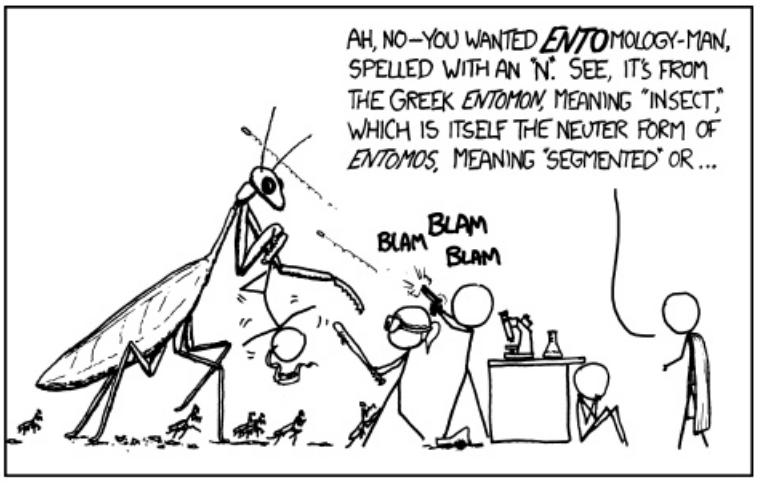
CS302 - Assignment 5

Due: Thursday, Feb. 28 at the beginning of class
Hand-in method: paper



http://www.xkcd.com/1012/

For this assignment you must use latex to generate your work.

1. **[8 points]** Even $O(n \log n)$ may not be fast enough for certain cases where the data set is very large and/or when response time is very important. One way to solve this problem is to use multiple machines to sort the data. In particular, assume we have m machines that we can use for sorting the data.

   As an example of how you can use multiple machines to implement an algorithm, here is an algorithm for using m machines to calculate the sum of an array:

   - Assume that all machines have access to the data

- Divide the data into n/m-sized chunks.
- Each machine calculates the sum of a particular n/m-sized chunk.
- Each machine then sends their sum to a single machine to calculate the sum of sums, which is returned as the overall sum of the array.

The run-time of a non-parallelized version would be $\Theta(n)$. The cost of the parallelized version (ignoring communication overhead) is $\Theta(n/m)$ for the sorting (each machine does this in at the same time in parallel) plus $\Theta(m)$ for the sum of sums. This gives us a total of $\Theta(n/m + m)$, which, if $n \gg m$ (a very reasonable assumption), is dramatically faster and gets faster by adding more machines.

For the following sorting methods: INSERTION-SORT, MERGE-SORT, QUICKSORT and COUNTING-SORT, which algorithm is the most amenable to parallelization? Why? What are the challenges for the others? Be clear, but concise.

2. **[12 points]** Consider the following sorting algorithm: sort the first two-thirds of the elements in the array, then sort the last two thirds of the array and finally sort the first two thirds again. Specifically:

```
TRIPLE-SORT(A, i, j)
1  if A[i] > A[j]
2          swap A[i] and A[j]
3  if i + 1 ≥ j
4          return
5  k ← ⌊(j − i + 1)/3⌋
6  TRIPLE-SORT(A, i, j − k) # sort the first two-thirds
7  TRIPLE-SORT(A, i + k, j)# sort the second two-thirds
8  TRIPLE-SORT(A, i, j − k) # sort the first two-thirds again
```

which you would call to sort A with TRIPLE-SORT$(A, 1, A.length)$.

(a) (4 points) Give an informal but convincing explanation of why the algorithm above is correct. The explanation does not need to be more than a few sentences, but be precise. (*Hint:* Given that the above procedure is recursive your "explanation" will likely be an inductive argument. If this is the case, make sure to argue both about the inductive case AND the base case.)

(b) (3 points) Describe the recurrence relation for the run-time of Triple-sort.

(c) (3 points) What is the run-time of Triple-sort (i.e. solve the recurrence)?

(d) (2 points) How does this algorithm compare to Insertion-Sort, Merge-Sort, Quicksort?