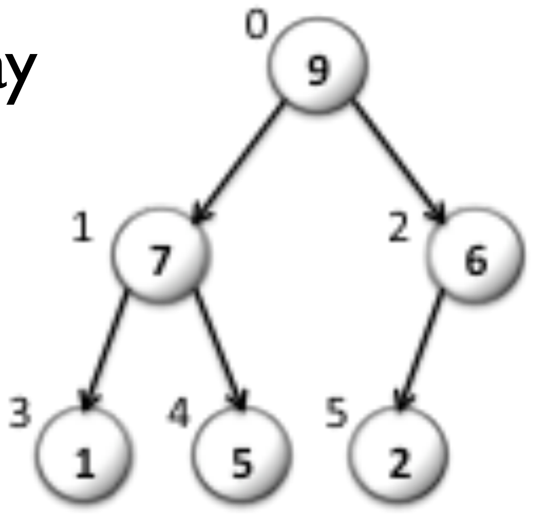
CS32 Discussion
Section 1B
Week 9

TA: Zhou Ren

Heaps

- A **heap** is a
  - complete binary tree
  - every node carries a value greater than or equal to its children's (maxHeap).
  - usually implemented as an array

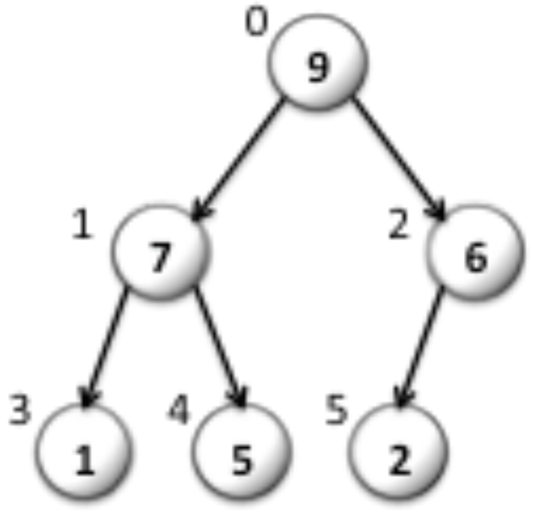
9	7	6	1	5	2



Heaps: operations

- 3 operations for heaps
  - findMax (search)
  - insertNode (insert)
  - deleteMax (remove)

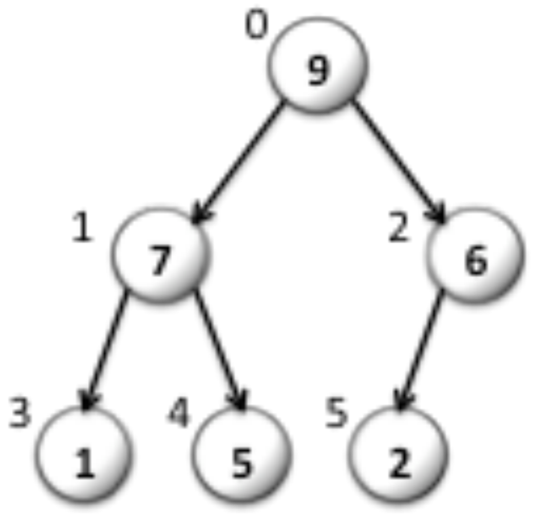
9	7	6	1	5	2



findMax

- What do you think?

