Quick Sort

Lecture 35
Section 13.3

Robb T. Koether

Hampden-Sydney College

Thu, Apr 16, 2009
Outline

1. The Quick Sort Algorithm
2. Example
3. The `qsort()` Library Function
   - The `compar()` Function
4. Assignment
The Quick Sort

- The Quick Sort is, on average, the fastest known sorting algorithm.
- The run time is of order $n \log n$ (with probability near 100%).
The Quick Sort Algorithm

- Choose an element of the list to be the pivot element and copy it to a temporary location.
  - Choose the first element, or
  - Choose the last element, or
  - Choose the middle element, or
  - Choose a random element, or
  - Choose the median of a small number of elements.
The Quick Sort Algorithm

- The Quick Sort is at its fastest when the pivot is the element that “belongs” in the middle of the list.
- It evenly divides the list into halves.
- The Quick Sort is at its slowest when the pivot is the element that “belongs” at one end of the list.
- All the elements end up on the same side.
The Quick Sort Algorithm

- The list is divided into four regions.
  - Smaller region: The elements determined to be less than the pivot.
  - Equal region: The elements determined to be equal to the pivot.
  - Larger region: The elements determined to be greater than the pivot.
  - NotDone region: The elements that have not yet been compared to the pivot.
The Quick Sort Algorithm

- The regions are arranged from left to right in the order
  - Smaller region
  - Equal region
  - NotDone region
  - Larger region

- Initially, all elements are in the NotDone region.
- Eventually, all elements will be in the other three regions.
The Quick Sort Algorithm

- Establish three indexes.
  - `smallerR` - Index of the rightmost member of the Smaller region.
  - `notDoneL` - Index of the leftmost member of the NotDone region.
  - `largerL` - Index of the leftmost member of the Larger region.
The Quick Sort Algorithm

- **Initial Values**
  - `smallerR` is initialized to 1 less than the lowest subscript in the array.
  - `notDoneL` is initialized to the lowest subscript in the array.
  - `largerL` is initialized to 1 greater than the highest subscript in the array.
The Quick Sort Algorithm

- Compare the element in position \texttt{notDoneL} to the pivot element.
- If it is less than the pivot, then
  - Increment \texttt{smallerR},
  - Swap the elements in the \texttt{notDoneL} and \texttt{smallerR} positions.
  - Increment \texttt{notDoneL}.  
The Quick Sort Algorithm

- If it is equal to the pivot, then increment `notDoneL`.
- If it is greater than the pivot,
  - Decrement `largerL`.
  - Swap the elements in the `notDoneL` and `largerL` positions.
The Quick Sort Algorithm

- Continue until the NotDone region is empty.
- Then apply the process recursively to the regions Smaller and Larger.
Example (The Quick Sort)

- The list to be sorted.

```
| 60 | 70 | 80 | 20 | 40 | 50 | 10 | 30 | 90 |
```

SR  NDL  LL
Example (The Quick Sort)

- Pivot is 40.

```
60 70 80 20 40 50 10 30 90
```

SR  NDL  LL
Example (The Quick Sort)

- Decrement LL; swap 60 and 90.

```
60 70 80 20 40 50 10 30 90
```

- SR
- NDL
- LL
Example

Example (The Quick Sort)

- Decrement LL; swap 90 and 30.

```
90 70 80 20 40 50 10 30 60
LLNDLSR
```

- **pivot**

```
40
```

- **SR**
- **NDL**
- **LL**
Increment SR and NDL.
Example (The Quick Sort)

- Decrement LL; swap 70 and 10.
Example (The Quick Sort)

- Increment SR and NDL.

```
30 10 80 20 40 50 70 90 60
```

40 pivot

SR  NDL  LL
Example

Example (The Quick Sort)

- Decrement LL; swap 50 and 80.

```
30 10 80 20 40 50 70 90 60
```

**pivot**

SR  NDL  LL
Example (The Quick Sort)

- Decrement LL; swap 50 and 40.

```
| 30 | 10 | 50 | 20 | 40 | 80 | 70 | 90 | 60 |
```

