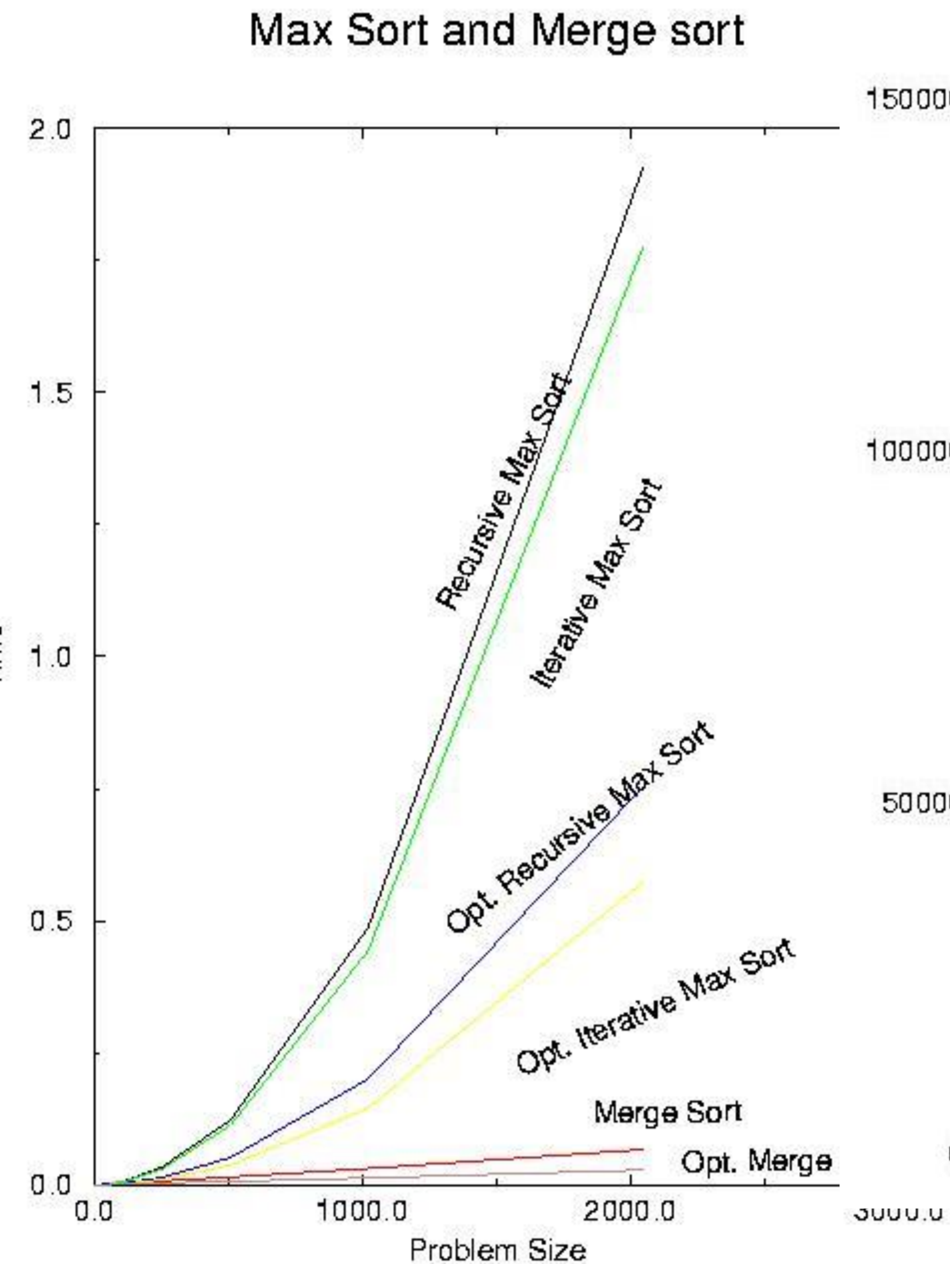
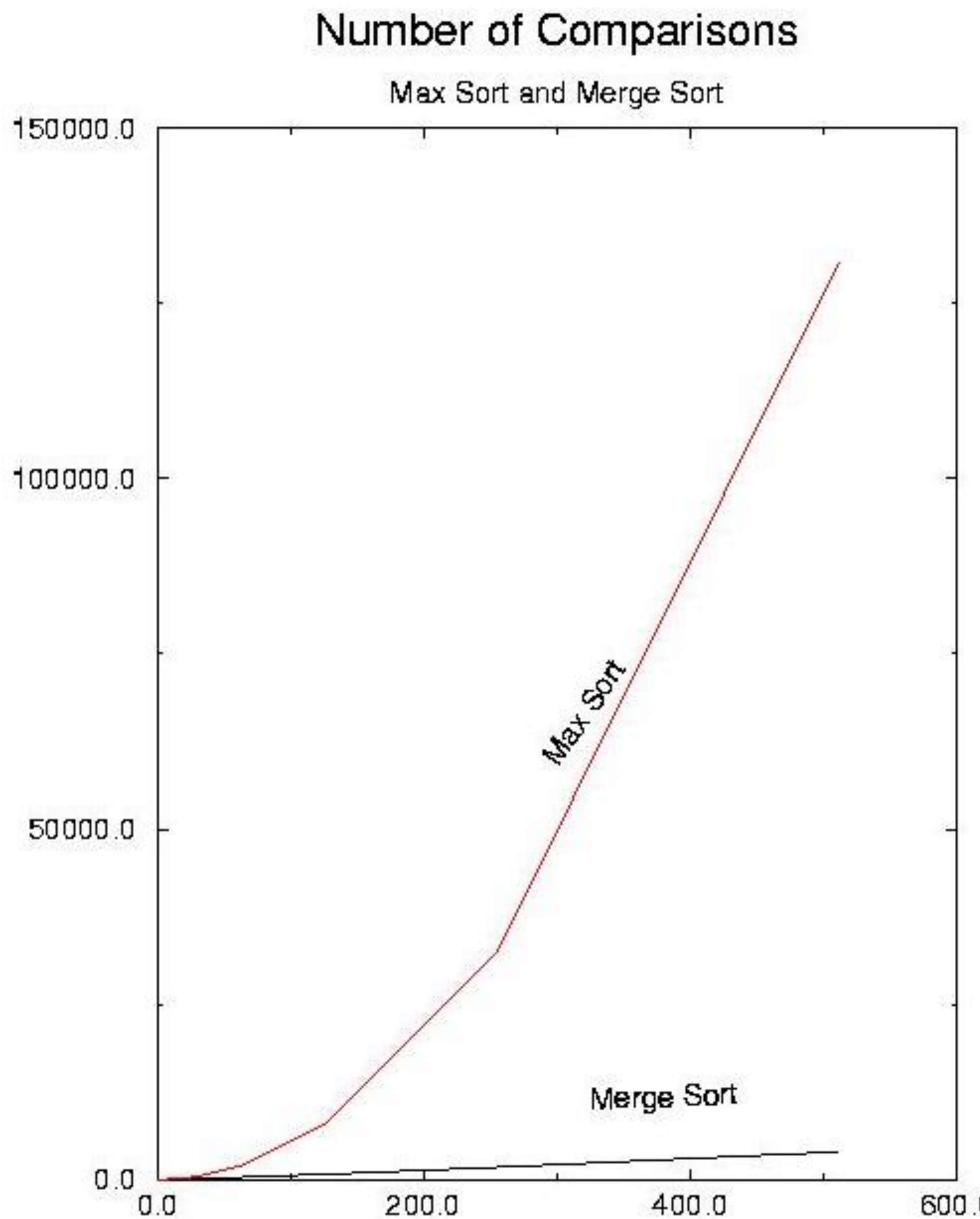