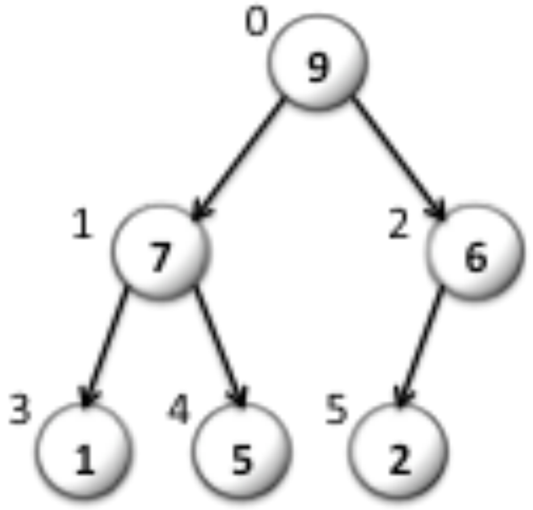
9	7	6	1	5	2



insertNode

- Not so trivial
- We first add the new node and fix it

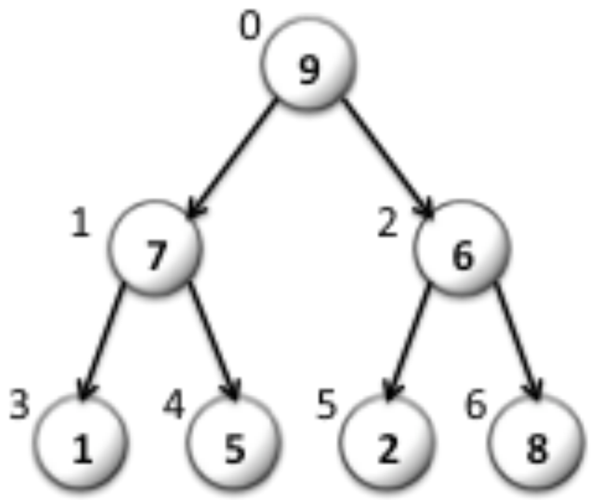
9	7	6	1	5	2



insertNode

1. Add the new node to the tail.
2. Ask:
   - Is the new value greater than its parent?
   - If so: ??
   - Else: ??

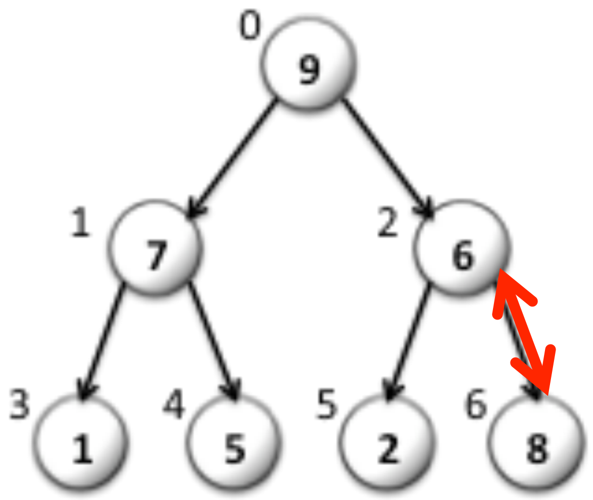
9	7	6	1	5	2	8



insertNode

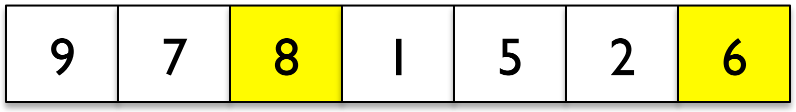
1. Add the new node to the tail.
2. Ask:
   - Is the new value greater than its parent?
   - If so: **swap**
   - Else: **done**

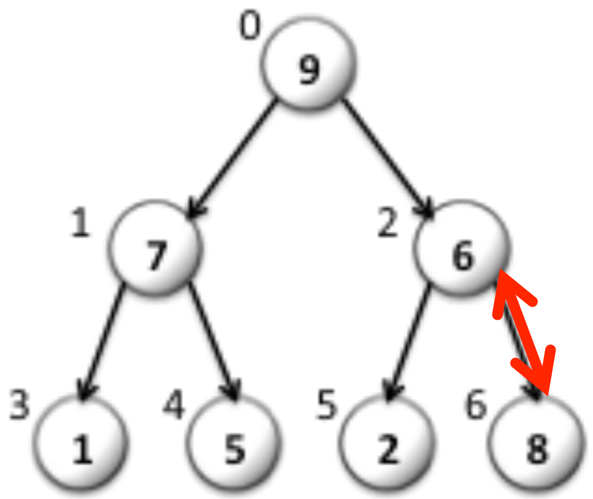
9	7	8	1	5	2	6



insertNode

- What is the index of node i's parent in the array?

