Precept 2:
Data structures, Searching, and Sorting

Qian Zhu

Feb 8, 2011

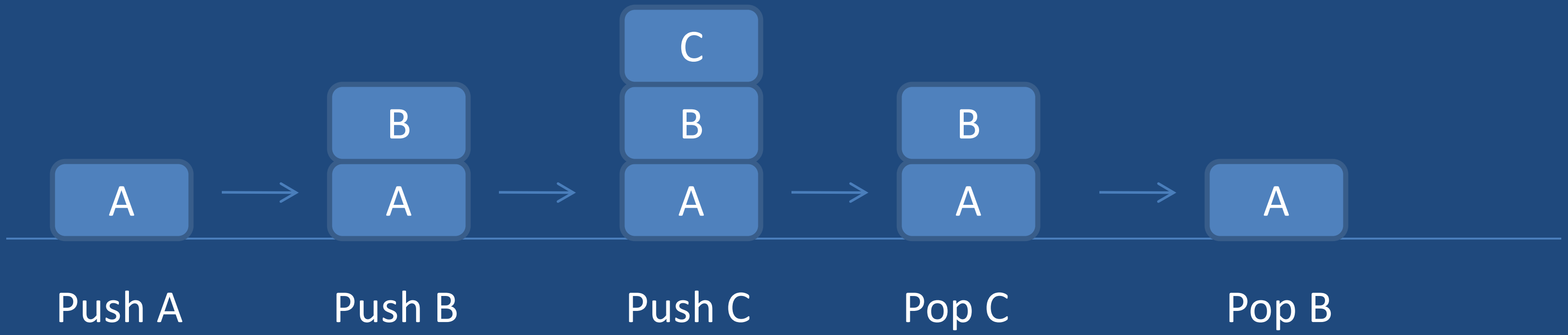
Agenda

- Linear data structures (queues and stacks)
- Tree structure (binary trees for searching)
- Sorting algorithms (merge sort)

- Assignment 2

Quick review: Stacks

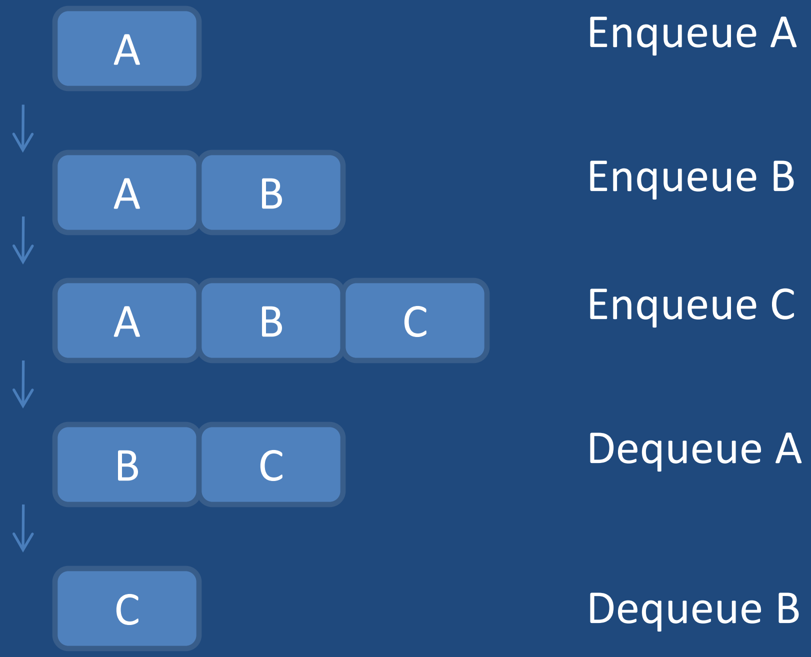
- Last in first out (LIFO)
- Imagine a stack of books on table