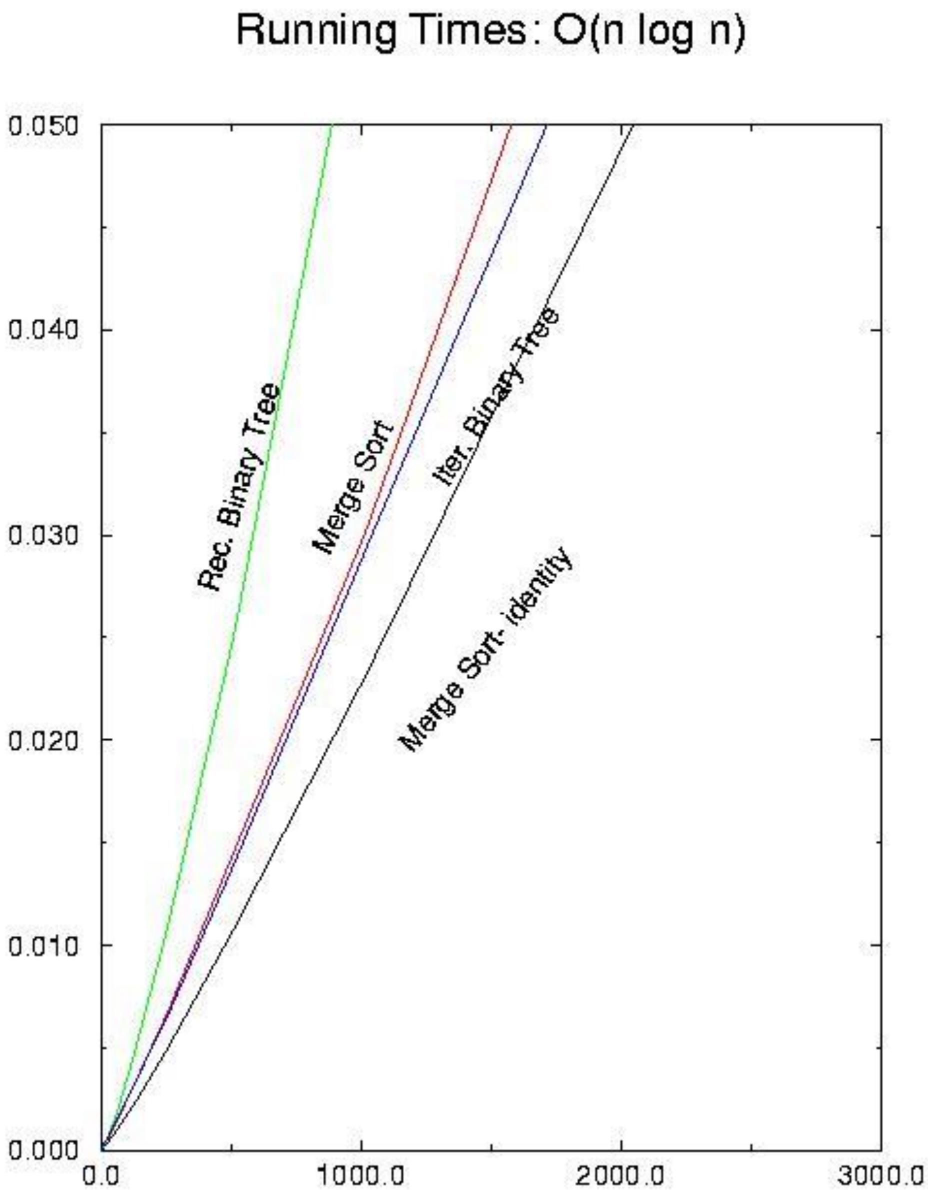
Actual Running Times of Some Sorting Algorithms