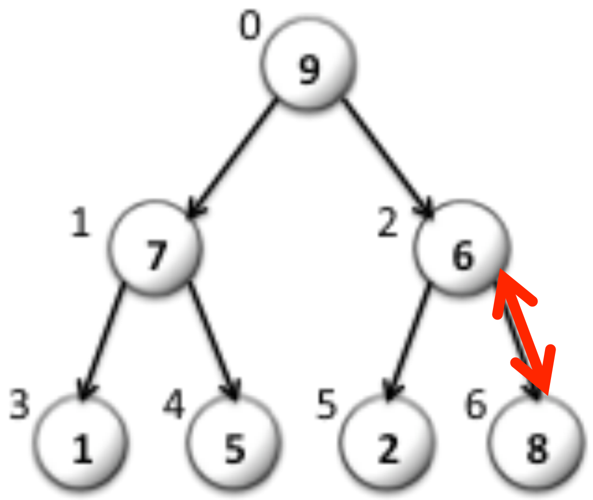




insertNode

- What is the index of node i’s parent in the array?
  - parent = (i - 1) / 2

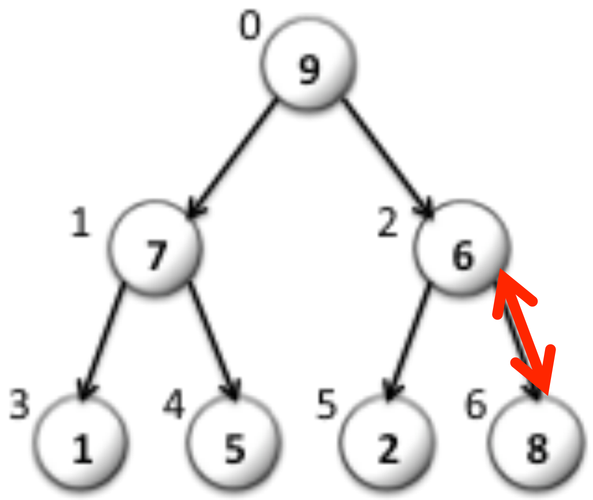
9	7	8	1	5	2	6



insertNode

- Running time?

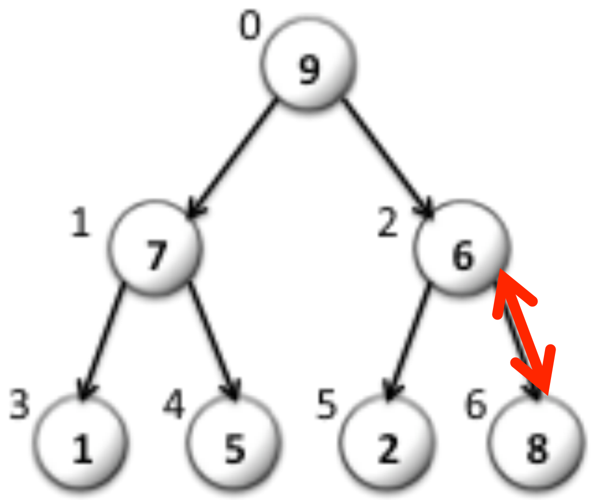
9	7	8	1	5	2	6



insertNode

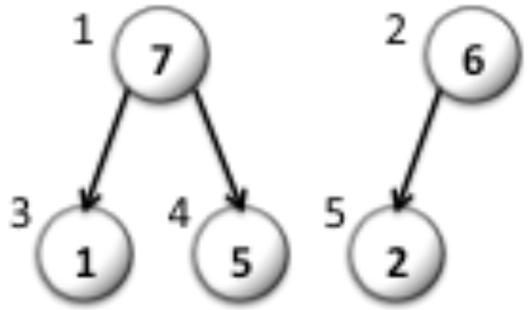
- Running time?
  - proportional to the height of the tree: **O(log n)**

9	7	8	1	5	2	6

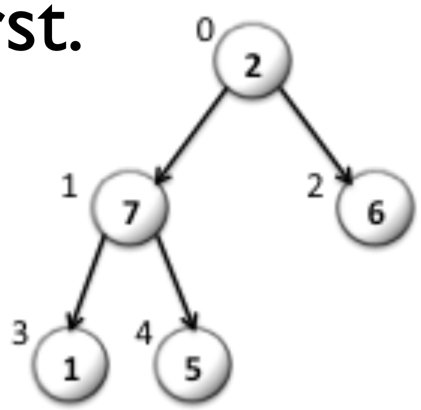


deleteMax

- Again, take the action first and fix it.



