CSE 332

JUNE 21 – WORKLISTS AND PRIORITY QUEUES

ASSORTED MINUTIAE

- **Midterm exam:**
  - July 14; 9:40-10:40
- **Canvas**
  - Site is up – EX01 is out
  - Project 1 out after class

ASSORTED MINUTIAE

- **Piazza**

ASSORTED MINUTIAE

- **Piazza**
  - Important course information and questions posted, link on the website

ASSORTED MINUTIAE

- **Piazza**
  - Important course information and questions posted, link on the website
  - All comments and questions will be put through there, so please register as soon as possible

TODAY'S SCHEDULE

- Testing
- Worklist ADT
- Priority Queues

TESTING

- **Implementation is great if it works on the first try**

TESTING

- **Implementation is great if it works on the first try**
- **In a large implementation, what is causing the problem?**
  - Data structure?
  - Client?
  - Wrapper?

TESTING

- Implementation is great if it works on the first try
- In a large implementation, what is causing the problem?
- Object oriented programming allows modularity – good testing can pinpoint bugs to particular modules

TESTING

- **Two primary types of testing**

TESTING

- **Two primary types of testing**
  - Black box
    - Behavior only, no peeking into the code

TESTING

- **Two primary types of testing**
  - Black box
    - Behavior only, no peeking into the code
  - White box (or clear box)
    - Where there is an understanding of the implementation that can be leveraged for testing

TESTING

- **Part 1 on the homework will involve writing tests for your own implementation. (White box)**
- **Part 2 will involve testing java .class files.**
  - Only the interface (TestQueue) and expected behavior are known

TESTING

- **Isolate the problem**

TESTING

- **Isolate the problem**
  - Write specific tests
  - Running the whole program doesn’t help narrow down problems

TESTING

- **Isolate the problem**
  - Write specific tests
  - Running the whole program doesn’t help narrow down problems
- **What are expected test cases?**

TESTING

- **Isolate the problem**
  - Write specific tests
  - Running the whole program doesn't help narrow down problems
- **What are expected test cases?**
  - In general: [0,1,n] are good starting points
  - White box testing can take advantage of boundary cases (e.g. the resize of an array)

TESTING

- **Many test cases (and large ones)**
  - You can prove that an algorithm is correct, but you cannot necessarily prove an arbitrary implementation is correct

TESTING

- **Many test cases (and large ones)**
  - You can prove that an algorithm is correct, but you cannot necessarily prove an arbitrary implementation is correct
- **More inputs can increase certainty**
  - Adversarial testing
  - The client is not your friend

TESTING

- This will come up a lot in the quarter
- HW1 Part 2 : .txt files

WORK LIST

- **Broad ADT**

WORK LIST

- **Broad ADT**
  - Comes up a lot in Project 1

WORK LIST

- **Broad ADT**
  - Comes up a lot in Project 1
- **Review – ADT is all about behavior**

WORK LIST

- **Broad ADT**
  - Comes up a lot in Project 1
- **Review – ADT is all about behavior**
  - Functionality:

WORK LIST

- **Broad ADT**
  - Comes up a lot in Project 1
- **Review – ADT is all about behavior**
  - Functionality:
    - Add(work)

WORK LIST

- **Broad ADT**
  - Comes up a lot in Project 1
- **Review – ADT is all about behavior**
  - Functionality:
    - Add(work)
    - peek()

WORK LIST

- **Broad ADT**
  - Comes up a lot in Project 1
- **Review – ADT is all about behavior**
  - Functionality:
    - Add(work)
    - peek()
    - next()

WORK LIST

- **Broad ADT**
  - Comes up a lot in Project 1
- **Review – ADT is all about behavior**
  - Functionality:
    - Add(work)
    - peek()
    - next()
    - hasWork()

WORK LIST

- Adds some information into a storage schema

WORK LIST

- **Adds some information into a storage schema**
- **Provides *some* ordering for that data to be processed**

WORK LIST

- How do we implement this ADT with what we currently know?

WORK LIST

- **How do we implement this ADT with what we currently know?**
  - Take a few minutes and think of some implementations that work off of things you've learned from 142/143

WORK LIST

- **How do we implement this ADT with what we currently know?**
  - Take a few minutes and think of some implementations that work off of things you've learned from 142/143
  - Consider design tradeoffs that we discussed a bit last week.

WORK LIST

- What is the big part of design here?

WORK LIST

- **What is the big part of design here?**
  - What do we consider to be next()?