In 1999, the CSC 225 students programmed various sorting algorithms in C and timed them on various inputs.

This is where the following plots came from.

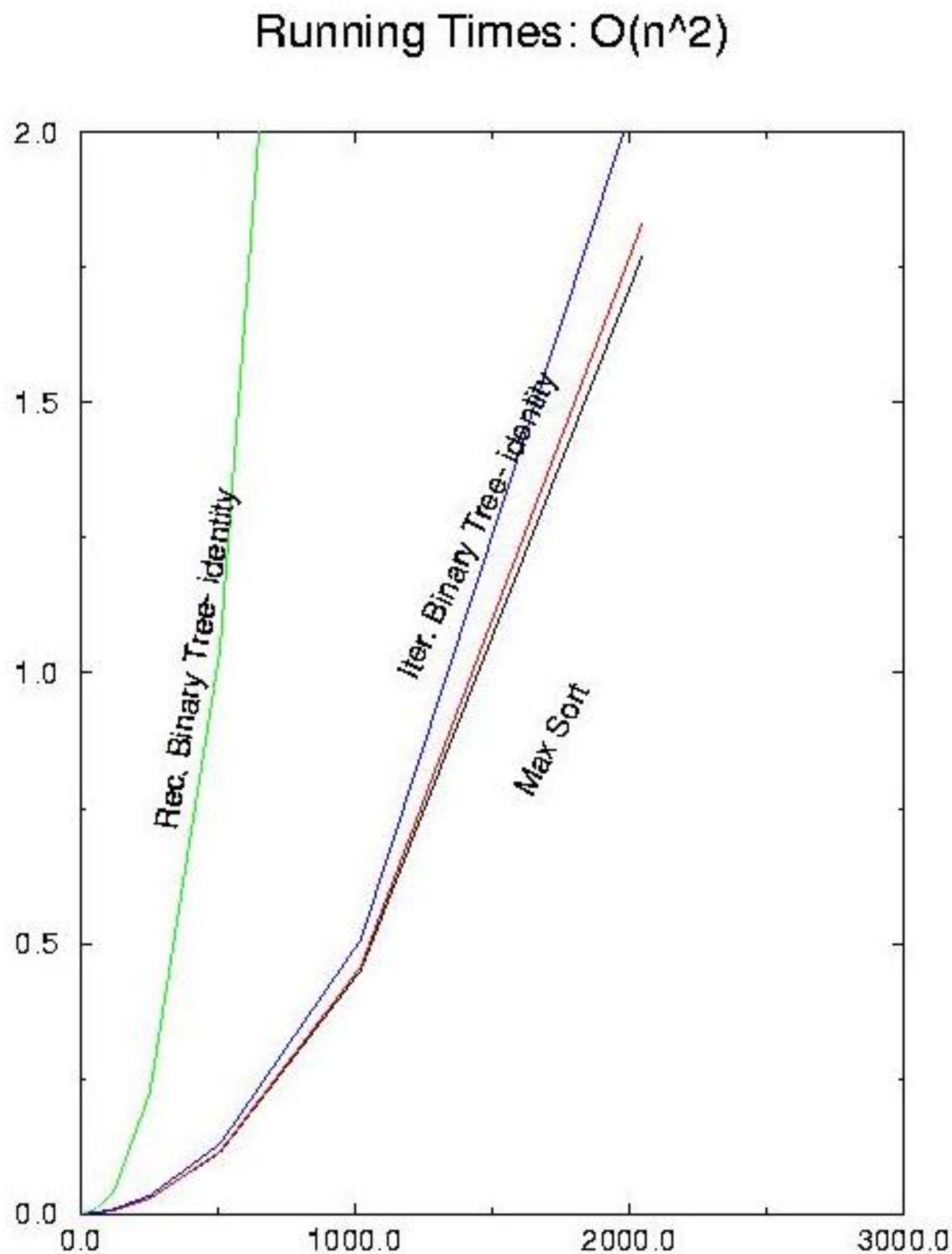

Max Sort and Merge sort
2.0
1.5
1.0
0.5
0.0
0.0
1000.0
2000.0
Problem Size
Recursive Max Sort
Iterative Max Sort
Opt. Recursive Max Sort
Opt. Iterative Max Sort
Merge Sort
Opt. Merge


Number of Comparisons
Max Sort and Merge Sort
150000.0
100000.0
50000.0
0.0
0.0
200.0
400.0
600.0
Max Sort
Merge Sort