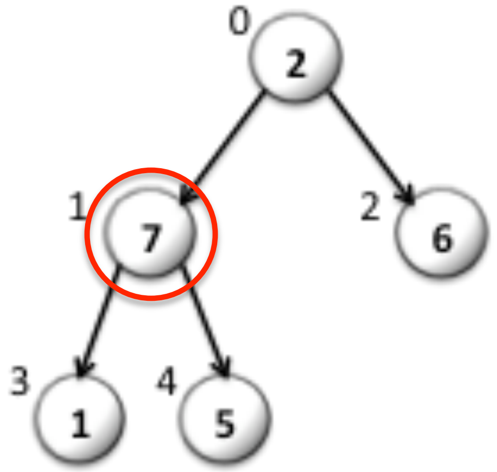
- Fill in the void first.



deleteMax

- Now compare the values of the two children, take the greater of the two (why?), and swap.
- What are the indices of
  - Left child:
  - Right child:

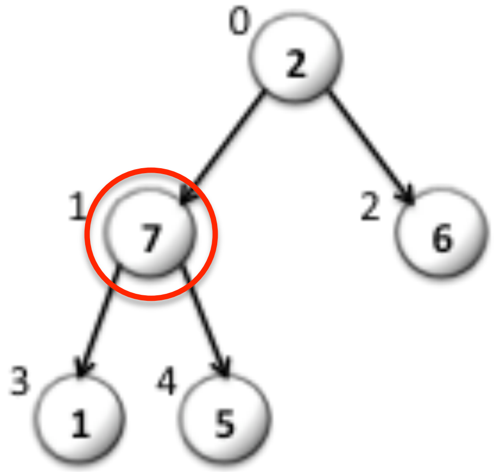
  of the node i?



deleteMax

- Now compare the values of the two children, take the greater of the two (why?), and swap.
- What are the indices of
  - Left child: 2 * i + 1
  - Right child: 2 * i + 2

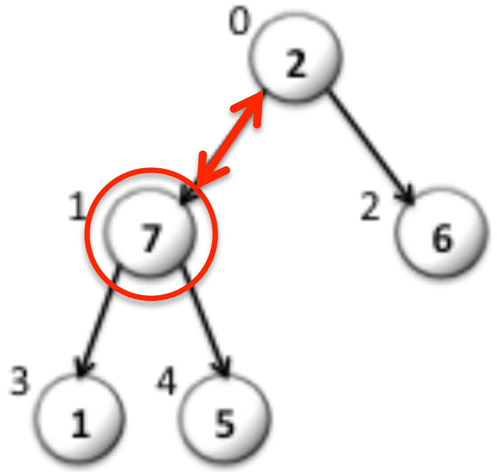
  of the node i?



deleteMax

- Now compare the values of the two children, take the greater of the two (why?), and swap.
- What are the indices of
  - Left child: 2 * i + 1
  - Right child: 2 * i + 2

  of the node i?



Heapsort

- Can you use a heap to sort a set of elements?

Heapsort

- Can you use a heap to sort a set of elements?
  - Insert all elements into a heap
  - Extract the maximum element from the heap one by one

Heapsort

- Can you use a heap to sort a set of elements?
  - Insert all elements into a heap
  - Extract the maximum element from the heap one by one
- Running time?

Heapsort

- Can you use a heap to sort a set of elements?
  - Insert all elements into a heap
  - Extract the maximum element from the heap one by one
- Running time?

  Inserting n items: n x O(log n) = O(n log n)