WORK LIST

- **What is the big part of design here?**
  - What do we consider to be next()?
    - Stack: LIFO

WORK LIST

- **What is the big part of design here?**
  - What do we consider to be next()?
    - Stack: LIFO
    - Queue: FIFO

WORK LIST

- **What is the big part of design here?**
  - What do we consider to be next()?
    - Stack: LIFO
    - Queue: FIFO
    - Other?: Random possibly?

WORK LIST

- **What is the big part of design here?**
  - What do we consider to be next()?
    - Stack: LIFO
    - Queue: FIFO
    - Other?: Random possibly?
    - The ADT doesn’t specify

WORK LIST

- **What is the big part of design here?**
  - What do we consider to be next()?
    - Stack: LIFO
    - Queue: FIFO
    - Other?: Random possibly?
    - The ADT doesn't specify
    - What if we wanted the client to be able to specify the work order?

PRIORITY QUEUE

- New ADT

PRIORITY QUEUE

- **New ADT**
- **Objects in the priority queue have:**
  - Data
  - Priority

PRIORITY QUEUE

- **New ADT**
- **Objects in the priority queue have:**
  - Data
  - Priority
- **Conditions**
  - Lower priority items should dequeue first
  - Items with the same priority should be first-in first-out
  - Change priority?

PRIORITY QUEUE

- **Applications?**

PRIORITY QUEUE

- **Applications?**
  - Hospitals
  - CSE course overloads
  - Etc…

PRIORITY QUEUE

- How to implement?

PRIORITY QUEUE

- How to implement?
- Array?

PRIORITY QUEUE

- **How to implement?**
- **Array?**
  - Must keep sorted
  - Inserting into the middle is costly (must move other items)

PRIORITY QUEUE

- **How to implement?**
  - Keep data sorted (somehow)
- **Array?**
  - Inserting into the middle is costly (must move other items)
- **Linked list?**
  - Must iterate through entire list to find place
  - Cannot move backward if priority changes

PRIORITY QUEUE

- These implementations will all give us the behavior we want as far as the ADT, but they may be poor design decisions
- Any other data structures to try?

PRIORITY QUEUE

- **Priority queue implementations?**

PRIORITY QUEUE

- **Priority queue implementations?**
  - Binary search tree?

PRIORITY QUEUE

- **Priority queue implementations?**
  - Binary search tree?
    - Faster insert

PRIORITY QUEUE

- **Priority queue implementations?**
  - Binary search tree?
    - Faster insert
    - Find? Always deleting the smallest (left-most) element

PRIORITY QUEUE

- **Priority queue implementations?**
  - Binary search tree?
    - Faster insert
    - Find? Always deleting the smallest (left-most) element
    - Maintaining FIFO?

PRIORITY QUEUE

- **Priority queue implementations?**
  - Binary search tree?
    - Faster insert
    - Find? Always deleting the smallest (left-most) element
    - Maintaining FIFO?
    - Changing priority?

PRIORITY QUEUE

- Want the speed of trees (but not BST)
- Priority Queue has unique demands

PRIORITY QUEUE

- Want the speed of trees (but not BST)
- Priority Queue has unique demands
- Other types of trees?

PRIORITY QUEUE

- Want the speed of trees (but not BST)
- Priority Queue has unique demands
- Other types of trees?
- Review BST first

PROPERTIES (BST)

- **Tree**

PROPERTIES (BST)

- **Tree (Binary)**
  - Root
  - (Two) Children
  - No cycles

PROPERTIES (BST)

- **Tree (Binary)**
  - Root
  - (Two) Children
  - No cycles
- **Search**

PROPERTIES (BST)

- **Tree (Binary)**
  - Root
  - (Two) Children
  - No cycles
- **Search**
  - Comparable data
  - Left child data < parent data

PROPERTIES (BST)

- **Tree (Binary)**
  - Root
  - (Two) Children
  - No cycles
- **Search**
  - Comparable data
  - Left child data < parent data
  - Smallest child is at the left most node

PROPERTIES (BST)

- Binary tree may be useful
- Search property doesn’t help

PROPERTIES (BST)

- **Binary tree may be useful**
- **Search property doesn't help**
  - Always deleting min

PROPERTIES (BST)

- **Binary tree may be useful**
- **Search property doesn't help**
  - Always deleting min
  - Put min on top!

