Sorting

Swap Sorts: Repeatedly scan input, swapping any out-of-order elements.

Inversions of an element: the number of smaller elements to the right of the element.

The sum of inversions for all elements is the number of swaps required by bubblesort.

ANY algorithm that removes one inversion per swap requires at least this many swaps.

What is the worst number of total inversions?

Heapsort

Heap: complete binary tree with the value of any node at least as large as its two children.

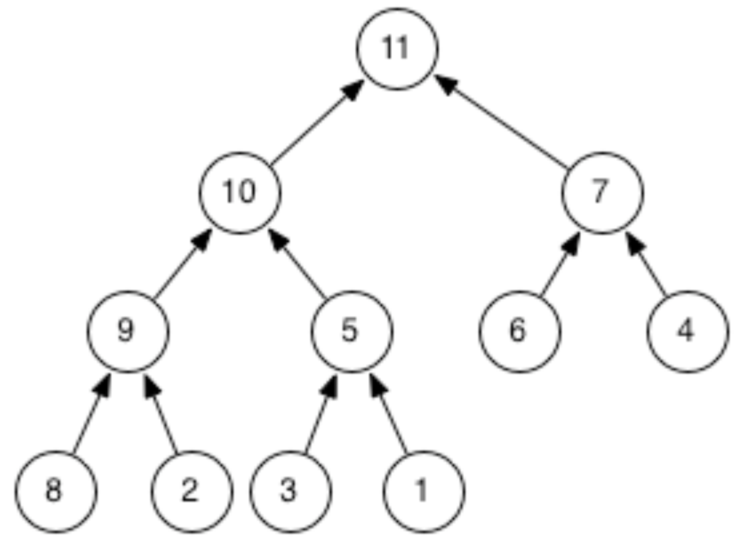
Algorithm:

Build the heap.

Repeat n times:

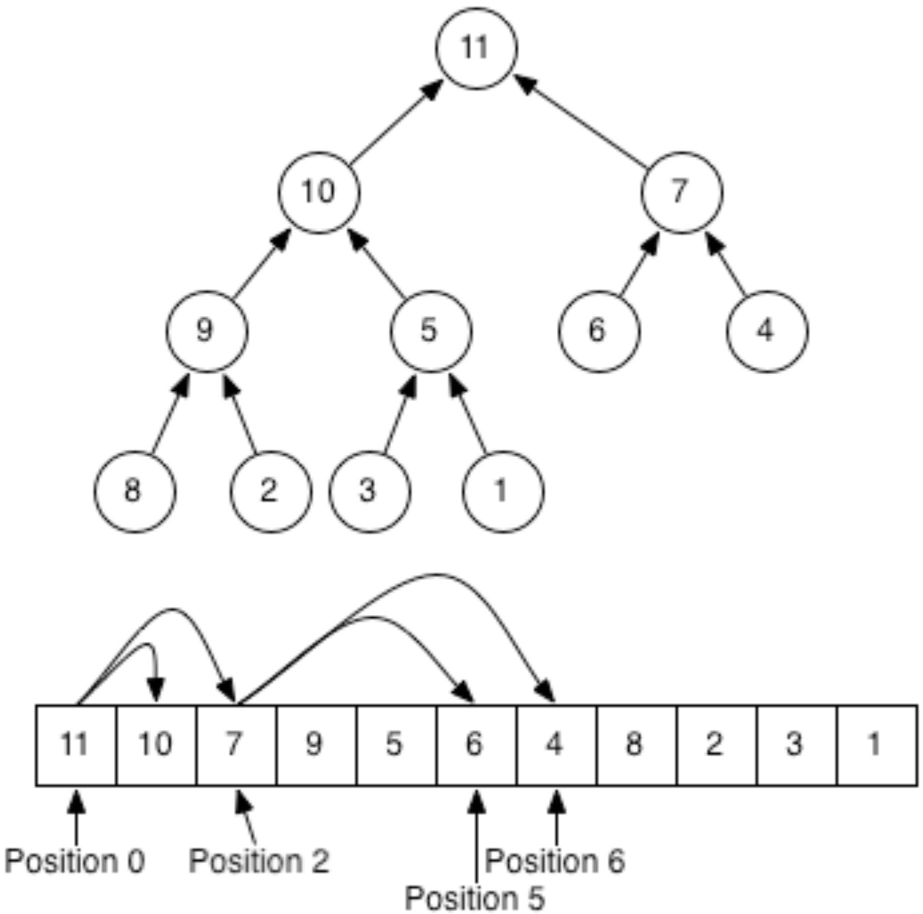
Remove the root.

