

## Why don't CS profs *ever* stop talking about sorting?!

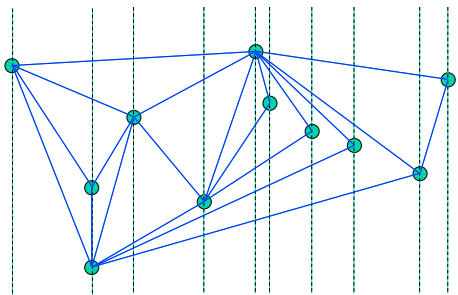
- Computers spend more time sorting than anything else, historically 25% on mainframes.
- Sorting is the best studied problem in computer science, with a variety of different algorithms known.
- Most of the interesting ideas we encounter in the course are taught in the context of sorting, such as divide-and-conquer, randomized algorithms, and lower bounds.

You should have seen most of the algorithms - we will concentrate on the analysis

## Applications of Sorting

- Searching *what else???*

## Convex Hulls



## Huffman Codes

If you are trying to minimize the amount of space a text file is taking up, it is silly to assign each letter the same length (i.e. one byte) code.

**Example:** *e* is more common than *q*, *a* is more common than *z*.

If we were storing English text, we would want *a* and *e* to have shorter codes than *q* and *z*.

## Selection Sort

A simple  $O(n^2)$  sorting algorithm

```

Selection-sort(A)
  for i = 1 to n
    for j = i+1 to n
      if (A[j] < A[i]) then swap(A[i],A[j])
  
```

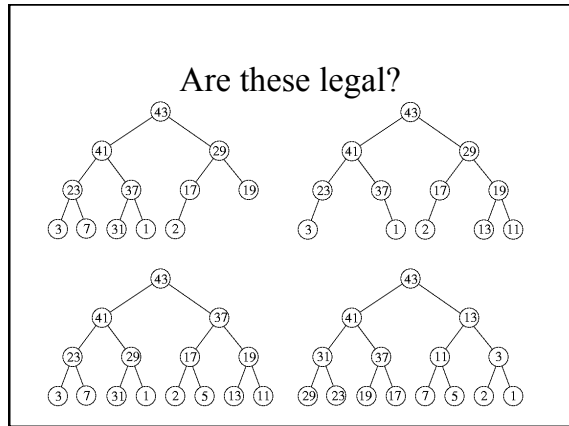
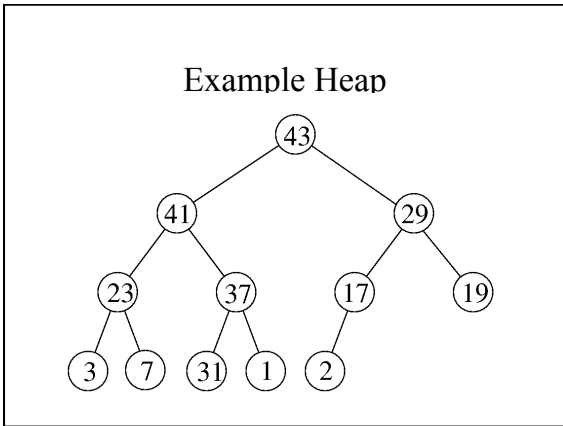
It is clear this algorithm must be correct from an inductive argument, since the  $i^{\text{th}}$  element is in its correct position

## Binary Heaps

A *binary heap* is defined to be a binary tree with a key in each node such that:

- 1: All leaves are on, at most, two adjacent levels.
- 2: All leaves on the lowest level occur to the left, and all levels except the lowest one are completely filled.
- 3: The key in root is greater than all its children, and the left and right subtrees are again binary heaps.

Conditions 1 and 2 specify shape of the tree, and condition 3 the labeling of the tree.



### Properties of Heaps

The ancestor relation in a heap defines a *partial order* on its elements, which means it is reflexive, anti-symmetric, and transitive.

*Reflexive*:  $x$  is an ancestor of itself.

*Anti-symmetric*: if  $x$  is an ancestor of  $y$  and  $y$  is an ancestor of  $x$ , then  $x=y$ .

*Transitive*: if  $x$  is an ancestor of  $y$  and  $y$  is an ancestor of  $z$ ,  $x$  is an ancestor of  $z$ .

Partial orders can be used to model hierarchies with incomplete information or equal-valued elements.

### Constructing Heaps

- Heaps can be constructed incrementally, by inserting new elements into the left-most open spot in the array.
- If the new element is greater than its parent, swap their positions and recur.

The height  $h$  of an  $n$  element heap is bounded because:

$$\sum_{i=0}^h 2^i = 2^{h+1} - 1 \geq n$$

so,  $h = \lfloor \log n \rfloor$  and insertions take  $O(\log n)$  time

### Heapify

The bottom up insertion algorithm gives a good way to build a heap, but Robert Floyd found a better way, using a *merge* procedure called *heapify*.

Given two heaps and a fresh element, they can be merged into one by making the new entry the root and trickling down.

To convert an array of integers into a heap, place them all into a binary tree, and call heapify on each node.

How long would this take?

### Heapify Example

Try to create a heap with the entries:

5, 3, 17, 10, 84, 19, 6, 22, 9

## Heap Extract Max

```
if heap-size(A) < 1
  then error "Heap Underflow";
max = A[1];
A[1] = A[heap-size(A)];
heap-size(A)--;
Heapify(A, 1);
return max;
```

## Heap Sort

To sort using the heap data structure, we first build the heap, and then just repeatedly extract the maximum.

Build Heap =  $O(n)$

Extract Maximum =  $O(\log n)$

Therefore:

Heap Sort =  $O(n) + n O(\log n)$   
=  $O(n \log n)$