Lecture 7

Monday, 29 July 2013
11:07 a.m.

Merge sort

On modern computers with multiple processors is can parallely process each comparisons cause they're are from separate arrays. It's a parallel program
With large amount of data on external memory it can split them up then process small chunks on internal memory with out having to transfer large chunks of memory
Because with small numbers merge sort isn't the fastest/most efficient we can have a statement that checks for how large n is and then use the more efficient method to sort

Quicksort - new algorithm

1. If the array is larger than 1 (the stopping case of recursion)
2. find the median value or pivot (the number that will be in the middle of the final sorted array) and it puts that in the middle of the array, using the partition algorythm
3. Then put all the large values than the pivot above it and all the smaller values below it
4. Then it repeats with the upper and lower sub arrays (like merge sort)

The tricky part is getting the pivot point right because if you pick the wrong one is that the sub arrays are going to be skewed and this can increase O() time a lot.
If the array is truly randomly selected any item in the array would have the same chance as being the median value.

The partition algorithm (for quicksort)

This is the method that does all the work for the quick sort, it picks the pivot value and then puts all the bigger items above and all the low items below
NOT REALLY GOOD WITH REPEATED NUMBERS

1. Assign the first value as the pivot
2. Make variables i,j that are pointing at the start (i) and end (j) of the array
3. Move these variables through the array until they find a value that should be on the larger (for i) or smaller (for j) of the pivot.
4. When both of them find a value it swaps these values and then carry's on.