- pivot
- LL
- NDL
- SR
Example (The Quick Sort)

- Increment NDL.

```
30 10 40 20 50 80 70 90 60
```

40 pivot

```
SR  NDL  LL
```
Example (The Quick Sort)

- Increment SR and NDL.

```plaintext
30 10 40 20 50 80 70 90 60
```

- SR
- NDL
- LL

40 pivot
Example

Example (The Quick Sort)

- The partitioning is done.

```
30 10 20 40 50 80 70 90 60
```

LL
NDL
SR

pivot

40

SR

LL

NDL
Now apply the algorithm recursively to the sublists {30, 10, 20} and {50, 80, 70, 90, 60}.
Example (The Quick Sort)

- **Sublist \{30, 10, 20\}. Pivot is 10.**
Example

Example (The Quick Sort)

- Decrement LL; swap 30 and 20.

```
30 10 20 40 50 80 70 90 60
```

pivot

SR
NDL
LL
Example (The Quick Sort)

- Decrement LL; swap 20 and 10.

```
20 10 30 40 50 80 70 90 60
LLNDLSR
```

- 20
- 10 (pivot)
- 30
- 40
- 50
- 80
- 70
- 90
- 60

- SR
- NDL
- LL
Example

Example (The Quick Sort)

- Increment NDL.

```
10 20 30 40 50 80 70 90 60
```

- SR
- NDL
- LL

10  pivot
As a special case, if size = 2, just compare and swap if necessary.
As a special case, if size = 2, just compare and swap if necessary.

Example (The Quick Sort)
Example

Example (The Quick Sort)

- Sort the sublist \{50, 80, 70, 90, 60\}. Pivot is 70.

```
10  20  30  40  50  80  70  90  60
```

- Pivot: 70
- LL: None
- NDL: None
- SR: None
Example (The Quick Sort)

- Increment SR and NDL.

```
10 20 30 40 50 80 70 90 60
```

- Pivot 70
- SR
- NDL
- LL
Example (The Quick Sort)

- Decrement LL; swap 80 and 60.

```
10 20 30 40 50 80 70 90 60
```

- LL
- NDL
- SR
- pivot
Example

Example (The Quick Sort)

- Increment SR and NDL.

```
10  20  30  40  70  60  90  80
```

- Pivot: 70
Example (The Quick Sort)

- Increment NDL.

10 20 30 40 50 60 70 90 80

10 20 30 40 50 60 70 90 80

pivot

SR NDL LL
Example (The Quick Sort)

- Decrement LL; swap 90 and 90.

```
10 20 30 40 50 60 70 90 80
LLNDLSR
70 pivot
```

- SR
- NDL
- LL
Example (The Quick Sort)

Sort the sublist \{50, 60\} by swapping (not necessary).
Example

**Example (The Quick Sort)**

- Sort the sublist \( \{90, 80\} \) by swapping (necessary).

```
10 20 30 40 50 60 70 90 80
```

- **pivot**: 70
- **LL**: 10
- **NDL**: 40
- **SR**: 30
- **NDL**: 80
Example (The Quick Sort)

- The list is sorted.

```
10 20 30 40 50 60 70 80 90
```

Pivot: 70

- LL
- NDL
- SR

Assignment
The `qsort()` Library Function

```c
void qsort(void* base, size_t nmemb, size_t size,
          int (*compar)(const void*, const void*));
```

The Standard C/C++ Library contains the function `qsort()` that performs a Quick Sort on an array.
- `base` is the address of the first element of the array.
- `nmemb` is the number of members in the array.
- `size` is the number of bytes in a single array element.
- `compar()` is a function that compares two array elements.
The function \texttt{compar()} returns

- A negative integer if the first element is less than the second element.
- Zero if the first element is equal to the second element.
- A positive integer if the first element is greater than the second element.
It is necessary that `compar()` represent a total order relation, or else `qsort()` will fail to sort the elements properly.
Assignment

Homework

- Read Section 13.3, pages 753 - 760.