Insert: always on top of stack
Remove: always from top of stack

Quick review: Queues

- First in first out (FIFO)
- Imagine a queue of people, first come first served



Enqueue A

Enqueue B

Enqueue C

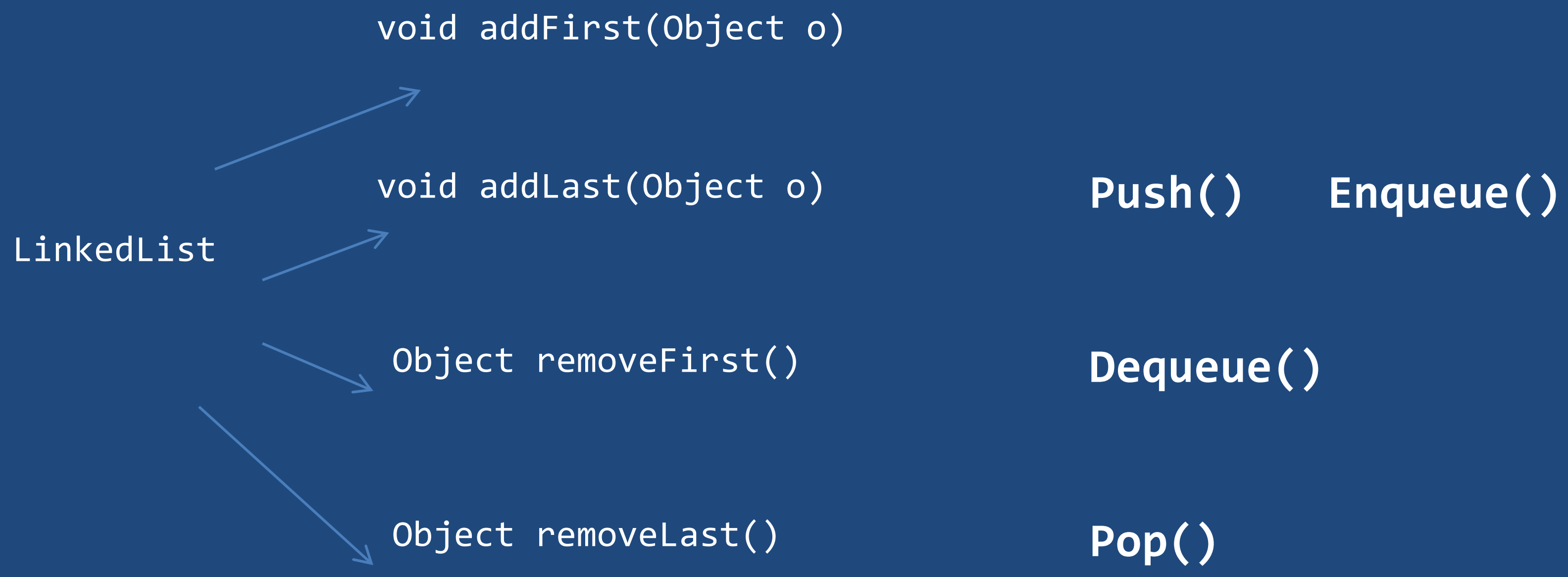
Dequeue A

Dequeue B

Insert: always at the tail
Remove: always at the head

Creating Stacks and Queues in Java

- Java provides the Stack and Queue implementations through a general data type, `LinkedList`.



`LinkedList`

`void addFirst(Object o)`

`void addLast(Object o)` **Push() Enqueue()**

`Object removeFirst()` **Dequeue()**

`Object removeLast()` **Pop()**

Declaring Queue and Stack

COS 126 Queue.java

```
Queue que = new Queue();
      que.enqueue(), que.dequeue()...
```

Standard Queue API from java.util

```
Queue que = new LinkedList();
       que.offer(), que.remove()...
```

Standard Stack API from java.util

```
 Stack st = new LinkedList();
        st.push(), st.pop()...
```

Want to enforce the same data type for all elements in a queue, or stack?

Answer: Generics

```
Stack<Integer> st = new LinkedList<Integer>();
```

```
Integer a = new Integer(3);
Integer b = new Integer(4);
Integer c = new Integer(5);

st.push(a); st.push(b); st.push(c);



Double d = new Double(3.0);
st.push(d); ?
```

Warning: only wrapper data type can go in <>.
i.e., `Stack<int>` is not allowed.

Searching in a queue or stack

- Which elements can you see in a queue and in a stack?

- Can you see elements in the middle of a queue? of a stack?