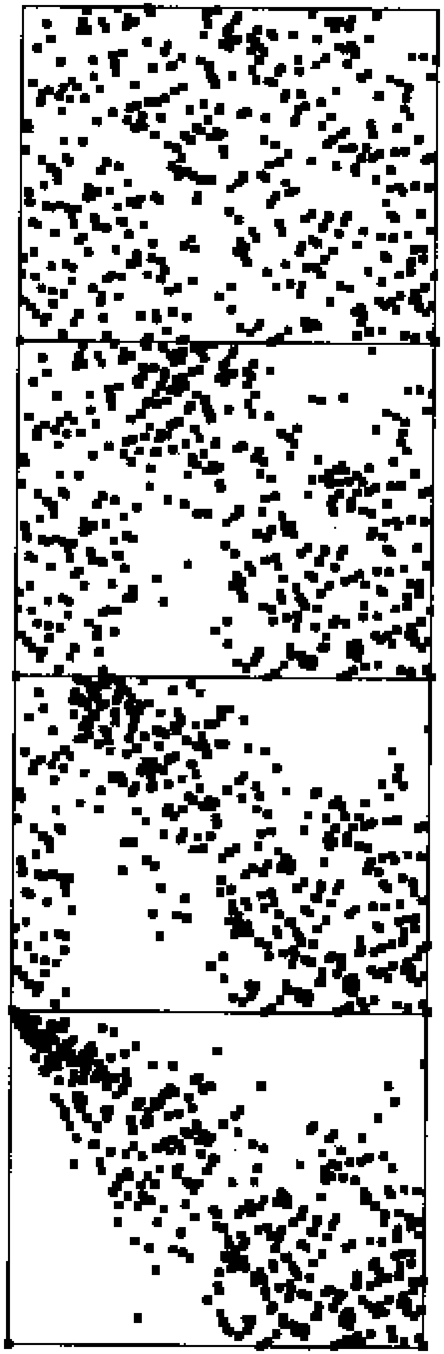


Binary
Tree
Sort

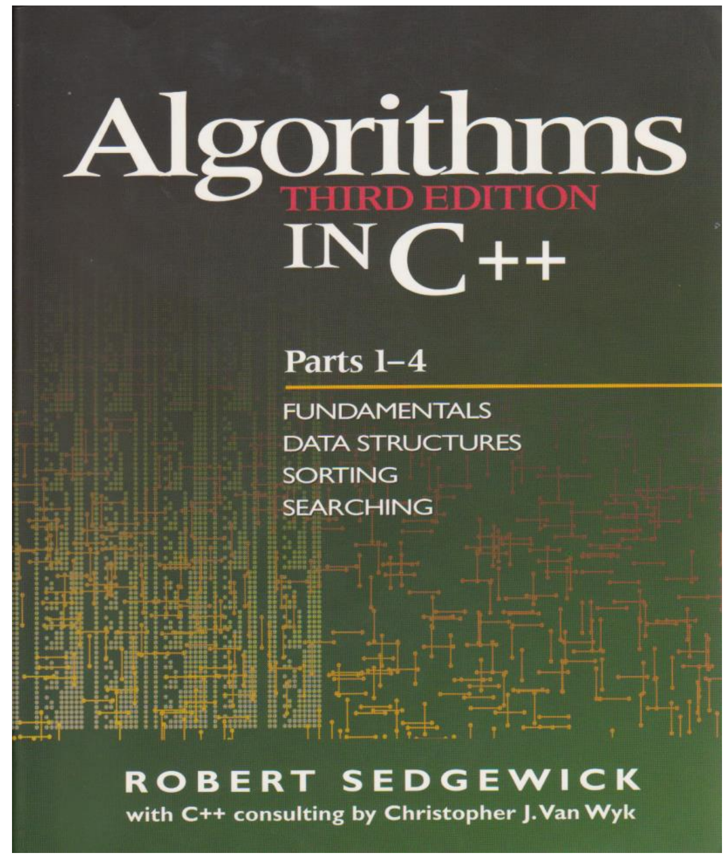
Random
Inputs



