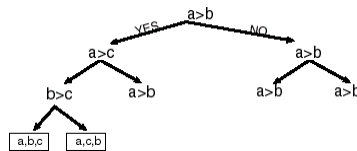


## Lower bound for sorting

- 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.

90

## More lower bound

- 1 leaf per each possible answer.  
 $n!$  different answers  $\rightarrow$  at least  $n!$  leaves.
- Even complete binary tree with  $n!$  leaves has to be  $\Omega(n \log n)$  deep!  
 $\rightarrow$  worst-case execution time is  $\Omega(n \log n)$
- In the comparison model, heapsort, 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)  
What if groups are  $n/10$  size each ?

91

## Counting Sort

- Is  $\Omega(n \log n)$  indeed the limit in all cases?? **Not true !!!**
- Example: Counting Sort  
Assume inputs are in  $[1, \dots, k]$ , 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 interm. sort requirement !

92

## Hashing

- Heaps support:
  - » Insert
  - » Delete
  - » Max/min
- How about [Search](#) ?  
(How would you implement “find” in a heap ?)
- Possible solutions:
  - » Ordered array: slow “insert” -  $\Omega(n)$ .
  - » Ordered list: both find and insert are slow !  
(can we do binary search in a list ?)

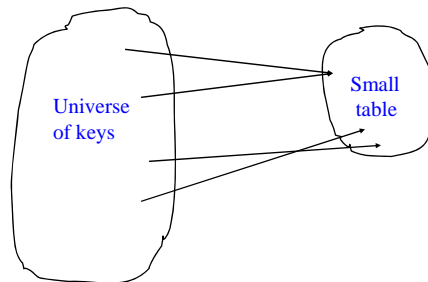
93

## Direct address table

- Maintain table  $T[i]$ :  $T[i] = \begin{cases} x & \text{if } x \in T, \text{key}(x) = i \\ \emptyset & \text{otherwise} \end{cases}$

- Disadvantage: **too much memory!**

- Idea: maintain small table:



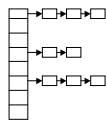
94

## Collisions

- $\text{size}(\text{Table}) < \text{size}(\text{Universe of keys}) \Rightarrow \text{Collisions!}$   
(collision = two keys map into the same slot in T)

- How to resolve collisions:

- » Chaining:



What is "next"?

- » Open addressing: if  $A[h(x)]$  full - try **next** slot.

95