Comparisons with arrays

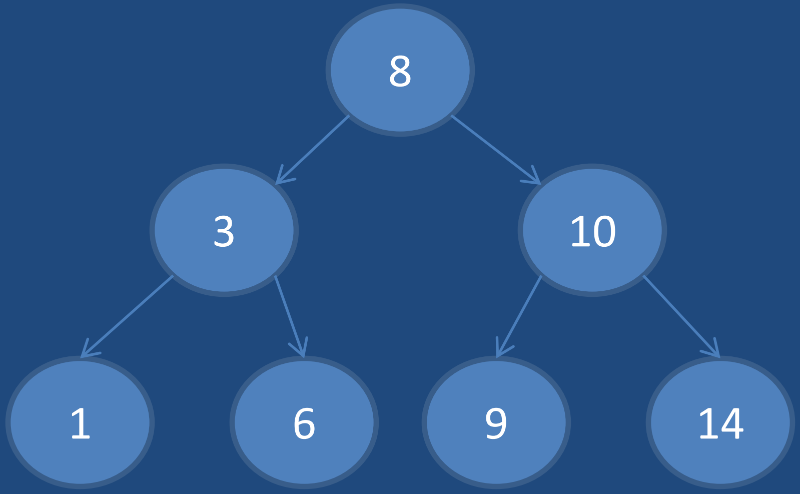
- Efficiency comparisons:

Input size: n

	LinkedList (with first, and last pointers)	**Array**
Indexed access	First: Last: Middle:	
Insert	First: Last: Middle:	
Delete	First: Last: Middle:	

