ECE 2574: Data Structures and Algorithms - Basic Sorting Algorithms

C. L. Wyatt

Today we will continue looking at sorting algorithms

- Bubble sort
- Insertion sort
- Merge sort
- Quick sort

Common Sorting Algorithms

Algorithm	Worst Case	Best Case
Selection Sort	n^2	n^2
Bubble Sort		
Insertion Sort		
Merge Sort		
Quick Sort		

Bubble Sort works by iteratively swapping adjacent elements if needed.

Given a list of keys {k1, k2, k3, ... kn}
Starting at positions 1 and 2 if key 2 $<$ key 1 swap them
continue for positions 2, 3 then 3, 4 all the way to n-1, n. Repeat for the list {k1, k2, k3, ... kn-1}
Stop when no exchanges are done in the traversal.

Bubble Sort Example

First pass

```
12 5 237 64 39 78
5 12 237 64 39 78
5 12 237 64 39 78
5 12 64 237 39 78
5 12 64 39 237 78
5 12 64 39 78 237
```

Bubble Sort Example

Next passes

```
5 12 64 39 78 237
5 12 64 39 78 237
5 12 64 39 78 237
5 12 39 64 78 237
5 12 39 64 78 237
```

Bubble Sort pseudocode (1-based indexing)

```
BubbleSort( in list:List )
last = list.length
do
    // swap if needed
    nswaps = 0
    for i = 1 to last-1
        if (list[i] > list[i+1])
           // swap the entry
           temp = list[i+1]
           list[i+1] = list[i]
           list[i] = temp
           nswaps += 1
    endif
    endfor
    last = last - 1
while (nswaps > 0 AND last > 1)
```

Bubble Sort: Number of compares and moves

In the worst case there are n-1 passes through the array, each time requiring (n-1), (n-2) . . . .(1) comparisons.
Thus the number of comparisons is n*(n-1)/2
If we have to swap each time then there are 3$(n$(n-1)/2) moves
Thus in the worst case Bubble sort is O(nˆ2)

Bubble Sort: Number of compares and moves

What about the best case?
What is the array is already sorted ?
Then there are (n-1) comparisons and 0 moves giving a complexity of O(n)

Common Sorting Algorithms

Algorithm	Worst Case	Best Case
Selection Sort	n^2	n^2
Bubble Sort	n^2	n
Insertion Sort		
Merge Sort		
Quick Sort		

Insertion sort

Divide the list into a sorted and an unsorted region.
Initially the sorted region is the first item.
Choose the first item in the unsorted list.
Locate its slot in the sorted list by sequential comparisons.
Insert the item.
Continue until the unsorted list is empty.

Insertion Sort Example

```
12 5 237 64 39 78
5 12 237 64 39 78
5 12 237 64 39 78
5 12 64 237 39 78
5 12 39 64 237 78
5 12 39 64 78 237
```

Insertion Sort Pseudocode (1-based indexing)

```
InsertionSort( in list:List )// assume list size > 1
i = 2 // index of first item in the unsorted region
do
    // insert in sorted list
    temp = list[i]
    for j = 1 to i
        if (temp <= list[j])
             // different for array versus linked list
             // and if i = j
            insert_before(temp, j)
            break
        endif
    endfor
    i=i+1
while (i <= list.length)
```

Insertion Sort: Number of compares and moves

The outer do-while loop executes n-1 times.
The inner for loop executes $1 + 2 + 3 + \ldots$ (n-1) = n*(n-1)/2 times, with one compare each time in the worst case.
The number of moves depends on the implementation but is linear for a linked list.
Thus the worst case complexity is O(nˆ2)

Insertion Sort: Number of compares and moves

What is the best case scenario?
What if the list was in reverse sorted order ?
Then the insert step would always insert at 1 and the complexity would be O(n).

Common Sorting Algorithms

Algorithm	Worst Case	Best Case
Selection Sort	n^2	n^2
Bubble Sort	n^2	n
Insertion Sort	n^2	n
Merge Sort		
Quick Sort		

Merge sort is a divide-and-conquer algorithm.

Invented by John von Neumann in 1945!
Let the list be given by L[first, last].
Let A, B, C be the sublists L[first, mid] L[mid+1,last] and L[first,last], where mid is the midpoint between first and last.
Recursively sort sublist A, then B, them merge them to give the sorted list C

Merge sort example: recursive calls

```
            12,5,237,64,39,78
       12,5,237          64,39,78
    12,5     237       64,39    78
  12    5           64     39    78
```

Merge sort example: merging

```
          5,12,39,64,78,237
     5,12,237         39,64,78
  5,12     237      39,64    78
12    5          64    39    78
```

Merge Sort psuedocode

```
function MergeSort(L, first, last)
// L is a list of keys
// first and last are indices
// defining the sublist
if(first < last)
  mid = floor ( (first + last)/2)
  MergeSort(L, first, mid)
  MergeSort(L, mid+1, last)
  Merge(L, first, mid, last)
endif
```

How many times is Merge called ?

Here we make a simplifying assumption, that the list is even at each call to merge sort, giving a full binary tree.
What is the maximum level in a binary tree with n nodes ?

How many times is Merge called ?

Here we make a simplifying assumption, that the list is even at each call to merge sort, giving a full binary tree.
What is the maximum level in a binary tree with n nodes ?
log_2(n)
So what is the time complexity of merge ?

Common Sorting Algorithms

Algorithm	Worst Case	Best Case
Selection Sort	n^2	n^2
Bubble Sort	n^2	n
Insertion Sort	n^2	n
Merge Sort	nlog_2(n)	nlog_2(n)
Quick Sort		

MergeSort is stable

A stable sort is one in which the order of repeated keys are retained after sorting.
Why would you care?
Example: you want to sort people/events by an integer priority, but if two people/events have the same priority, they are treated in the order of arrival.

The space complexity of MergeSort

The space complexity is the additional memory (beyond the list itself) required to do the search, i.e. the scratch space.
Merge requires a temporary list at most as large as the list, so its space complexity is O(n).

QuickSort (partition-exchange sort)

Quicksort is also a divide and conquer algorithm, published by Tony Hoare in 1961.

Quicksort is not stable, but has O(n log(n)) best and average time complexity and O(log n) space requirements.

Quicksort

1. Pick an element, called a pivot, from the list.
2. Partitioning: reorder the list so that all entries less than the pivot come before the pivot, while all entries greater than the pivot come after. Note, the pivot is now in its final position.
3. Recursively apply the above steps to the sub-lists to either side of the pivot. The base case is an empty list or list of size 1.

Quicksort pseudocode

```
funciton QuickSort(L, first, last)
  if first < last
    pivot = partition(L, first, last)
    quicksort(L, first, pivot)
    quicksort(L, pivot + 1, last)
  endif
endfunction
```

Choice of pivot

Often this is just the first element. This is the worst case when the list is already sorted giving O(n^2).
Already sorted lists are encountered frequently, so instead one can choose the pivot:

- randomly
- median of first, middle and last element of the partition

That completes our list of Common Sorting Algorithms

Algorithm	Worst Case	Best Case
Selection Sort	n^2	n^2
Bubble Sort	n^2	n
Insertion Sort	n^2	n
Merge Sort	nlog_2(n)	nlog_2(n)
Quick Sort	n^2	nlog_2(n)

There are many sorting algorithms . . .

Two other popular ones are:

- IntroSort
- TimSort

What is the best possible sort?

With Information Theory one can prove that no sorting algorithm can do better than O(n log(n)) on average!

Next Actions and Reminders

- Midterm exam is in-class Friday. Be on time!
- P3 is due 10/31 by 11:59 pm