Data Structures and Program Design in C

Topic 9: Sorting Algorithms

Dr Manjusri Wickramasinghe

Outline

- The Sorting Problem
- Basic Sorting Techniques
- Advanced Sorting Techniques

The Sorting Problem …(1)

- Sorting is the task of putting a set of elements in a predefined order.
  - Numbers from 1 to 100
  - Set of names according to alphabetical order

- The primary step in sorting is to identify a criteria to sort on.
  - Once the criteria are decided, there are many ways to sort the data.

- Sorting is a part of any basic algorithms course due to the nature of the problem it tries to solve efficiently.

The Sorting Problem …(2)

- Elementary Sorting Algorithms
  - Insertion Sort
  - Selection Sort
  - Bubble Sort

- Efficient Sorting Algorithms
  - Quick Sort
  - Merge Sort
  - Shell Sort

Insertion Sort ...(1)

- This algorithm sorts the array one element at a time by inserting an element in the correct place.

- Algorithm
  1. Set the marker for the unsorted elements (at the beginning index 1, right side)
  2. Select the first unsorted number
  3. Swap the number with the left side until the number is in the proper place
  4. Repeat steps 2 and 3 until there are no unsorted elements left.

Insertion Sort …(2)

Pseudo Code

```
insertion sort(array[])
int i,j;
for(i = 1; i < array_length; i++)
     temp = array[i]
     for(j = i; j > 0 && temp < array[j-1]; j--)
          array[j] = array[j-1]
     array[j] = temp
```

Insertion Sort ...(3)

- What can we say about the behavior of the insertion sort algorithm for the following inputs:
  - 1,2,3,4,5,6,7
  - 7,6,5,4,3,2,1
  - 6,2,4,3,1,5,7

- What are the pros and cons of this algorithm?

Selection Sort …(1)

- This algorithm localizes the exchange of array elements by finding a misplaced element and putting it in the final place.

- Algorithm
  1. Set a marker MIN for the first element of the array (at the beginning index 0)
  2. Set the marker for the entire array starting at index MIN + 1.
  3. Find the minimum value of the array and swap it with the value at the marker MIN.
  4. Increment the MIN marker to the next element.
  5. Repeat steps 2 – 4 until the end of array is reached.

Selection Sort …(2)

Pseudo Code

```
selection sort(array[])
int i,j,MIN;
for(i = 0; i < array_length - 1; i++)
    for(j = i+1, MIN = i; j < array_length; j++)
        if(array[j] < array[MIN])
            MIN = j
    swap(array, MIN, i)
```

Selection Sort …(3)

- No elements are swapped during the best-case input.

- What is the worst case of this algorithm?

Bubble Sort...(1)

- The bubble sort, as its name suggests, bubbles the smallest element into place from the end of the array.

- Algorithm
  1. Scan the array from the end to the beginning and swap elements if they are out of order.
  2. Repeat the process until all the elements are sorted.

Bubble Sort…(2)

Pseudo Code

```
bubble sort(array[])
int i,j,MIN;
for(i = 0; i < array_length - 1; i++)
     for(j = array_length -1; j > i; --j)
          if(array[j] < array[j-i])
               swap(array, j, j-i);
```

Bubble Sort...(3)

- What is the best case and the worst case of this algorithm?

- What are the pros and cons of this algorithm?

Big-Ω Notation

"f(n) is Ω(g(n)) **if there exist positive number** c **and** N **such that** f(n) ≥ c.g(n) **for all** n ≥ N."

- Big-omega notation is also known as a lower-bound analysis that ensures a given function will not perform better than what is defined in g(n)

Big-Θ Notation

"f(n) is Θ(g(n)) **if there exist positive numbers** c_1, c_2 **and** N **such that** $c_1.g(n) \leq f(n) \leq c_2.g(n)$ **for all** $n \geq N$."

- Big-Θ notation represents a tighter bound.

- It could be also read that if f(n) is Θ(n) if f(n) is O(g(n)) and f(n) is Ω(g(n)).

Analysis of Basic Sorts

	Insertion Sort	Selection Sort	Bubble Sort
Upper Bound	$O(n^2)$	$O(n^2)$	$O(n^2)$
Lower Bound	$\Omega(n)$	$\Omega(n^2)$	$\Omega(n)$
Tight Bound		$\Theta(n^2)$	

Efficient Sorting Algorithms

- The objective of an efficient sorting algorithm is to reduce the asymptotic complexity.
- These algorithms are based on different search strategies such as divide and conquer, base change, efficient data structures, etc.
- As these techniques do not involve linear search strategies, computing their asymptotic complexity is not trivial.

Merge Sort….(1)

- This was one of the first sorting algorithms used on a computer.
- Merge sort is a divide-and-conquer technique to sort the array.
- Has worst case and an average case performance of O(n.lg n)

Merge Sort….(2)

- If you are given two sorted arrays A and B, how would you create a single sorted array C using them?
  - What are the steps?
    - Initialized three-pointers to A, B and C
    - For each position in C compare the values at pointers to A and B and find the lowest
    - Replace the lowest in C
    - Increment A or B pointer depending which pointer was selected
    - Repeat until both arrays A and B are processed

Merge Sort….(3)

- Algorithm
  1. Divide the array into two sub-arrays
  2. Divide the sub-arrays further until one element is remaining
  3. Merge the single-element arrays by putting elements in the proper place
  4. Repeat until all sub-arrays are processed and merged.

Merge Sort….(3)

```
mergesort(array){
      n ← array.length()
      if(n < 2) return
      mid ← n/2
      left_sub = array[0 to mid]
      right_sub = array[mid + 1 to n -1]
           mergesort(left_sub)
           mergesort(right_sub)
           merge(left_sub, right_sub, array)
```

Quicksort …(1)

- Sorts the arrays using a pivot.
- Has a worst-case complexity of $O(n^2)$
- Has an average case complexity of $O(nlgn)$

Quicksort …(2)

- Algorithm
  1. Select a pivot value
  2. Partition the array into two parts using the pivot value
  3. Repeat steps 1 and 2 until the whole array is sorted.

Quicksort …(3)

```
quicksort(array, first, last){
    if(first < last)
    {
        pivot ← partition(array, first, last)
        quicksort(array, first, pivot)
        quicksort(array, pivot + 1, last)
    }
}
```

Quicksort …(4)

- How can the worst case be generated in this sorting algorithm?

- Why is this better than Mergesort?

Classification of Sorting Algorithms …(1)

- **Computational complexity:-** Sorting algorithms range from $O(nlogn)$ **to** $O(n^2)$ **.**
  - $O(nlog)$ → Merge Sort, $O(n^2)$ → Selection Sort

- **In-place and out-place sorting:** Whether an algorithm processes the same array or new memory is needed.
  - In-place sorting → All basic sorts
  - Out-place sorting → Merge Sort, Tree Sort

Classification of Sorting Algorithms …(2)

- **Stability:-** Whether an algorithm sorts equal elements in the same order they appear.
  - Stable sorts → Merge sort, Insertion sort, Bubble sort
  - Unstable sorts → Selection sort, Quick sort

- **Adaptability:-** Whether an algorithm considers and takes advantage of the existing order of the input.
  - Insertion sort, Shell sort

Classification of Sorting Algorithms …(3)

- **Online Algorithm:-** whether the algorithm can process information serially in the order the inputs are presented to the algorithm.

- Comparison sorts or recursive sorts.

Questions?