Find \( L \), \( G \): also \( O(n) \)

Combine \( L \), \( E \), \( G \)

\( L \), \( E \), \( G \) are already sorted and in the right order. Simply combine them: \( O(1) \).
Quick Sort

Pivot Choice

- Multiple choices. Could affect the final complexity.
- Ideally, hope $L, G$ have equal sizes. Then choose the median as the pivot.
Quick Sort

Pivot Choice

- Multiple choices. Could affect the final complexity.
- Ideally, hope $L, G$ have equal sizes. Then choose the median as the pivot.
- Find the median: $O(n)$. Find $L, G$: also $O(n)$
Quick Sort

Pivot Choice

- Multiple choices. Could affect the final complexity.
- Ideally, hope $L, G$ have equal sizes. Then choose the median as the pivot.
- Find the median: $O(n)$. Find $L, G$: also $O(n)$

Combine $L, E, G$

- $L, E, G$ are already sorted and in the right order. Simply combine them: $O(1)$. 
Quick Sort: Divide

85, 24, 63, 45, 17, 31, 96, 50

24, 45, 17, 31

24, 17

24

45

85, 63, 96

85, 63

85
Quick Sort: Conquer

17, 24, 31, 45, 50, 63, 85, 96

17, 24, 31, 45

17, 24

24

·

45

63, 85, 96

63, 85

85

·
Running Time

- Let $T(n)$ denote the time of merge-sort on $n$ items.
Running Time

- Let $T(n)$ denote the time of merge-sort on $n$ items.
- Assume finding the median $O(n)$, then we have

$$T(n) = 2T(n/2) + O(n), \forall n > 2, T(1) = O(1), T(2) = O(1).$$
Running Time

- Let $T(n)$ denote the time of merge-sort on $n$ items.
- Assume finding the median $O(n)$, then we have
  
  $$T(n) = 2T(n/2) + O(n), \forall n > 2, \ T(1) = O(1), \ T(2) = O(1).$$

- From the above, we have $T(n) = O(n \log n)$. 