**CS430 Lecture 1 Activities**

Opening Questions

1. Algorithm A takes 5 seconds to sort 1000 records, and Algorithm B takes 10 seconds to sort 1000 records. You have the code for both algorithms. When deciding which algorithm to use to sort up to 1,000,000 records, why might Algorithm B be the better choice?

2. Why is it helpful to sometimes define a problem in its most basic mathematical terms?

3. In your own words explain what a loop invariant is.

4. What are the three kinds of growth in run time analysis we may do on an algorithm.

5. For recursive algorithms, what do we need to define and solve to do the runtime analysis?

Sorting as a Case Study for Algorithm Analysis

1. Write pseudocode for one of these iterative sorts: InsertionSort, BubbleSort, SelectionSort. Then draw pictures for a sample run on 5 random numbers showing comparisons/swaps.

2. Write the loop invariant for your sort and prove your sort is correct by proving Initialization, Maintenance, and Termination.

Resource (memory or runtime) Use Analysis – Resource use analysis usually depends on the size of the input to the algorithm. You can write a function T(n) that matches the behavior of the resource use of the algorithm.

NOTE: For recursive algorithms we develop and solve a recurrence relation to find the T(n), the resource use function.

3. For the iterative sort you wrote above, construct a run time analysis function T(n) by assigning a different constant (c1, c2, c3, etc) to each type of statement (the run time for statements of that type), and counting how many times each statement executes for an input size n. Then sum up the constants times the execution counts. You may need to define variables other than n if there are event controlled iterations with an unknown number of loops.

For a more descriptive analysis we need to consider the various generic execution flows and input structures that result in those generic execution flows. There are 3 more descriptive resource use analyses: 1) worst case (usually used) 2) average case (sometimes used) 3) best case (hardly ever used)

4. For your T(n) function from above determine the best case, worst case and average case T(n)s.

Asymptotic Analysis – To simplify comparing the resource usage of different algorithms for the same problem

* ignore machine dependent constants; look at the growth of T(n) as n-> infinity
* as you double n, what does T(n) do?? double?? square??

Theta Notation (more details in future lectures)

* Drop lower order terms;
* Ignore leading constants
* Concentrates on the growth

5. For your best case, average case, worst case T(n) functions from above give the asymptotic function (Theta notation)

6. Given the problem sizes and worst case runtime for one of the problem sizes, and what you know about each algorithm, predict the missing runtimes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | n=100 | n=200 | n=400 | n=800 |
| Linear search | 10 seconds |  |  |  |
| Binary search |  | 8 seconds |  |  |
| Insertion Sort |  |  | 320 seconds |  |