**Sorting Algorithms**

**bubbleSort(a)**

n = a.last

for i = n downto 2

for j = 1 to i-1

if a[j] > a[j+1]

temp = a[j]

a[j] = a[j+1]

a[j+1] = temp

**selectionSort(a)**

n = a.last

for i = 1 to n-1

minPos = i

for j = i+1 to n

if a[j] < a[minPos]

minPos = j

if minPos != i

temp = a[minPos]

a[minPos] = a[i]

a[i] = temp

**insertionSort(a)**

{

n = a.last

for i = 2 to n

{

val = a[i]

j = i – 1

while (j >= 1 && val < a[j])

{

a[j + 1] = a[j]

j = j – 1

}

a[j + 1] = val

}

}

**mergeSort(a, i, j)**

{

if (i == j)

return

m = (i + j) / 2

mergeSort(a, i, m) **// How do we measure the time**

mergeSort(a, m+1, j) **// for a recursive algorithm?**

merge(a, i, m, j)

}

**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?

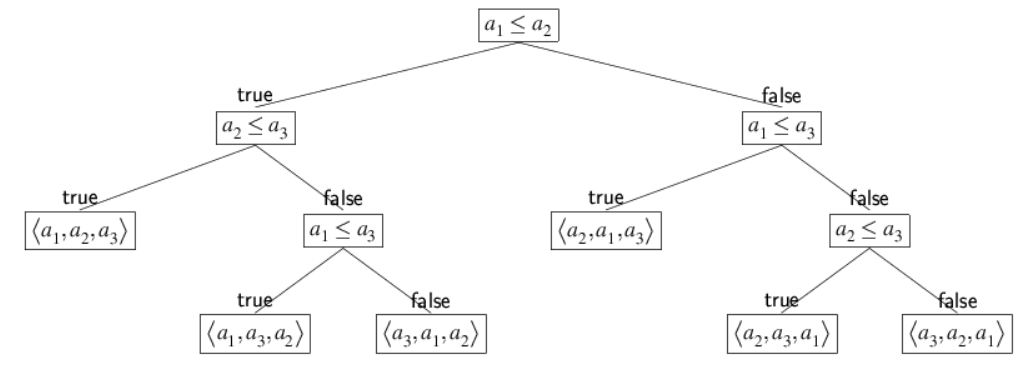
}

**Section 6.3: A Lower Bound for the Sorting Problem**

MergeSort and QuickSort are two efficient sorting algorithms, both running in time O(*n* lg n). But can we do any better than this?

**Theorem** Any comparison-based sorting algorithm has worst case time Ω(n lg n).

Consider the decision tree for an array with 3 elements. There are 6 leaves in this tree, and the tree has height 3.



**Generalize**: The decision tree for comparing and ordering an array of n elements has n! leaves.

In the worst case, the number of comparisons ≥ height of tree

≥ lg (n!)

= Ω(n lg n)

**Conclusion**: Any comparison-based sorting algorithm must take time **at least** n lg n. There is no hope of finding any faster comparison-based algorithm.

**Section 6.4 Sorting in Linear Time**

CountingSort can only be used on a restricted type of data set. The data must be a set of integers in the range 0…*m*. Generally, the value *m* should be “small”, say *m* = *O*(*n*).



Trace through countingSort on the array a below. Note that the range of the data is 0…6. So the call to the algorithm would be countingSort(a, 6). (Show work on separate paper.)

|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **a** | **4** | **3** | **0** | **3** | **6** | **2** | **0** | **1** | **3** | **4** |

RadixSort can only be used on a restricted type of data set. Every integer in the data set must have a fixed number of digits, say *k*.



Determine the running time of each of the algorithms.