**LECTURE 2**

**ANALYSIS OF INTSERTION SORT**

Running time T(n) =

c1n + c2(n-1) + c4(n-1) + c5S tj + c6S (tj-1) + c7S (tj-1) + c8(n-1)

Here:

1. Subscripts refer to statements in the pseudocode.

2. tj is the number of times the while loop test is executed for that value of j.

3. Summation S runs over all values of j, that is j=2 to j=n.

Best case occurs for already sorted array, in which case no values need to moved from one place to another. Running time is Q(n).

Worst case occurs for reverse sorted array, in which case tj, the number of executions of the loop test, is at its maximum. Running time is Q(n2).

Even if we assume that tj = j/2 on average, we still get running time Q(n2).

**DIVIDE AND CONQUER APPROACH**

1. Divide the problem into a number of sub-problems.

2. Solve the sub-problems recursively, but if a sub-problem is “small enough", solve it in a “straightforward” manner, without further sub-division.

3. Combine the solutions to sub-problems into the solution to the original problem.

EXAMPLE:

**MERGE-SORT( A, p, r )**

**if p < r**

**q = floor((p+r)/2)**

**MERGE-SORT( A, p, q )**

**MERGE-SORT( A, q+1, r )**

**MERGE( A, p, q, r )**

For the function MERGE, please see [CLRS] <-- **SELF STUDY**.

**ANALYSIS OF RUNNING TIME**

DIVIDE step: Constant running time, denoted as D(n) = Q(1).

RECURSION step: T(n) = 2T(n/2)

MERGE step: C(n) = Q(n).

Combining these:

Q(1), if n = 1

T(n) =



2T(n/2) + Q(n), if n > 1

Solution is T(n) = Q(*n* log2*n*) ) = Q(*n* lg *n*).

[We accept without proof, but we can always check by back-substitution.]

Solution can be understood by making use of a balanced binary tree, assuming for that purpose that n = 2k, for some integer k.

A so-called “master theorem" sets out the general case, as we will see later.

OTHER EXAMPLES

In Bubble-sort, we count the number of exchanges. Running time Q(*n*2).

In matrix multiplication, we count the number of scalar multiplications. Simple algorithm gives running time Q(*n*3).

Binary search in a sorted array takes running time Q(log2*n*).

Finding the maximum or minimum element in an unsorted array takes running time Q(*n*).