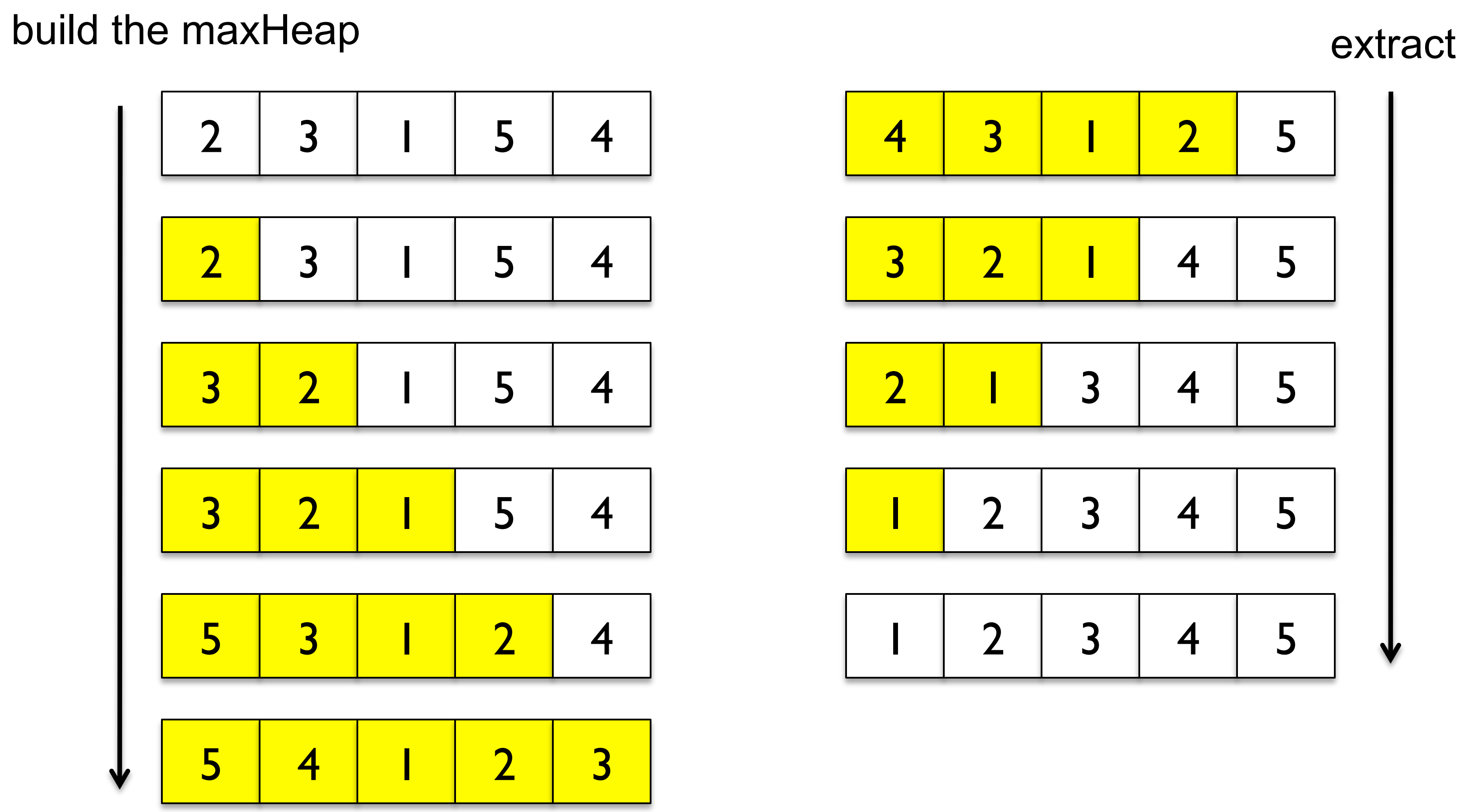
  Extracting n items: n x O(log n) = O(n log n)

  O(n log n) + O(n log n) = **O(n log n)**

In-place Heapsort

- Heapsort is an in-place sorting algorithm – you don’t need an auxiliary structure for the sorting operation.
- Let us try using a **maxHeap** to sort the elements in an array in the **increasing order**.