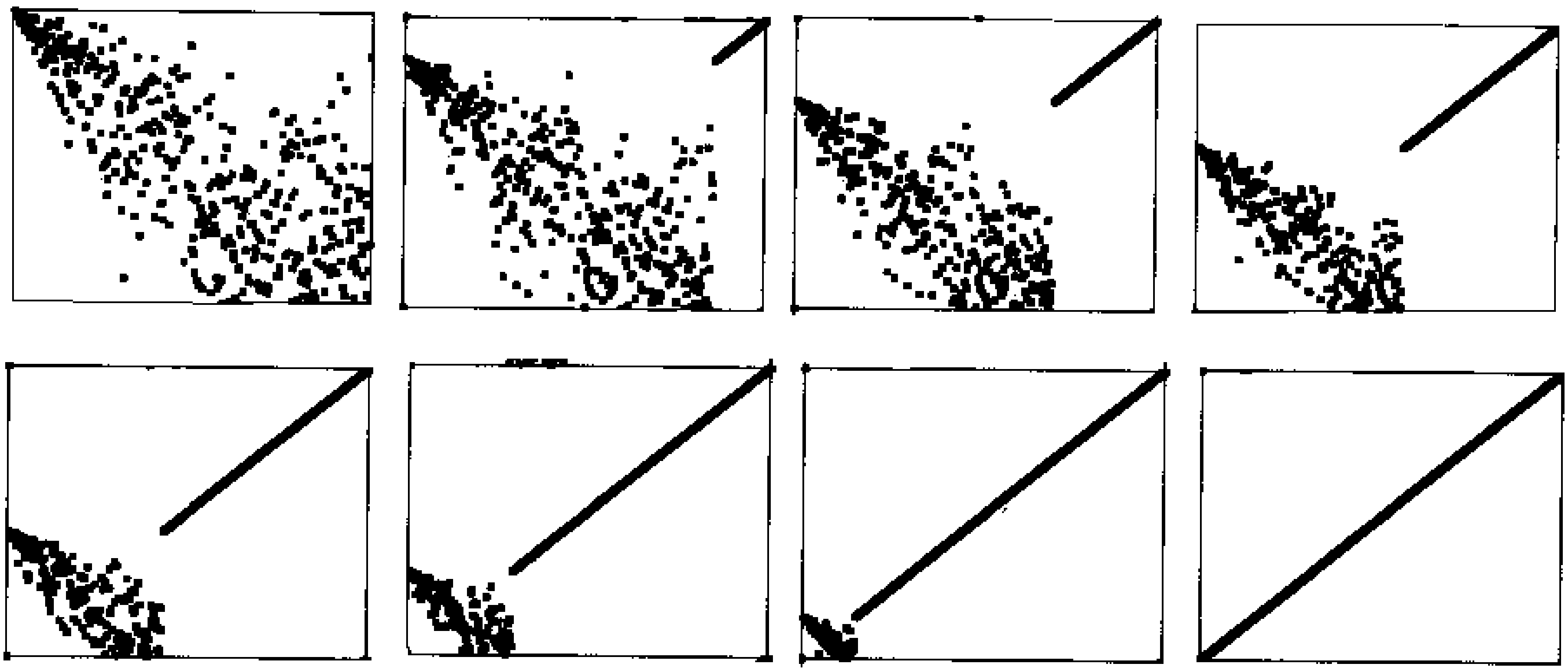
Sorted
Inputs



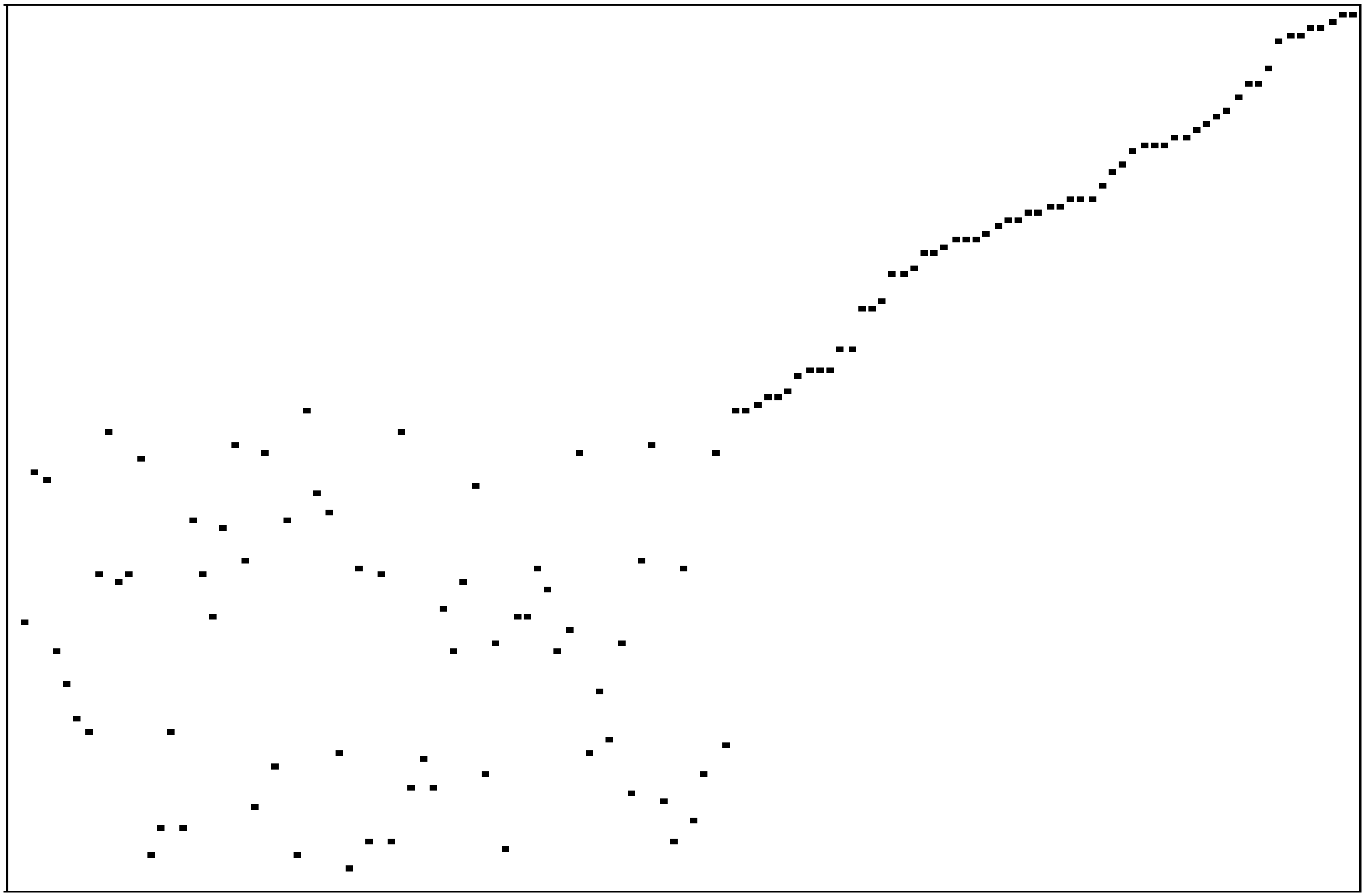
Building the heap- which algorithm is this?



