Sorting Algorithms

May 17, 2013

Insertion Sort, Selection Sort, Heapsort, Mergesort

Some Definitions Before Beginning:

In place sort:

Sorting takes place inside the same array that is holding items. This way, it uses only O(1) or O(log n) additional memory

Divide and Conquer: (divide-conquer-combine)

Algorithm design based on multi-branched recursion;

Breaks down problem into smaller instances of same problem (subproblems) Conquer subproblems by solving them recursively Glue answers together as a solution to the original problem.

Sorting Algorithms

1 Insertion Sort

Run time: **O(n^2)**
Invariant: List S is sorted

How it works:
Start with empty list S and unsorted list I with n times

For (each item x in I) {
Insert x into list S, positioned so S remains in sorted order
}

In place insertion sort:

Partition array into two pieces:
Left is initially empty (will be sorted list S)
Right is holding the items n (I)

Iterate through the array beginning at the left most item in I and insert into S. Iterate through I to find the item with the smallest key, then SWAP it to the position in S so that S remains in sorted order. With each iteration, the dividing line b/t S and I moves one step to the right. Continue down the array; S will evenually grow to consume all of I.

- Basically, swap as you go if the next item is smaller

2 Selection Sort

Run time: **O(n^2)** – doesn't matter if array or linked list...finding smallest item takes $\Theta(n)$time, so selection sort takes $\Theta(n^2)$, even in best case.

Invariant: S is sorted list

How it works:
S= empty list
I = unsorted list of n items

Walk through I, **pick out minimum item**, append to end of S
For (i=0; i<n; i++) {
Let x be the item in I having smallest key
Remove x from I.
Append x to end os S.
}

In place selection sort:

Search for smallest element in array and swap into first position. Then, search the rest of the entries for the next smallest element and swap into second position. Continue for rest of the elements until the array is completely sorted.

How it works:

Need int variable x to store integer for smallest integer we've found so far. Loop through the array. Once we've reached the end, x holds the index of the smallest entry in the array... swap to the first position. Initialize second entry as the smallest, test against rest of the entries. Swap. Continue.

3 Heapsort Selection Sort (doing selection sort on heap)

Runtime:

There are 2 parts: bottomUpHeap(), and removeMin(). BottomupHeap() runs on linear time, and each removeMin() takes O(logn) time, so Heapsort runs

in **O(nlogn) time**

How does it work?

2 steps:

1) Start with empty list S and unsorted list I. 1. Put all items I into a heap h (without caring about the heap order property), then do : h.bottomUpHeap(); which then enforces the heap order property for items in h.

2) Remove min item in h, append to S for all items in heap. Start with an empty list S and an unsorted list I of n input items. toss all the items in I onto a heap h (ignoring the heap-order property).

Do: h.bottomUpHeap(); // Enforces the heap-order property

Do:for (i = 0; i < n; i++) { x = h.removeMin(); Append x to the end of S. }

Notes: Better for arrays; worse for linked lists. Use mergesort for linked lists Mergesort

O(n) time: two input lists are sorted, so there are only two items to test each time.

Each level of the tree involves O(n) operations, and there are O(logn) levels

4 Mergesort

Runtime: runs in **O(nlogn)** time

Selection sort with two sorted queues... merge into one sorted list in linear time

How does it work?

At each iteration, it choose item having smallest key from two input lists, appends it to output list (start off with the front of each of the two lists). Memory efficient for linked lists, memory inefficient for arrays. Mergesort is **not an in-place algorithm.**

Let Q1 and Q2 be two sorted queues. Let Q be an empty queue.

while (neither Q1 nor Q2 is empty) {
item1 = Q1.front();
item2 = Q2.front();
move the smaller of item1 and item2 from its present queue to end of Q. }
concatenate the remaining non-empty queue (Q1 or Q2) to the end of Q.

Recursive divide-conquer algorithm (reunite what we divided)

How does ^this work? :

Start with unsorted list I of n items

Break I into halves I1 and I2, having ceiling (n/2)
Sort I1 recursively (keep breaking into two)... to yield sorted list S1
Sort I2 " " ... yield S2
Merge S1 and S2 into sorted list S