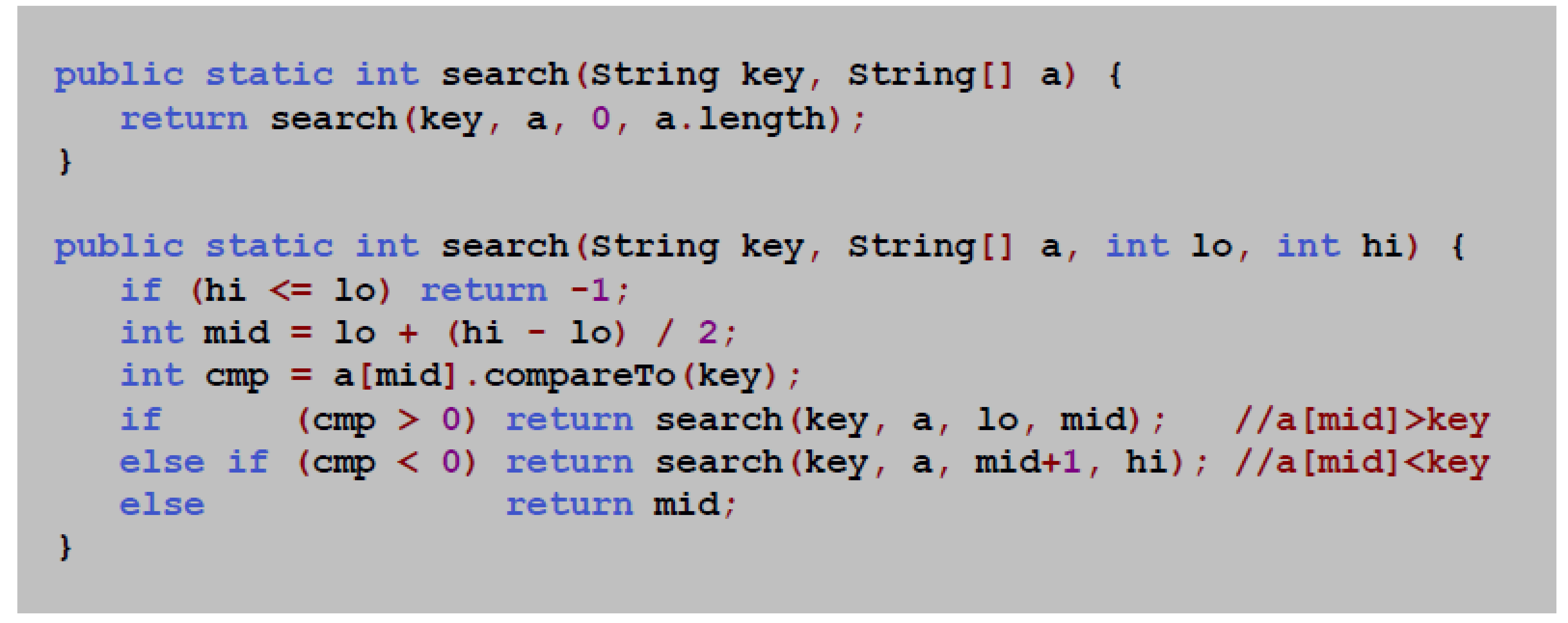
Tree structure

- Binary tree (parent, two children)
- Traversal



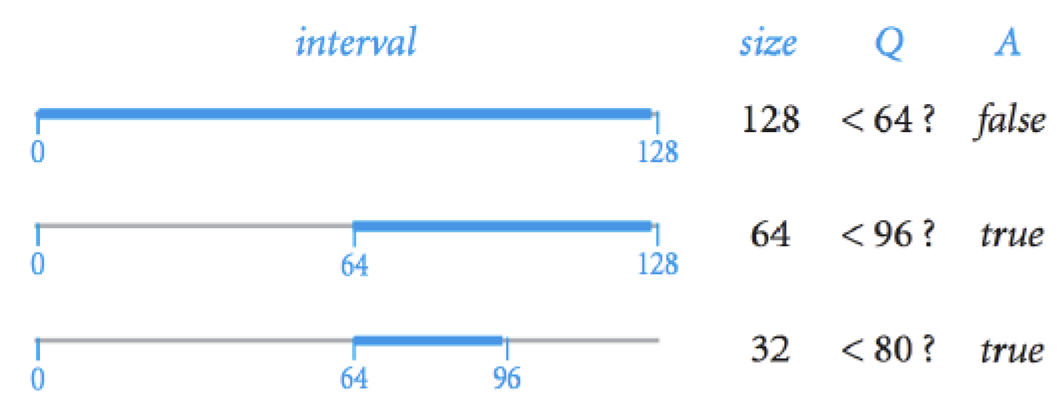
In – order: left, root, right
Post – order: left, right, root
Pre – order: root, left, right

Binary search using array



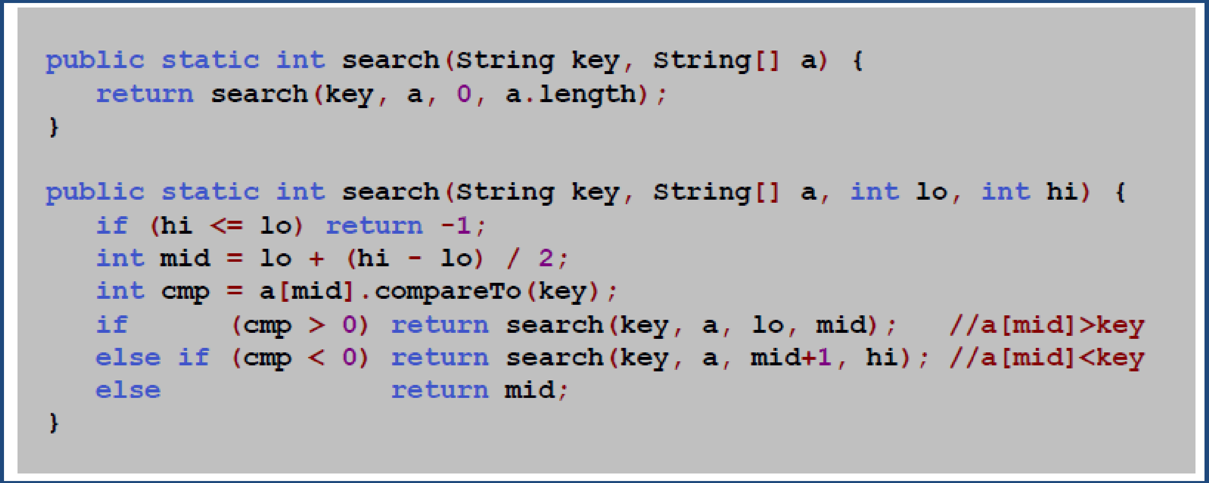
```java
public static int search(String key, String[] a) {
   return search(key, a, 0, a.length);
}

public static int search(String key, String[] a, int lo, int hi) {
   if (hi <= lo) return -1;
   int mid = lo + (hi - lo) / 2;
   int cmp = a[mid].compareTo(key);
   if      (cmp > 0) return search(key, a, lo, mid);   //a[mid]>key
   else if (cmp < 0) return search(key, a, mid+1, hi); //a[mid]<key
   else              return mid;
}
```