Dynamic Performance of Heapsort

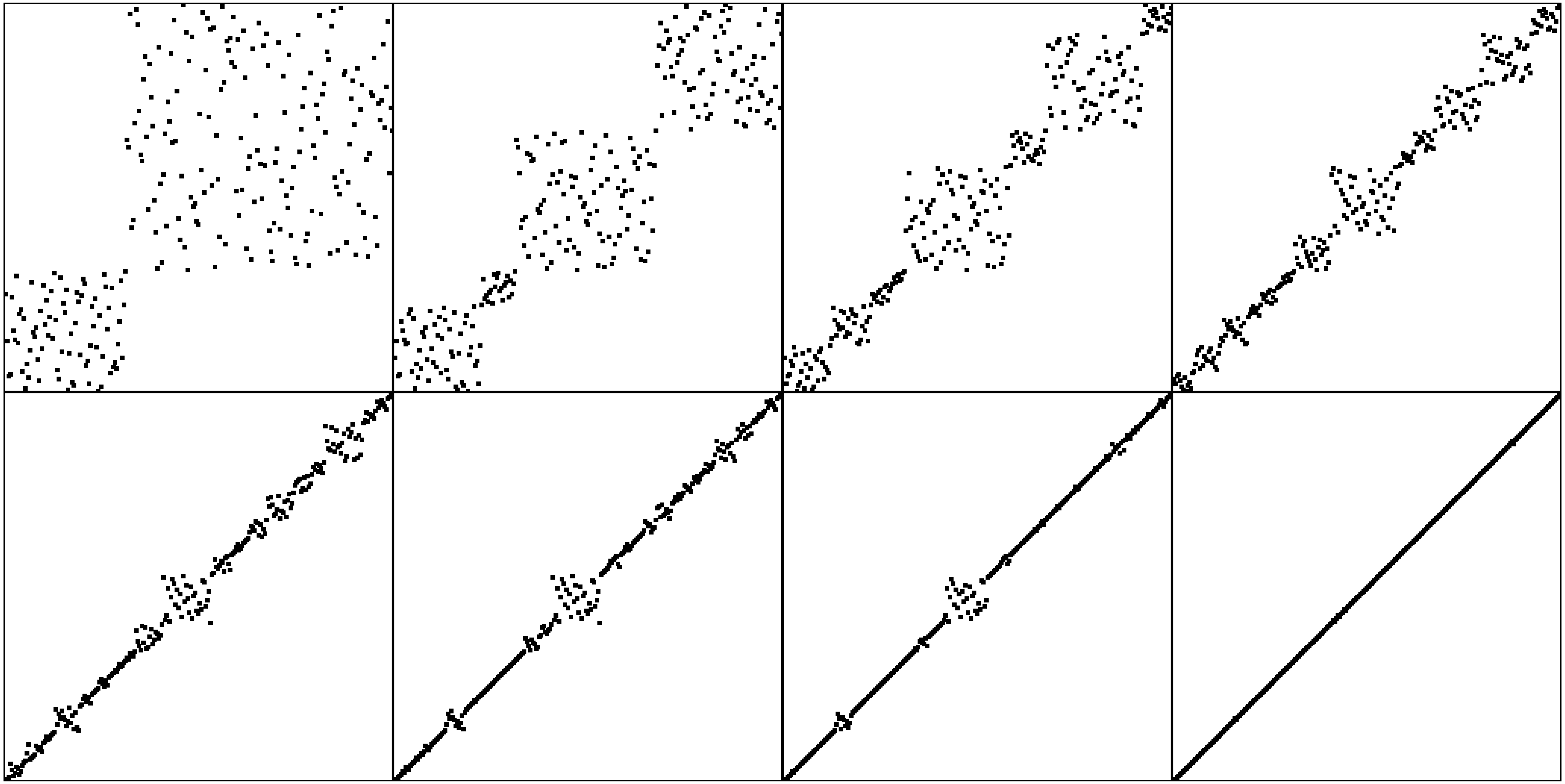


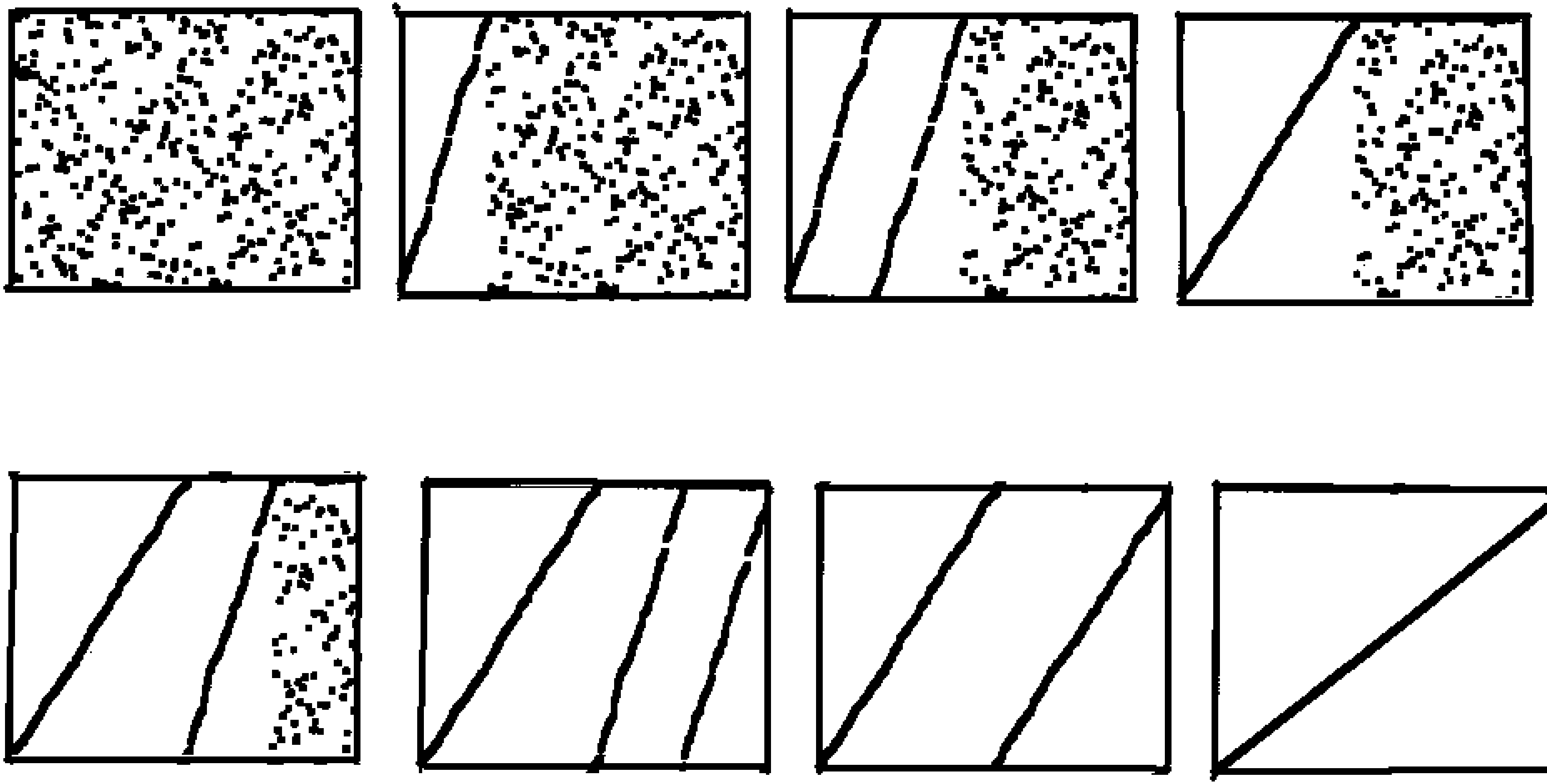
MaxSort



From: [LW95] Kenneth Lambert and Thomas Whaley, *An Invitation to Computer Science Laboratory Manual*, West Publishing Company, 1995. Conference, 12:5 (1997) 57—70.

