Lecture 7, Tuesday 4/24/01

Sorting using heaps

- We can first build heap, then repeat: remove max.
- In place:

```
BuildHeap
for i=n down to 2
   exch A(1), A(i)
   Heapify(A,1,i-1)
```

- Essentially same as the first approach.
  We use the fact that:
  - » heap becomes smaller after "remove max",
  - » last array entry becomes free.

Example

- Sort:

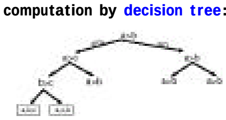
```
     8      exchange      1       heapify       7       exchange
   5    7              5    7                5     1
 3  2  1  ?          3  2  8  ?            3  2  8  ?

     2                    5
   5    1   heapify    3     1
 3  7  8  ?          2  7  8  ?
```


Variations on Heaps

- Min instead of max.
- K-ary instead of binary
  - » Time for Insert: $\log_k n$ (n = #elems. in heap).
  - » Time for extract-max: $k \cdot \log_k n$
  - » Best value of k to use is determined by application:
    - Mostly inserts: use big k (e.g. $k = \sqrt{n}$)
    - Mostly extract-max: use small k (i.e. k=2 or 3)

Lower bound for sorting

- All sorting algs that we saw : **comparison-sorts**
  only operation allowed on data is comparison.
- Is **O(n log n)** the best we can do in this case ?
- Represent computation by **decision tree**:



- Execution - walk from root to a leaf.

More lower bound

- 1 leaf per each possible answer.
  n! different answers at least **n!** leaves.
- Binary tree with **n!** leaves has to be Ω**(n log n)** deep !
  worst-case execution time is Ω**(n log n)**
- In the comparison model, quicksort, mergesort, etc are optimum.
- HW: Why doesn't this work for selection ??
  What if instead of sorting, we need to divide into groups of, say, 10, and sort the groups
  (all element of 1st group < all elements of 2nd group, etc)

Counting Sort

- Is Ω**(n log n)** indeed the limit ?? NO !
- Example: Counting Sort
  Assume inputs are in [1,…,k], integers.

```
for i=1 to k   C(i)=0
for j=1 to n   C(A(j))++
compute prefix sum C(i)=C(i)+C(i-1)  for all i=2 to k
put element x into B(C(x)), C(x)--,  for all x in A().
```

- Example: Input 1,1,5,5,7
  C(1)=2, C(5)=2, C(7)=1.
  After prefix sum: C(1)=2, C(5)=4, C(7)=5
  in particular, first 5 goes into 4th position, as it should.
- Read **Radix sort:** note the stability of intermediate sort requirement !