**Recurrences are studied in determining the running times of recursive algorithms**

General idea of mergeSort. Let T(*n*) be time to run algorithm on array input of size *n*.

**mergeSort(a, i, j)** // time = T(n)

if (i == j) // time = 1

return

m = (i+j)/2 // time = 1

mergeSort(a, i, m) // time = T(n/2)

mergeSort(a, m+1, j) // time = T(n/2)

merge(a, i, m, j) // time = θ(n)

**merge(a, i, m, j)**

{

// merge two sorted subarrays a[i..m-1] and a[m..j] into one

// sorted subarray a[i..j]

// How much time is required for the merge operation?

}



The amount of time *T*(*n*) is expressed recursively as a function of the time to solve a smaller subproblem.

This does not give us an asymptotic bound – only a recursive formula for the amount of time on a problem half its size.

We need to **solve** the recurrence to determine a formula for the amount of time as a function of the input size *n*.

How do we solve this type of recurrence??

T(*n*) = *O*(???) or θ(???)

**Solving recurrences by the Iteration Method**

T(*n*) = 2 \* T(*n*/2) + n

= 2 \* [ 2 \* T( ) + ] + n

= 2 \* [ 2 \* [ ] + ] + n

=

(See PowerPoint for Iteration Method for MergeSort)

Draw a recursion tree to demonstrate finding total amount of time.