size	Q	A
128	< 64 ?	*false*
64	< 96 ?	*true*
32	< 80 ?	*true*

Binary search using array, tracing



```
public static int search(String key, String[] a) {
   return search(key, a, 0, a.length);
}

public static int search(String key, String[] a, int lo, int hi) {
   if (hi <= lo) return -1;
   int mid = lo + (hi - lo) / 2;
   int cmp = a[mid].compareTo(key);
   if      (cmp > 0) return search(key, a, lo, mid);   //a[mid]>key
   else if (cmp < 0) return search(key, a, mid+1, hi); //a[mid]<key
   else              return mid;
}
```

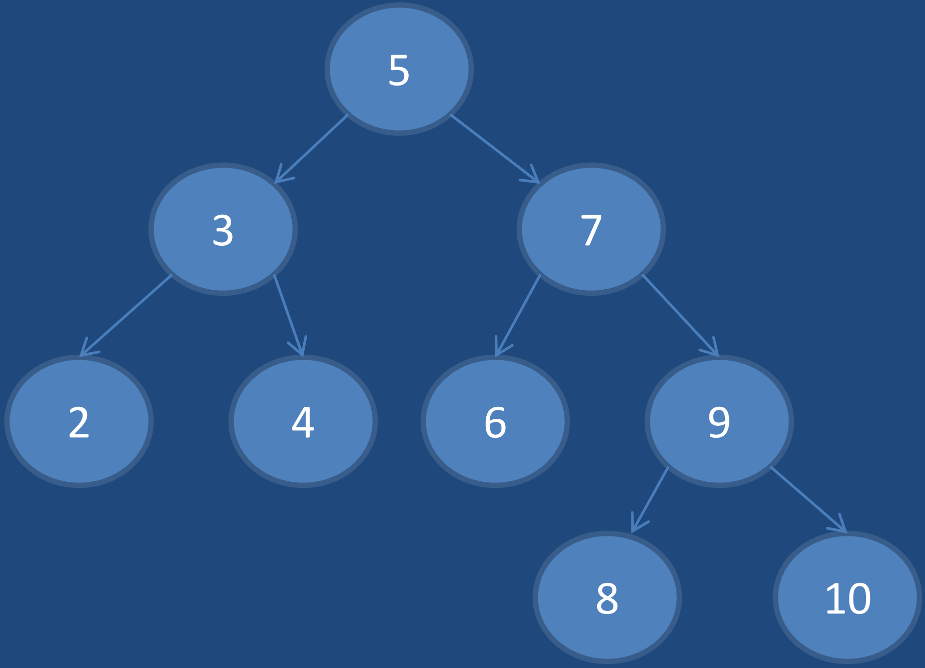
- Input: [2, 3, 4, 5, 6, 7, 8, 9, 10]

