Implementing priority queues

5/15/26

Recall: Priority Queue ADT

- Store collection of values and read them out in priority order (as given by a Comparator)
- Operations:
  - void enqueue(value) //add value to collection
  - value dequeue() //remove & return next val
  - value peek() //return next val w/o removing
  - int size() //get collection size

Application: Sorting

- You have an array of values and want to put them in order

Application: HPC system simulation

- High-performance computing (HPC) system
  - 10,000+ processors in a room w/ fast network
  - Programs (“jobs”) run on many processors at once
  - Jobs submitted to a “queue” w/ a number of processors and an estimated running time

- Simulating it:
  - Read list of jobs from a file (size, duration, estimated duration, submission time)
  - Keep track of state of system over time

Mechanics of simulation

- Have a current time t
- Have some jobs with arrival time > t
- Have some jobs running, each with a completion time
- Have a “queue” of waiting jobs to run in order

Suppose a Priority Queue is implemented using a **unsorted** linked list. Which of the following are the running times of enqueue and dequeue respectively?

A. O(1) and O(log n)
B. O(1) and O(n)
C. O(n) and O(1)
D. O(n) and O(n)
E. None of the above

Suppose a Priority Queue is implemented using a **unsorted** linked list. Which of the following are the running times of enqueue and dequeue respectively?

A. O(1) and O(log n)
B. O(1) and O(n)
C. O(n) and O(1)
D. O(n) and O(n)
E. None of the above

Suppose a Priority Queue is implemented using a **sorted** linked list (highest priority first). Which of the following are the running times of enqueue and dequeue respectively?

A. O(1) and O(log n)
B. O(1) and O(n)
C. O(n) and O(1)
D. O(n) and O(n)
E. None of the above

Suppose a Priority Queue is implemented using a **sorted** linked list (highest priority first). Which of the following are the running times of enqueue and dequeue respectively?

A. O(1) and O(log n)
B. O(1) and O(n)
C. O(n) and O(1)
D. O(n) and O(n)
E. None of the above

First implementation ideas

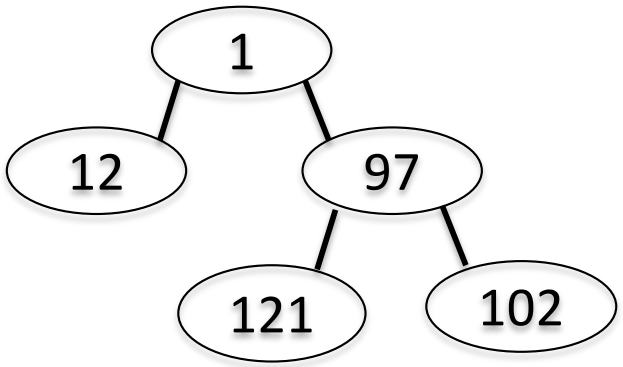
- Linked list sorted by priority:
  - O(n) enqueue, O(1) dequeue
- Unsorted linked list:
  - O(1) enqueue, O(n) dequeue

Heaps

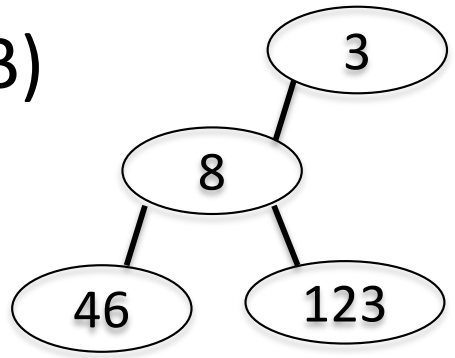
- A heap is a data structure specifically for priority queues
- Displayed as a complete binary tree (two children for every node except possibly nodes on the right of the bottom level)
- Each parent is <= either child (by Comparator)

Which of the following is a valid heap?

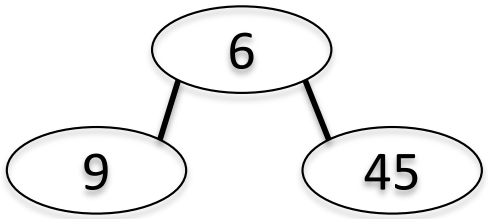
A)



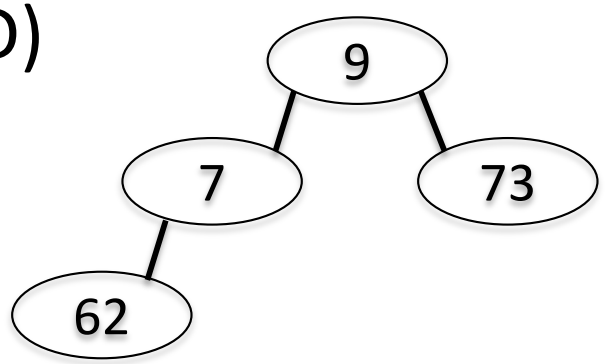
B)



C)