HEAP-ORDER PROPERTY

- **Still a binary tree**

HEAP-ORDER PROPERTY

- **Still a binary tree**
- **Instead of search (left < parent),**

HEAP-ORDER PROPERTY

- **Still a binary tree**
- **Instead of search (left < parent),**
  parent should be less than children

HEAP-ORDER PROPERTY

- Still a binary tree
- Instead of search (left < parent),
  parent should be less than children
- How to implement?
- Insert and delete are different than BST

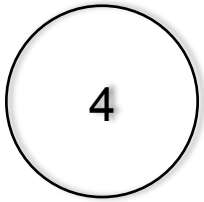
HEAP-ORDER PROPERTY

- Still a binary tree
- Instead of search (left < parent),
  parent should be less than children
- How to implement?
- Insert and delete are different than BST

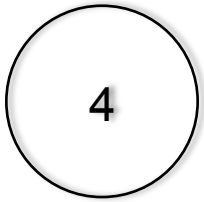
HEAP EXAMPLE

- Only looking at priorities
- Insert something priority 4

HEAP EXAMPLE

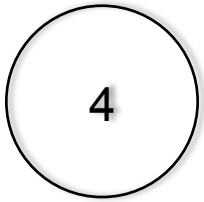


HEAP EXAMPLE



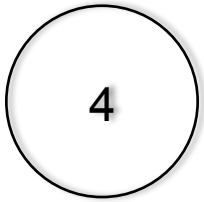
- **Now insert priority 6?**

HEAP EXAMPLE



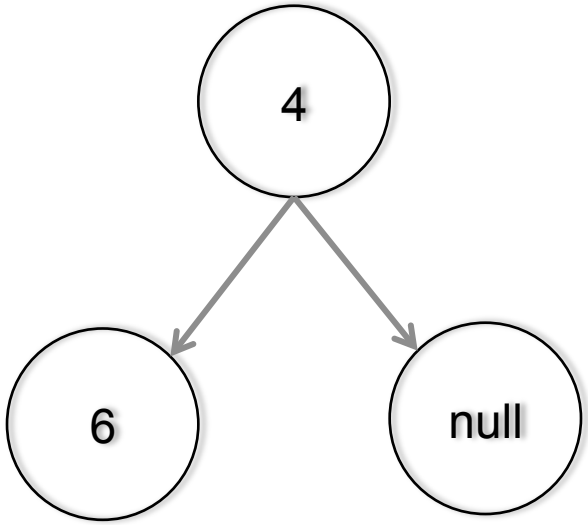
- **Now insert priority 6?**
- **Should come after 4, but no preference right over left?**

HEAP EXAMPLE



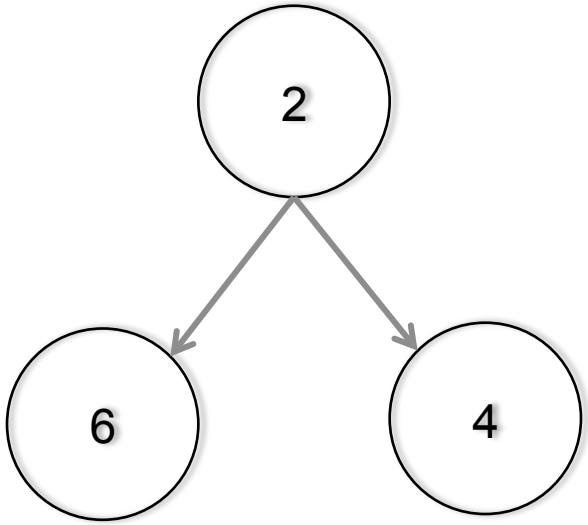
- **Now insert priority 6?**
- **Should come after 4, but no preference right over left?**
- **Solution: fill the tree from top to bottom left to right.**

HEAP EXAMPLE



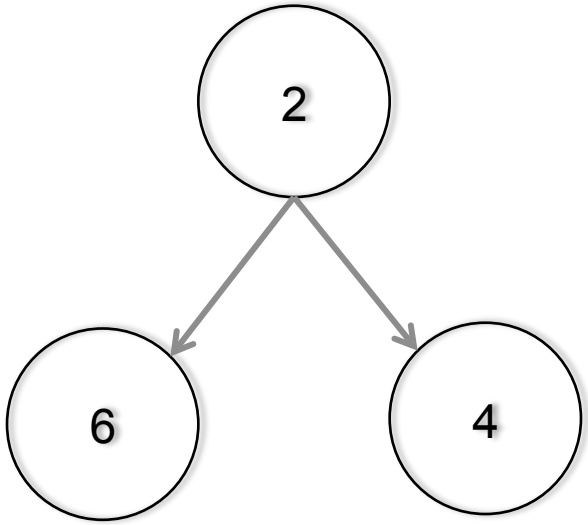
Now insert 2.