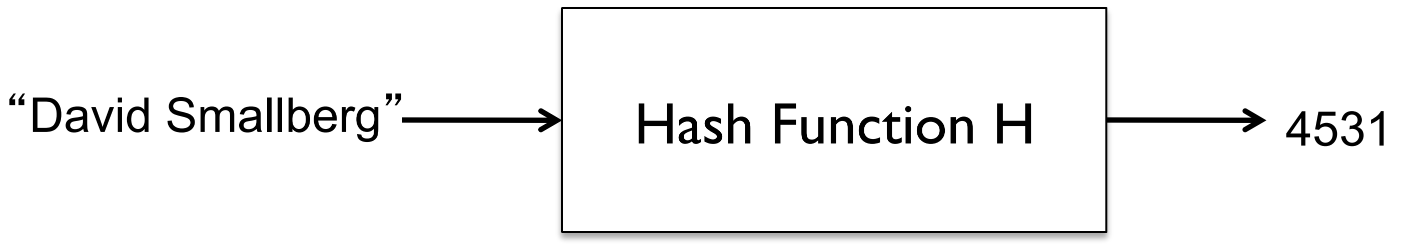
In-place Heapsort



🟨 part of the maxHeap

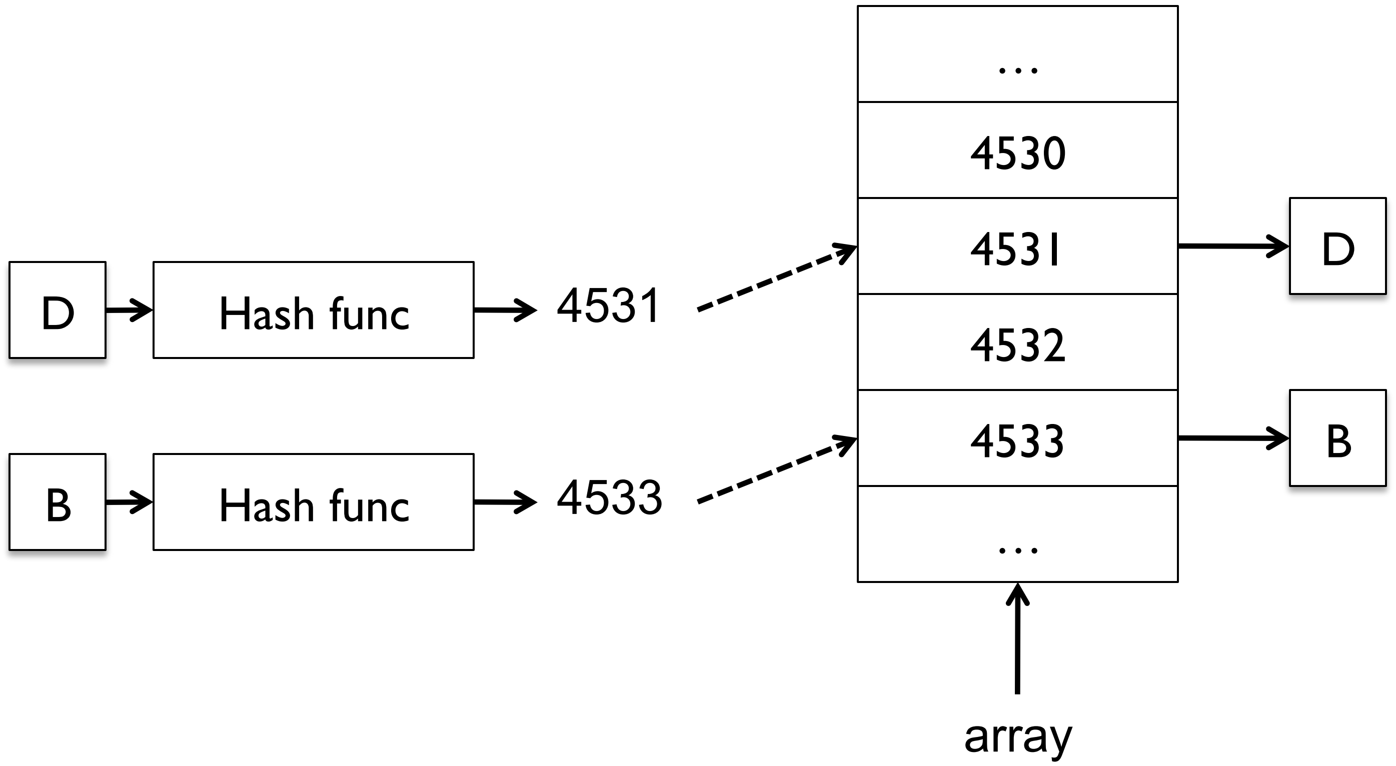
Hash Functions

- Hashing
  - Take a “key” and map it to a number

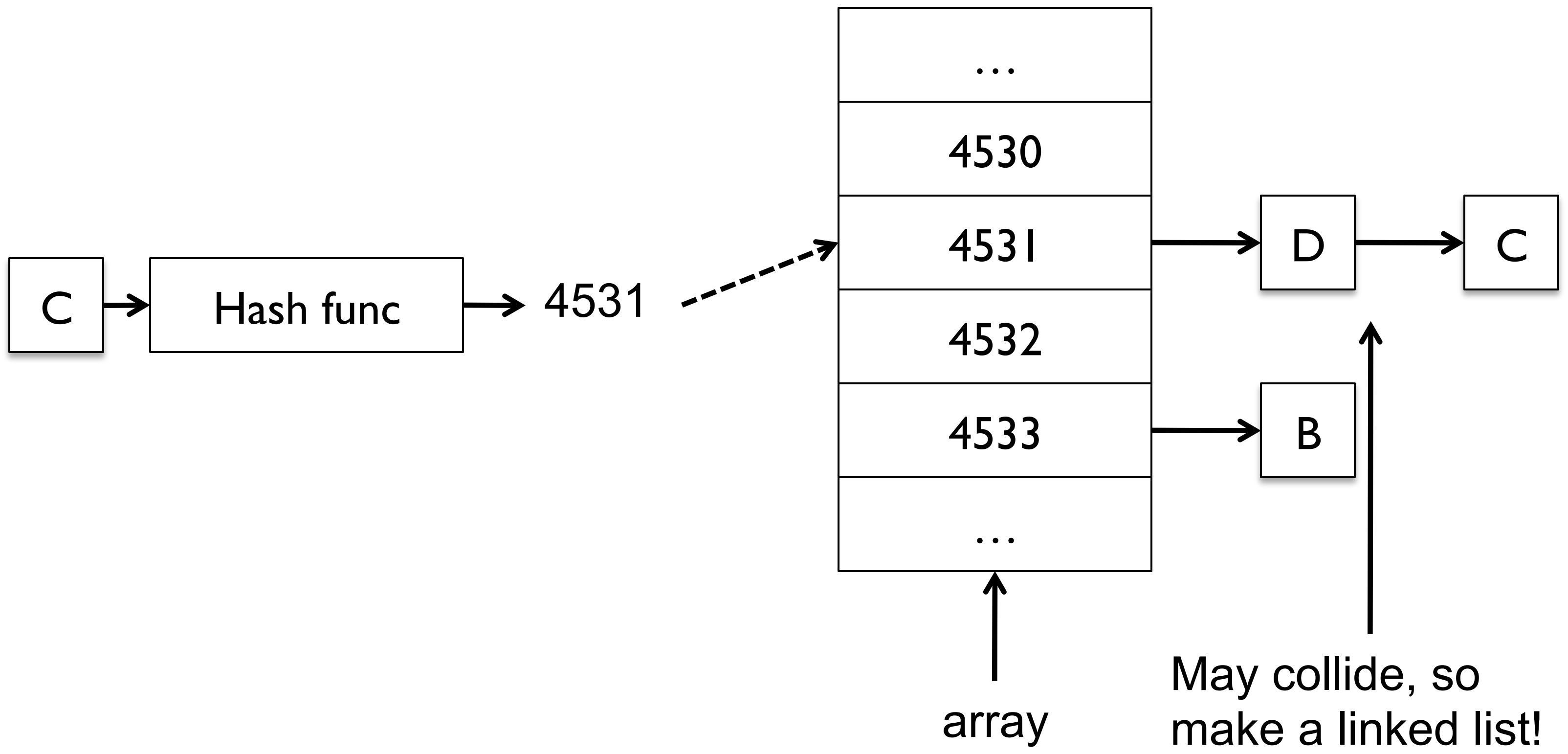


- A requirement for hash function H: should return the same value for the same key.