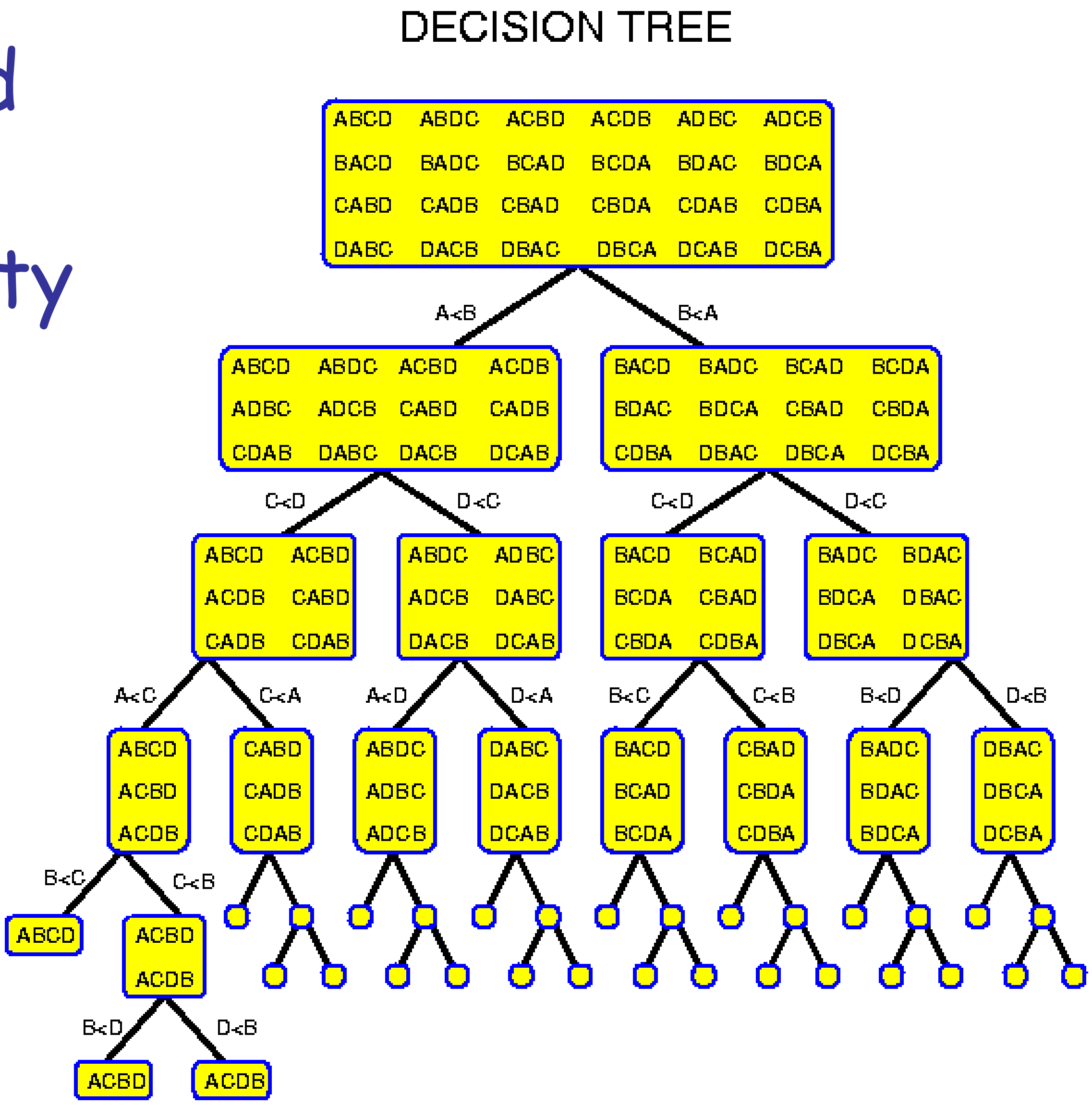
Quicksort





Mergesort

A Lower Bound on the Worst Case Complexity for Sorting



The Comparison Model:

The problem: Sort n integers.

Operations permitted on the data: comparisons and swaps.

It's very hard to prove good lower bounds for algorithm time complexities.

An easy lower bound for sorting is that any algorithm must take time which is $\Omega(n)$ because if the algorithm does not examine all the data items, then an adversary can change the value of an unexamined data item and make the answer wrong.

We can do better:

Theorem:

For the comparison model, any sorting algorithm requires at least $\Omega(n \log n)$ time in the worst case.

This theorem cannot be beat in the Big Oh sense because we have algorithms which take time in $O(n \log n)$ in the worst case which means it is a tight lower bound.

