**CS430 Lecture 2 Activities**

Opening Questions

1. How did we approach average case runtime analysis of iterative algorithms previously? How can we improve on this?

Expectation of a Random Variable

A random variable is a variable that maps an outcome of a random process to a number. Examples:

* Flipping a coin, If heads X=1., if tails X=0
* Y=sum of 7 rolls of a fair die
* Z=in insertion sort, the number of swaps needed to move the ith item to its correct position in items 1 thru (i-1)

|  |  |
| --- | --- |
| The expected value of a random variable X is sum over all outcomes of the value of the outcome times the probability of the outcome. |  |

1. What is the expected outcome when you roll a fair die once? What about a loaded die where the probability of a side coming up is the value of the side divided by 21?

2. Calculate the expected outcome when you roll a fair die twice and sum the results. Do this two different ways.

Now let’s use expectation of a random variable to improve our average case runtime for insertion sort (similar for bubble sort or selection sort).

* Sort *n* distinct elements using insertion sort
* *Xi* is the random variable equal to the number of comparisons used to insert *ai* into the proper position after the first *i-1* elements have already been sorted. *1<=Xi<=i-1*

E(Xi) is expected number of comparisons to insert *ai* into the proper position after the first *i-1* elements have been sorted.

E(X) = E(X2)+E(X3)+ . . . +E(Xn) is the expected number of comparisons to complete the sort (our new average case runtime function).

3. Write equations for the following and simplify.

E(Xi)

E(X)

4. What if the data is not random?

Recursive Sorting – Mergesort

* divide and conquer (and combine) approach, recursive algorithm
* key idea: you can merge two sorted lists of total length n in THETA(n) linear time
* base case: a list of length one element is sorted

5. Demonstrate how you can merge two sorted sub-lists total n items with n compares/copies. How much memory do we need to do this? Write pseudocode to do this.

2 3 7 8 1 4 5 6

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

Mergesort(A, p, r) { // initial call Mergesort (A, 1, n)

if (p<r) {

q = (p+r)/2 // integer division

Mergesort(A, p, q) // recursively sort 1st half

Mergesort(A, q+1, r) // recursively sort 2nd half

Merge(A, p, q, r) // merge 2 sorted sub-lists

}

}

6. Demonstrate Mergesort on this data

3 41 52 26 38 57 09 49

7. A recurrence relation describes runtime function recursively for a recursive algorithm. Write a recurrence relation for the Mergesort algorithm. HINT: try to count the number of executions of each statement.

Solving Recurrence Relations – Recurrence Tree Method

We solve a recurrence relation to get a function in its closed (non-recursive) form. The recurrence tree method is a visual method of repeatedly substituting in the recurrence relation for T(n) on smaller and smaller “n” until you reach the base case, and then summing up all the nodes in the tree.

Asymptotic Analysis (more details)

BIG-O Notation – Upper bound on growth of a runtime function

f(n) ∈ O(g(n)) “f(n) is big-O of g(n)”

If there exists C, no such that

0 < f(n) < Cg(n) when n> no

8a. Use the definition of big-O to show 2n^2 is big-O n^3 (find a C and no that works in the above)

8b. Use the definition of big-O to show T(n)=3n^3-4n^2+3lgn-n = O(n^3)