Repair the heap.



Since the heap is a complete binary tree, it can be stored in an array. How can we store this heap?

Heapsort



What are the functions:
leftChild(i)
rightChild(i)
parent(i)
element at position i?

Heapsort

Algorithm:

Build the heap.

Repeat n times:

Remove the root.

Repair the heap.

Cost?

Quicksort

Algorithm:

Pick a pivot value.

Split the array into elements less than the pivot and elements greater than the pivot.

Recursively sort the sublists.

Quick Sort in Haskell:

```
quicksort [] = []
quicksort (x:xs) = quicksort small ++ (x :
quicksort large)
        where small = [y | y <- xs, y <= x]
              large = [y | y <- xs, y > x]
```

What is the cost of the split?

Quicksort

Algorithm:

Pick a pivot value.

Split the array into elements less than the pivot and elements greater than the pivot.

Recursively sort the sublists.

Quick Sort in Haskell:

```
quicksort [] = []
quicksort (x:xs) = quicksort small ++ (x :
quicksort large)
        where small = [y | y <- xs, y <= x]
              large = [y | y <- xs, y > x]
```

What is the worst case for picking pivot value?

Quicksort Average Cost

Algorithm:

Pick a pivot value.

Split the array into elements less than the pivot and elements greater than the pivot.

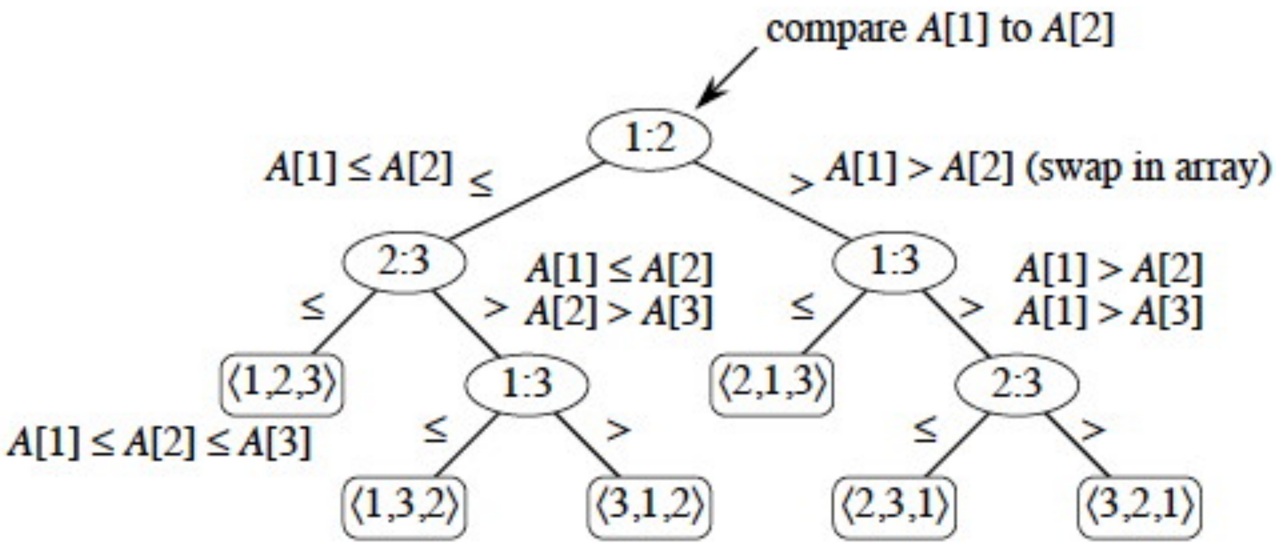
Recursively sort the sublists.

$$f(n) = \begin{cases} 0 & n \leq 1 \\ n - 1 + \frac{2}{n}\sum_{i=0}^{n-1} f(i) & n > 1 \end{cases}$$

Can you tell the order of growth?
Want to crank through the math?