1. Sort these words in lexicographic order:

eat

either

earn

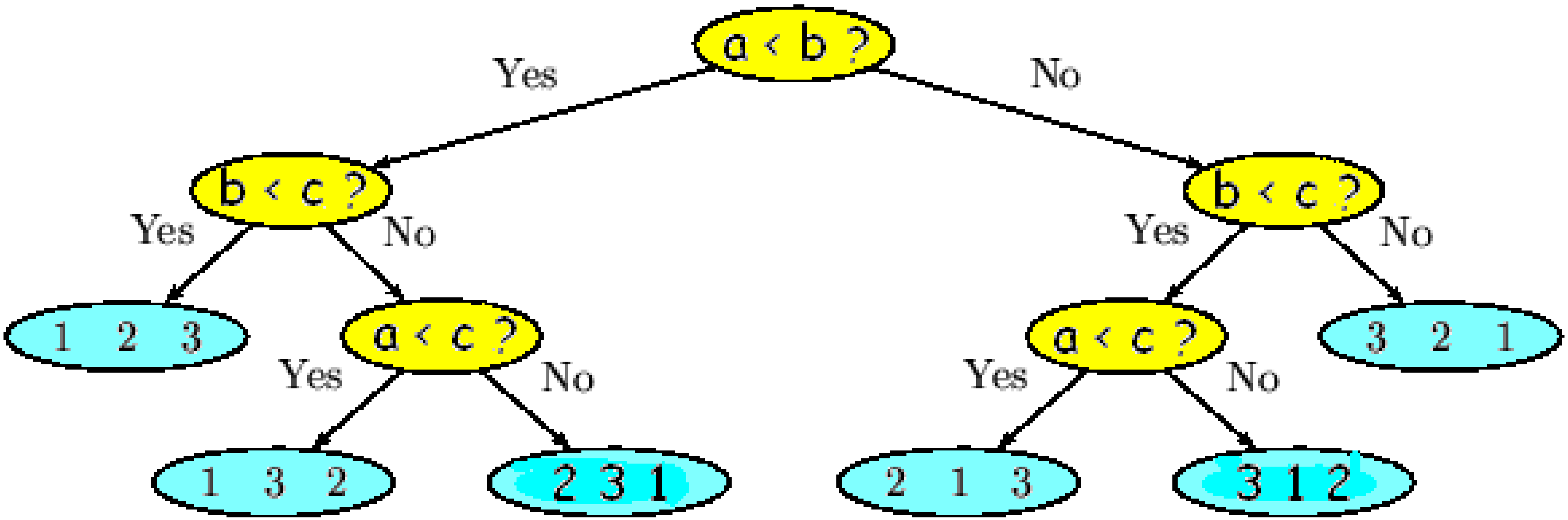
eaten

2. Write down a definition of lexicographic order.

The permutations on 4 symbols listed in lexicographic order (by columns):

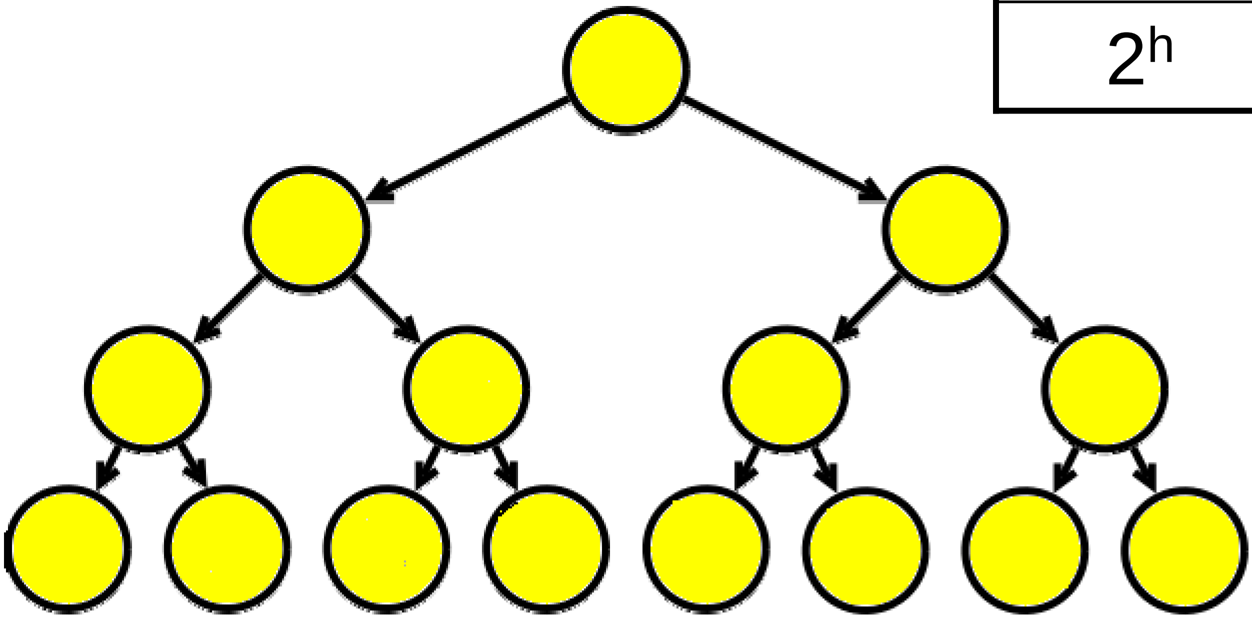
1 2 3 4	2 1 3 4	3 1 2 4	4 1 2 3
1 2 4 3	2 1 4 3	3 1 4 2	4 1 3 2
1 3 2 4	2 3 1 4	3 2 1 4	4 2 1 3
1 3 4 2	2 3 4 1	3 2 4 1	4 2 3 1
1 4 2 3	2 4 1 3	3 4 1 2	4 3 1 2
1 4 3 2	2 4 3 1	3 4 2 1	4 3 2 1

A Decision Tree: Input is a, b, c



Note that a complete binary tree which has r leaves has height $\theta(\log_2 r)$:

Leaves	Nodes	Height
?	1	0
2	3	1
4	7	2
8	15	3
	…	…
2^h	$2^{h+1} - 1$	h



We can use our tactics for lower and upper bounding to prove that:

$$\log_2(\, n! \,) \in \Theta(n \log_2 n)$$

Which sorting algorithms have optimal time complexities for the comparison model (in a Big Oh sense)?

These $\theta(n \log_2 n)$ in the worst case:

Heapsort, Mergesort, Mediansort

Not optimal since worst case is $\theta(n^2)$:

Quicksort, Maxsort, Binary Tree Sort

Draw the decision tree corresponding to:

```
if (A[1] < A[2])
{
   if (A[0] > A[1])
   {
      if (A[0] > A[2])
         swap(A[0], A[2])
      swap(A[0], A[1])
   }
}
else
{
   if (A[0] > A[1])
      swap(A[0], A[2])
   else
   {
      if (A[0] > A[2])
         swap(A[0], A[2])
      swap(A[1], A[2])
   }
}
```