Last Lectures

- We saw how to ...
  - Understand how tree sort works
  - Sort an array using tree sort
  - Analyze time/space efficiency of tree sort
Learning Outcomes

At the end of the next few lectures, a student will be able to:

- Define the following data structures:
  - Binary search tree
  - Balanced binary search tree (AVL)
  - Binary heap
  as well as demonstrate and trace their operations
- Implement the operations of binary search tree and binary heap
- Implement and analyze sorting algorithms: tree sort and heap sort
- Write recursive solutions to non-trivial problems, such as binary search tree traversals
Today’s menu

- Our goal is to
  - Define binary heaps
  - Demonstrate and trace their recursive operations
Array-based implementations of Binary Tree

- There is another flavour of Binary Tree that can be efficiently implemented using an array-based implementation
A complete binary tree can be implemented efficiently using an array since there are no gaps in such tree.
Maximum Binary Heap

- Maximum Binary Heap is a ...
  - complete binary tree, and
  - the key value of a node in such heap is $>$ or $\geq$ to key value of its children (if any), and
  - the node’s left and right subtrees are also maximum binary heaps

- In such a heap, the root contains the element with the largest key value
Example

Maximum Binary Heap

Example:

Complete?
Max Heap?

Root = Largest key value.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Minimum Binary Heap

- Minimum Binary Heap is a …
  - complete binary tree, and
  - the key value of a node in such heap is \(<\) or \(\leq\) to key value of its children (if any), and
  - the node’s left and right subtrees are also minimum binary heaps

- In such a heap, the root contains the element with the smallest key value
Example
Heap operations

- Insert
- Remove
  - Remove the root
- Retrieve
  - Behaves like Min( ) or Max( ) of BST
Note about the tracing of Binary Heap algorithm

- In our lecture notes, all binary heap algorithms (insert, remove and sort) manipulate an array since a binary heap is implemented using an array as its underlying data structure.
- This is the reason why we often draw an array as we are tracing the execution of the algorithm.
- However, as we are tracing the execution of the algorithm, we also often draw the (conceptual) tree representation of the binary heap.
- The reason we do this is because it is often easier to visualize how the algorithm executes when the binary heap is represented as a tree.
- However, understand that the algorithm does not create the tree representation of the binary heap, hence it does not require additional memory space for the tree.
Insertion into a Min Binary Heap

Algorithm:

indexOfRoot = 0
indexOfBottom = elementCount
insert new element @ "bottom" of heap (@ indexOfBottom)
elementCount++
reHeapUp( indexOfBottom )

// If the element has a parent and ...
if ( indexOfBottom > indexOfRoot )

indexOfParent = floor((indexOfBottom - 1) / 2)
// ... key value of parent > key value of child then ...
if ( heap[indexOfParent] > heap[indexOfBottom] )

//... swap the element with its parent
swap heap[indexOfParent] with heap[indexOfBottom]
reHeapUp( indexOfParent )
Insertion – Let’s Try!

Insert ____ in Min binary heap:
Insertion – Let’s Try!

- Insert ____ in Min binary heap:

```plaintext
[211x345] 15 3 3 4 5 7
```
Insertion – Let’s Try!

- Insert ____ in Min binary heap:
Insertion – Let’s Try!

- Insert 5, 3, 2, 6, 0 into a Min binary heap:
Insertion into a Max Binary Heap

Algorithm:

indexOfRoot = 0
indexOfBottom = elementCount

insert new element @ "bottom" of heap (@ indexOfBottom)

elementCount ++

reHeapUp( indexOfBottom )

// If the element has a parent and ...

if ( indexOfBottom > indexOfRoot )

indexOfParent = floor((indexOfBottom - 1) / 2)

// ... key value of parent < key value of child then ...

if ( heap[indexOfParent] < heap[indexOfBottom] )

//... swap the element with its parent

swap heap[indexOfParent] with heap[indexOfBottom]

reHeapUp( indexOfParent )
Insertion – Let’s Try!

- Insert ____ in Max binary heap:
Insertion – Let’s Try!

- Insert ____ in Max binary heap:

```
7
5  7
4  3
```
Insertion – Let’s Try!

- Insert 5, 3, 2, 6, 0 into a Max binary heap:
Removal from a Min Binary Heap

Algorithm:

indexOfRoot = 0
Call retrieve( ) (optional)
  -> return element stored in root of heap (array[indexOfRoot])
Replace element stored in root with element stored at the bottom of heap i.e., last element
  -> copy element stored at the bottom (at index elementCount-1) into root of heap
  -> elementCount --
reHeapDown( indexOfRoot )
  if ( heap[indexOfRoot] is not a leaf )
    set indexOfMinChild to index of smallest child of root
    if ( heap[indexOfRoot] > heap[indexOfMinChild] )
      swap heap[indexOfRoot] with heap[indexOfMinChild]
      reHeapDown( indexOfMinChild )
Removal – Let’s Try!

- `indexOfRoot`:

- `indexOfMinChild`:

```
Removal from a Max Binary Heap

Algorithm:

indexOfRoot = 0
Call retrieve() (optional)

- return element stored in root of heap (array[indexOfRoot])

Replace element stored in root with element stored at the
bottom of heap i.e., last element

- copy element stored at the bottom (at index elementCount-1)
  into root of heap

- elementCount --

reHeapDown( indexOfRoot )

if ( heap[indexOfRoot] is not a leaf )
  set indexOfMaxChild to index of largest child of root
  if ( heap[indexOfRoot] < heap[indexOfMaxChild] )
    swap heap[indexOfRoot] with heap[indexOfMaxChild]
    reHeapDown( indexOfMaxChild )
Removal – Let’s Try!

- indexOfRoot:

- indexOfMaxChild:
Time Efficiency of Binary Heap’s operations

- Insertion and Removal ->
  - During ReHeapUp and ReHeapDown:
    - Visit each level of heap
    - Compare parent with child and swap if necessary
    - Therefore:

- Retrieve ->
  (Remember: Retrieve returns the element located at the root - Minimum or maximum)
What about Search?

- Can we efficiently search a heap as we do with a BST or an AVL?
- Why?
Searching a Binary Heap

- Considering a node in the binary heap, no sort ordering constraint dictates the position of its children. The only sort ordering constraint that exists in a binary heap is the rule that dictates the position of a child versus its parent.
Forte of Binary Heaps

- Finding the element with the largest/smallest key value
✓ Learning Check

- We can now ...
  - Define binary heaps
  - Demonstrate and trace their recursive operations
Next Lectures

- Midterm
- Heap Sort