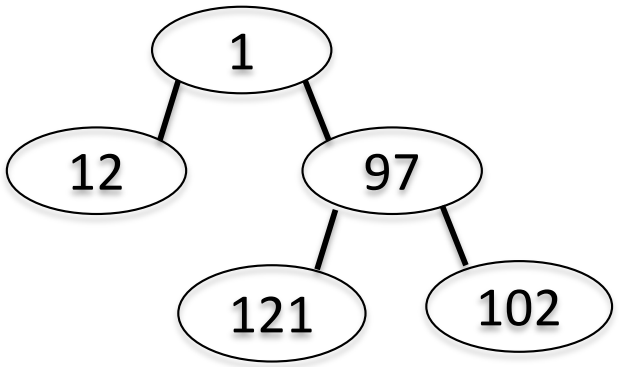
D)



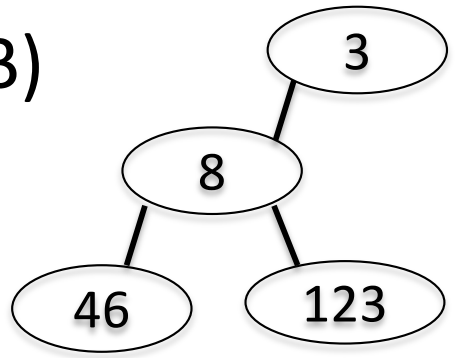
E) Not exactly one of the above

Which of the following is a valid heap?

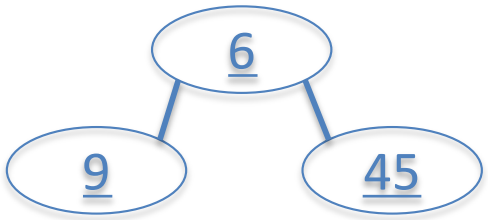
A)



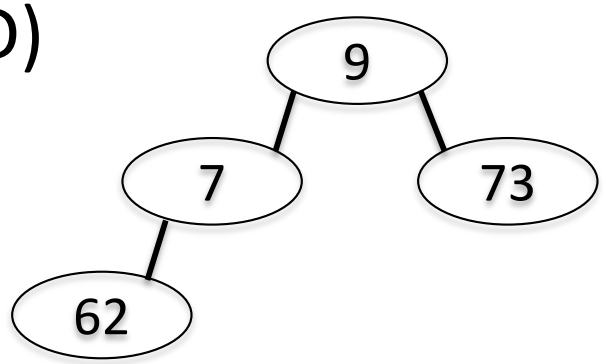
B)



C)



D)



E) Not exactly one of the above

Efficient representation of a heap

- Values stored in an array indexed from 1
- Root stored in index 1
- Can move thru the tree via index arithmetic:
  - left child of value in index i: index 2i
  - right child of value in index i: index 2i+1
  - parent of value in index i: index i/2 (round down)

Which of the following is a valid heap?

A.

1	2	3	4	5	6	7	8	9
4	5	6	7	8	12	18	142	215

B.

1	2	3	4	5	6	7	8	9
4	6	5	12	8	142	7	18	215

C.

1	2	3	4	5	6	7	8	9
4	7	5	142	12	8	6	18	215

D.

1	2	3	4	5	6	7	8	9
4	8	5	12	142	6	215	7	18

E. Not exactly one of the above

Which of the following is a valid heap?

A.

1	2	3	4	5	6	7	8	9
4	5	6	7	8	12	18	142	215

B.

1	2	3	4	5	6	7	8	9
4	6	5	12	8	142	7	18	215

C.

1	2	3	4	5	6	7	8	9
4	7	5	142	12	8	6	18	215

D.

1	2	3	4	5	6	7	8	9
4	8	5	12	142	6	215	7	18

E. Not exactly one of the above (A and B)

How do we add a value to a heap?

Upheap: letting values float up

```
void upheap(int i) {
    //i is index that may need to move up
    while((i > 1) && (A[i/2] > A[i])) {
        swap A[i] and A[i/2]
        i = i / 2;
    }
}
```

What heap results when 64 is added to the heap below?

1	2	3	4	5	6	7	8	9	10
4	6	5	12	8	142	7	18	215	

What heap results when 2 is added to the heap below?

1	2	3	4	5	6	7	8	9	10
4	6	5	12	8	142	7	18	215	

How do we dequeue the minimum value?

Downheap: letting values sink down

```
void downheap(int i) {
    while(2*i <= size) {
        int j = 2*i;          //smaller child's index (val to replace A[i])
        if((j < size) && (A[j] > A[j+1]))
            j++;
        if(A[i] <= A[j])
            break;
        swap A[i] and A[j]
        i = j;
    }
}
```

What heap results when the minimum is removed from the heap below?

1	2	3	4	5	6	7	8	9
4	6	5	12	8	142	7	18	215

Implementing heap sort

- Can turn array into heap by running downheap on each element, starting at end ("heapifying" an array)

- As values are removed from the heap, it shrinks and frees up space in the array