HEAP EXAMPLE



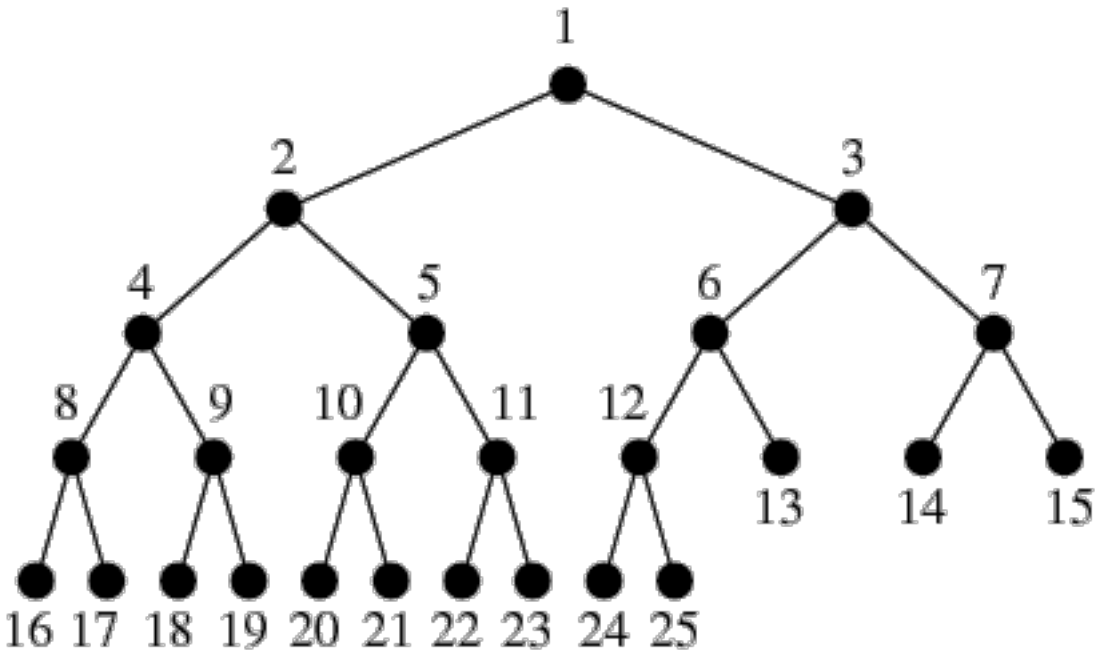
Now insert 2.

HEAP EXAMPLE

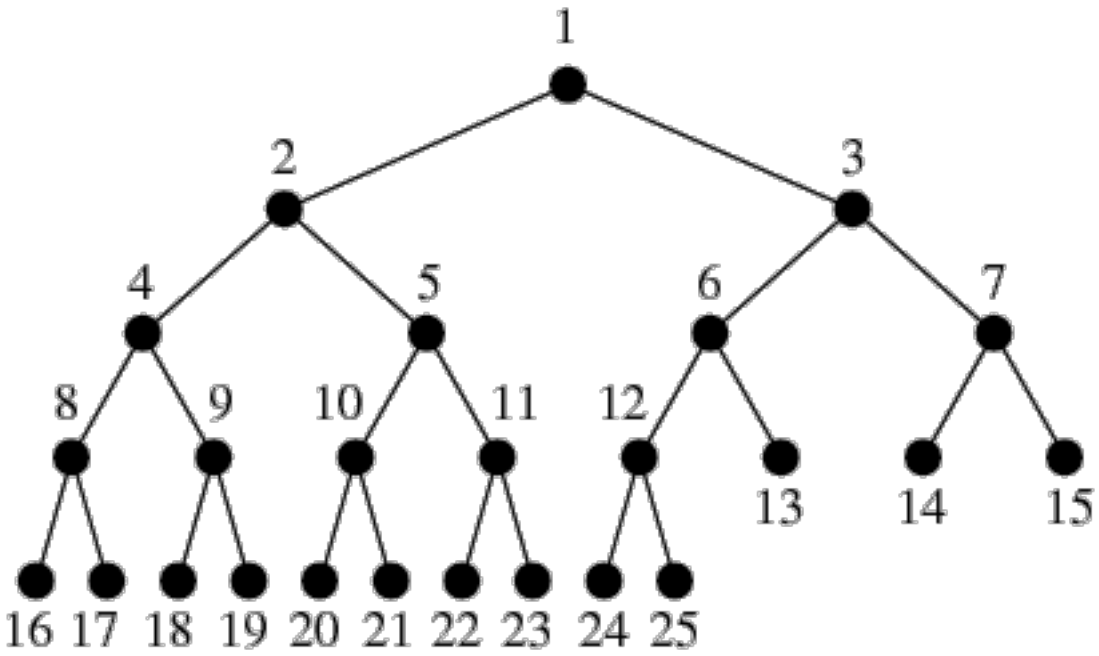


Could easily have been 4 on the left, but our left to right top to bottom strategy determines this solution