- A good hash function
  - spreads out the values: two different keys are likely to result in different hash values
  - computes each value quickly

Hash Table



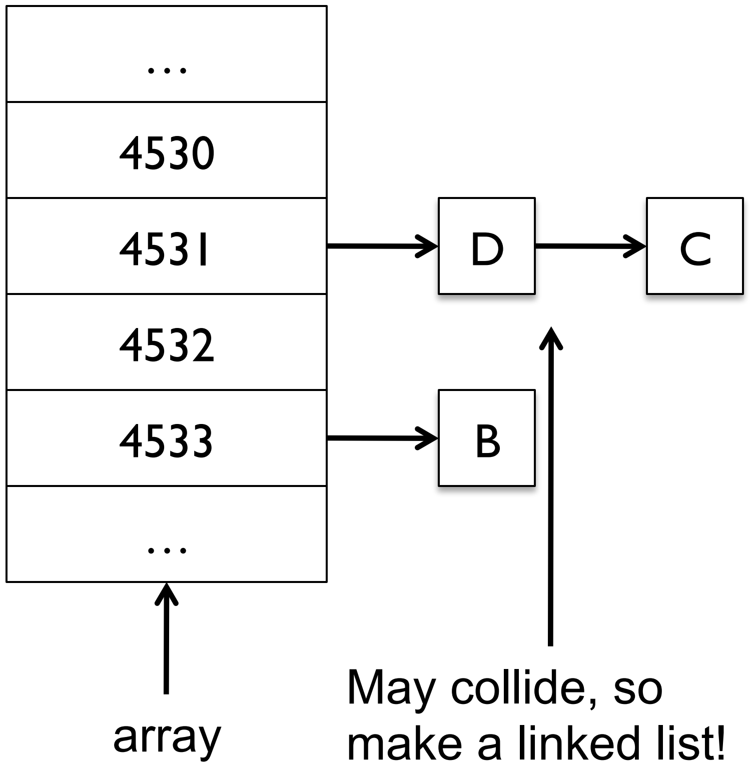
Hash Table



C
Hash func
4531
...
4530
4531
4532
4533
...
D
C
B
array
May collide, so make a linked list!

