Week09: Searching and Sorting
Dr. W. Zhang

CS2123 Data Structures
Searching and Sorting

Dr. Weining Zhang
Department of Computer Science
University of Texas at San Antonio
August 28, 2009

Outline

1 Searching and Sorting
   - Searching
   - Sorting

Tasks of the Week

- Trace the selection, insertion and bubble sorts
- Implement and trace linear and binary search
- Case Study: (review from CS1713) The Search class - Implementing linear and binary search for an array of Comparable
- Case Study: (review from CS1713) The QuadraticSort class - Implementing selection, insertion and bubble sort for an array of Comparable
- Case Study: The FastSort class - Implementing quick sort and merge sort for an array of Comparable
- Trace through the algorithms for quick sort and merge sort
- Analyze the divide and conquer strategy for various sorting algorithms

Searching Algorithms

- Linear Search
  - Use a loop (for-loop or while-loop) to check every element in a list or in an array
  - Can work on any list or array, sorted or unsorted
  - Time complexity is $O(n)$ for list/array of $n$ elements

- Binary Search
  - Can be iterative or recursive
  - Always check the middle element in the remaining list. Each comparison cuts the list into half
  - Can only work on list that is already sorted
  - Time complexity is $O(\log_2 n)$ for list of $n$ elements
### Binary Search Example

**Example**

Search for 6 in following list

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>7</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
</tbody>
</table>

1. Check position 4, since 6 < 12, ignore positions 5 to 8
2. Check position 1, since 4 < 6, ignore position 0
3. Check position 2, since 6 < 7, ignore position 3
4. Since no more position to check, 6 is not in the list

---

### Quadratic Sorting Algorithms

- Quadratic sorting algorithms are all iterative, and have complexity of $O(n^2)$ for list of $n$ elements
- **Selection sort**
  - In each iteration, place the smallest element in the remaining elements to the next position
- **Insertion sort**
  - In each iteration, insert the next element into the sorted list of previous elements
- **Bubble sort**
  - In iteration, pass through the list once and only adjust orders of adjacent elements

---

### Merge Sort Algorithm

- A recursive method uses divide-and-conquer strategy to sort a list of $n$ elements
  - **Base case**: if $n = 1$, return the list
  - **Recursive case**:
    - Split the list into two equal-sized sublists $SL_1$ and $SL_2$ at the index of $\text{mid} = (\text{min} + \text{max})/2$
    - $\text{Mergesort } SL_1$
    - $\text{Mergesort } SL_2$
    - Merge the sorted sublists in one sorted list
  - Time complexity is $O(n \log_2 n)$ for list of $n$ elements

---

### Quick Sort Algorithm

- Another recursive method that uses the divide-and-conquer strategy for sorting
  - **Base case**: if the list is empty, return
  - **Recursive case**
    - Partition the list into two sublists $SL_1$ and $SL_2$ according to a pivot element, so that, every element in $SL_1$ is less and every element in $SL_2$ is larger than the pivot. The whole list looks like $[SL_1, \text{pivot}, SL_2]$
    - $\text{QuickSort } SL_1$
    - $\text{QuickSort } SL_2$
  - Time complexity is $O(n \log_2 n)$ for list of $n$ elements