Lower Bound on Sorting

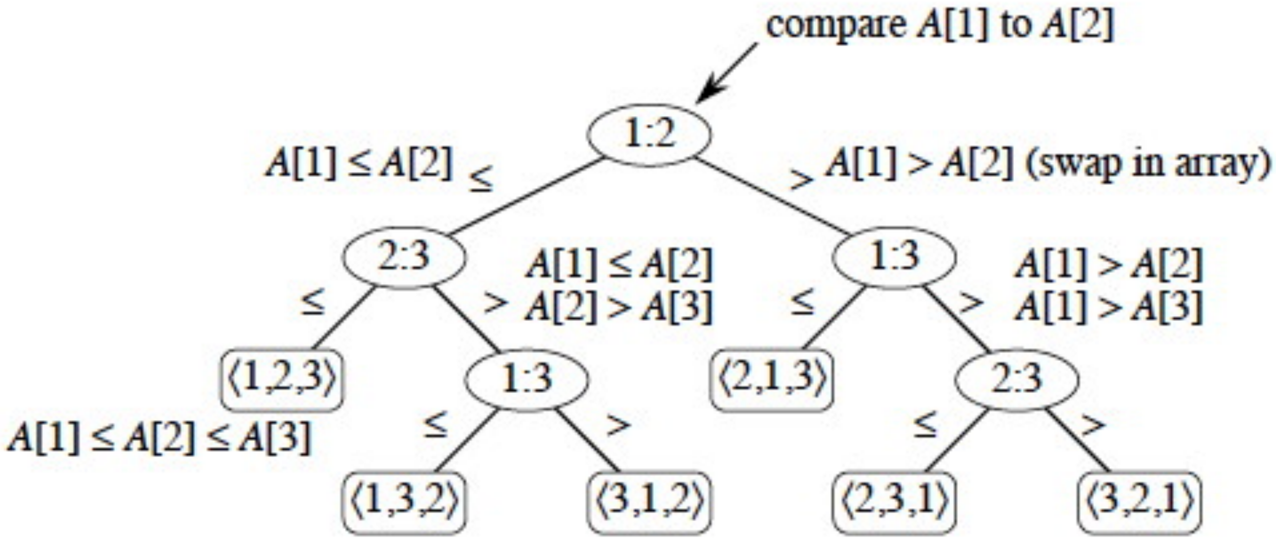
Look at decision tree to get lower bound on number of comparisons:



How many leaves are there in the decision tree if we are sorting n elements?

Lower Bound on Sorting

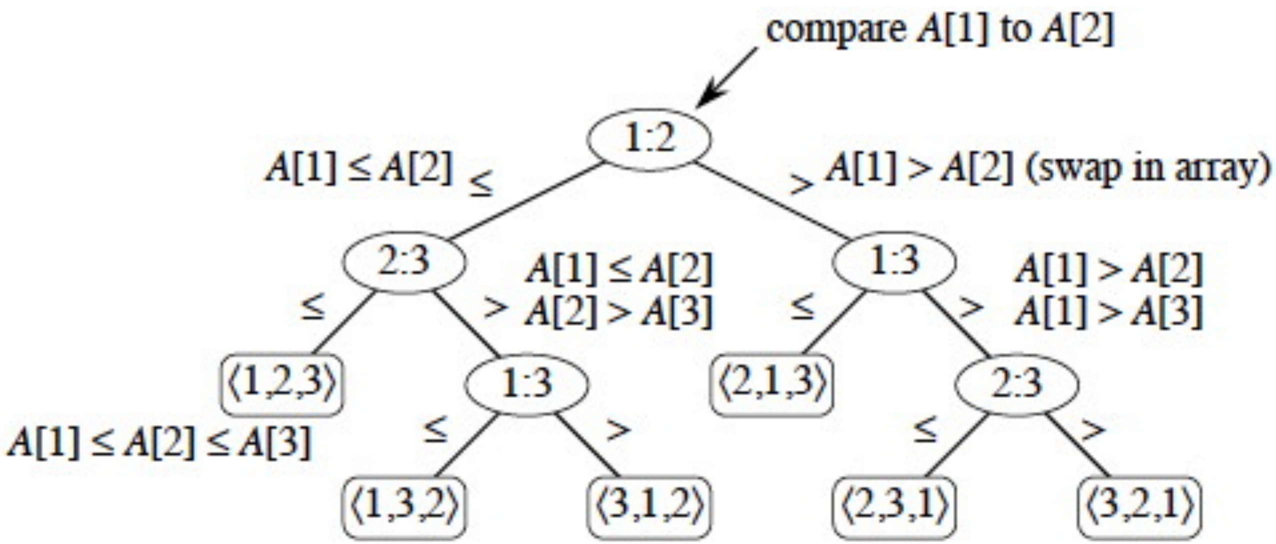
Look at decision tree to get lower bound on number of comparisons:



Any binary tree of height h as $k<2^h$ leaves.
Prove this by induction.

Lower Bound on Sorting

Look at decision tree to get lower bound on number of comparisons:



So a binary tree with n! leaves must have height at least ________.

Lower Bound on Sorting

A decision tree for sorting has height at least log(n!).

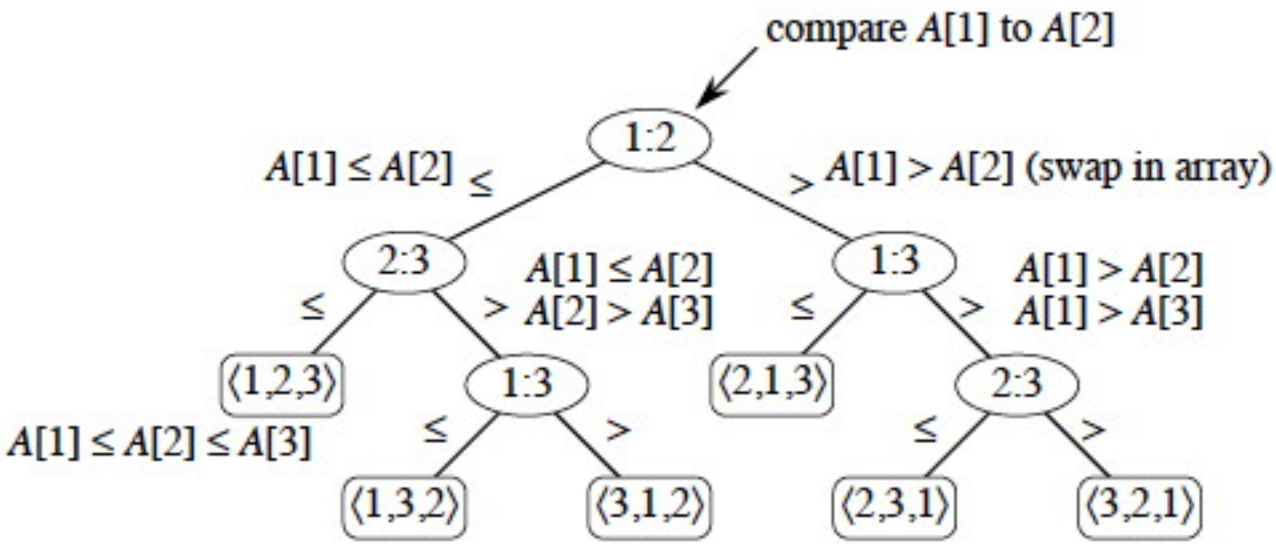
$$\frac{n}{2}^{\frac{n}{2}} \leq n! \leq n^n$$

So log(n!) is O(n log n)

Why?

Lower Bound on Sorting

Look at decision tree to get lower bound on number of comparisons:



If the height of any decision tree for sorting is O(n log n) then why is the time for sorting at least O(n log n)?

Changing the Analysis Model

If we know something about the structure of the data, it is possible to sort it without comparing elements to each other.

Counting Sort

Requires that keys to be sorted are integers in $\{0, 1, \ldots k\}$.

For each element in the input, determines how many elements are less than that input. Then we can place the element directly in a position that leaves room for the elements that come after it.

What is the order of growth?