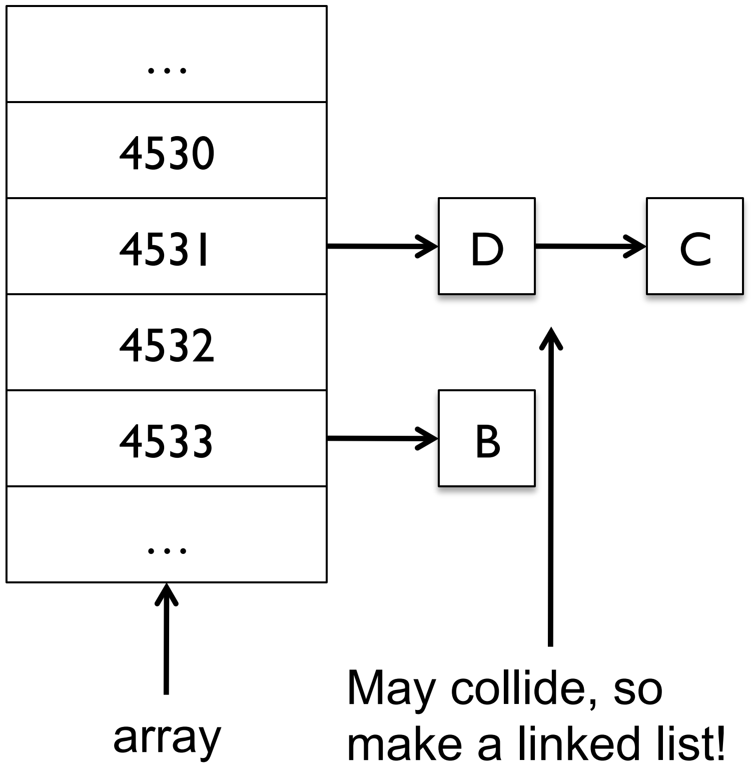
Hash Table

- Running time
  - Insert?
  - Remove?
  - Search?



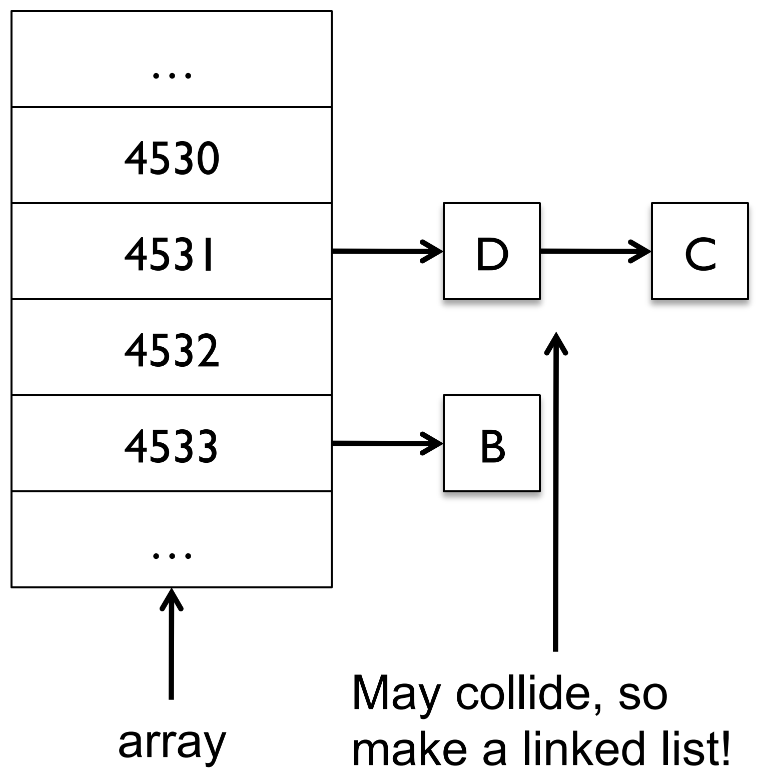
Hash Table

- Running time
  - Insert? **O(1)**
  - Remove? **O(1)**
  - Search? **O(1)**



Hash Table

- Running time
  - Insert? **O(1)**
  - Remove? **O(1)**
  - Search? **O(1)**
- Looks great, but what are the limitations?


…
4530
4531
4532
4533
…
D
C
B
array
May collide, so make a linked list!