COMPLETENESS



COMPLETENESS



Filling left to right and top to bottom is another property - completeness

HEAPS

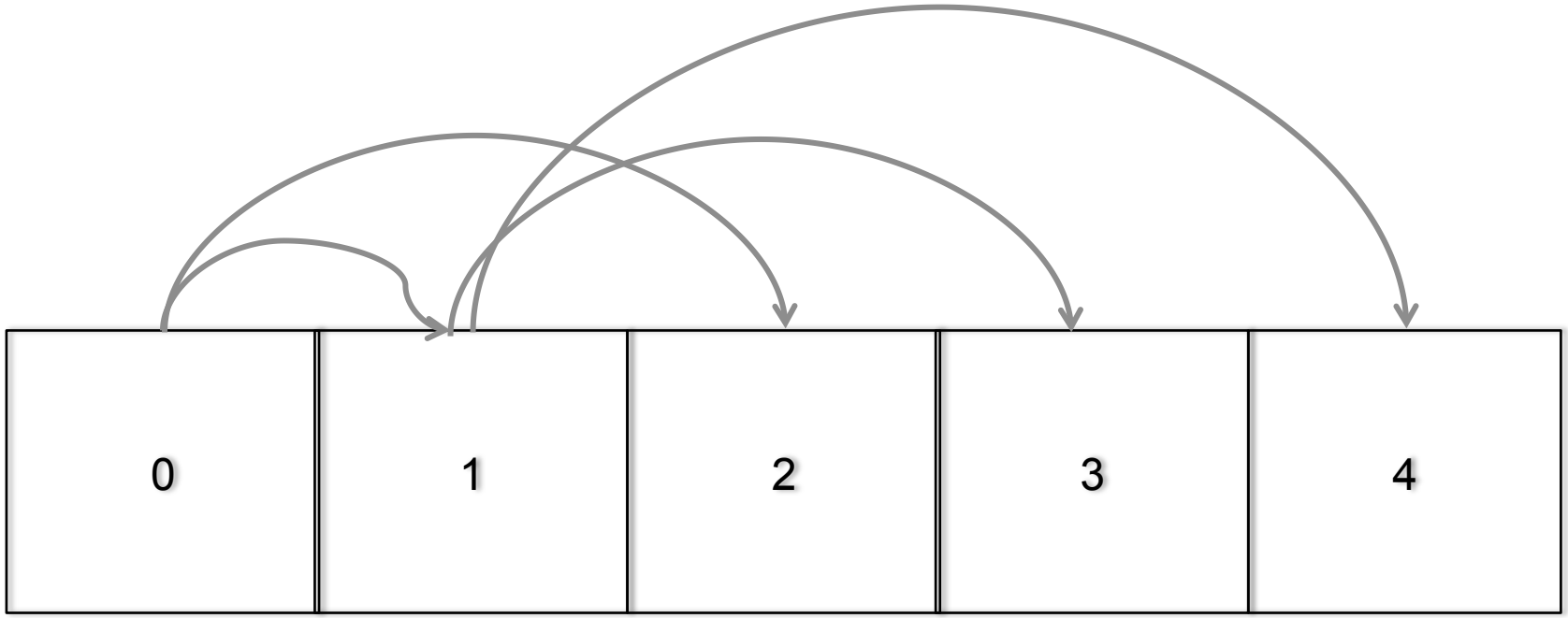
- Heap property (parents < children)
- Complete tree property (left to right, bottom to top)
- How does this help?

HEAPS

- **Heap property (parents < children)**
- **Complete tree property (left to right, bottom to top)**
- **How does this help?**
  - Array implementation

HEAPS

- Insert into array from left to right
- For any parent at index i,
  children at 2*i+1 and 2*i+2



HEAPS

- How to maintain heap property then?

HEAPS

- **How to maintain heap property then?**
  - Parent must be higher priority than children

HEAPS

- **How to maintain heap property then?**
  - Parent must be higher priority than children
- **Two functions – percolate up and percolate down**

HEAPS

- **Does the heap work for the Priority Queue problem?**

HEAPS

- **Does the heap work for the Priority Queue problem?**
  - FIFO preservation?

HEAPS

- **Does the heap work for the Priority Queue problem?**
  - FIFO preservation?

    No. Only comparisons are priority.