Search 2:
Lo: 0, 0, 0, 0
Hi: 9, 4, 2, 1
Mid: 4, 2, 1, 0

Search 10:
Lo: 0, 5, 8
Hi: 9, 9, 9
Mid: 4, 7, 8

Binary search using tree

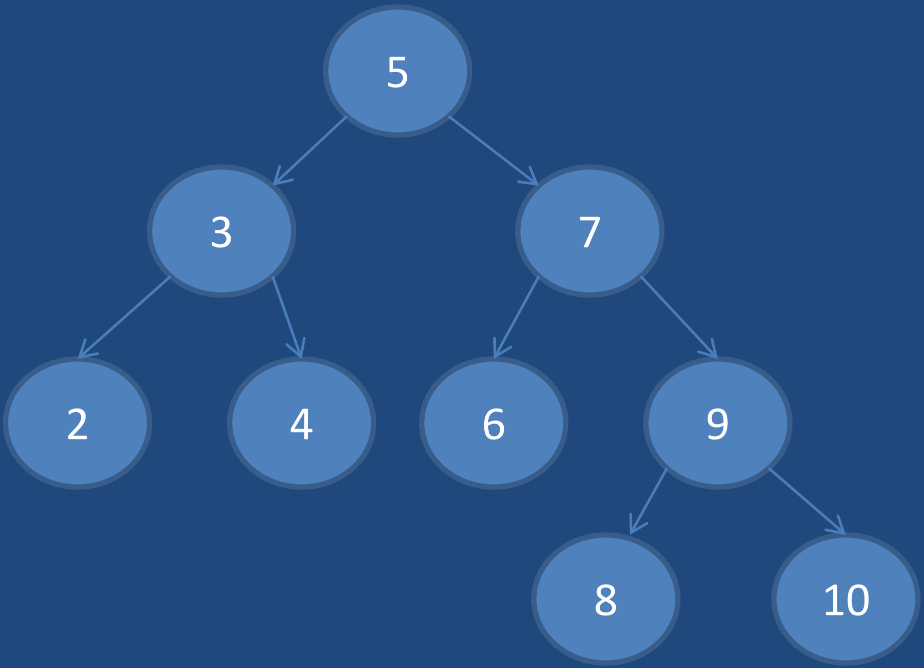
- [2, 3, 4, 5, 6, 7, 8, 9, 10]
- Represent using binary tree, in in-order order



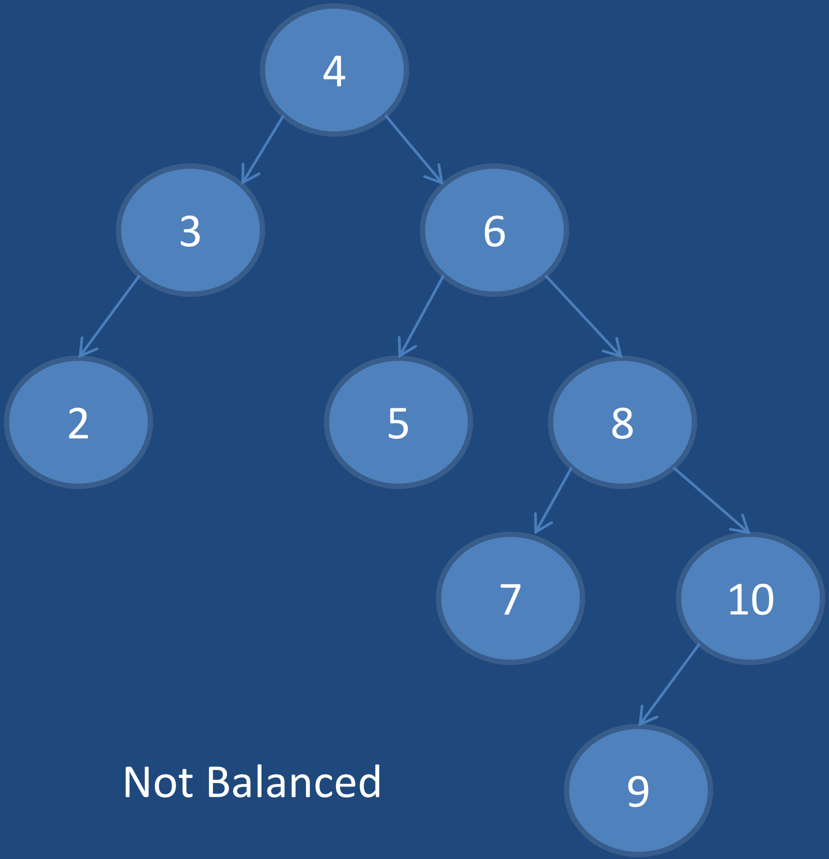
Start at root.
1) Compare value of current node to search element.
If =: return
If <: search T rooted at left child. Go to 1)
If >: search T rooted at right child. Go to 1)

Binary search using tree

- Searching is most efficient if the binary tree is *balanced*



Balanced



Not Balanced

Binary search using tree

- Creating a **balanced** binary search tree is easy if the array is **static.**
- If the array is **dynamic** (i.e., support update operations), maintaining a balanced tree on the fly is a hard problem.
  - *Self-balancing binary search tree*: efficiently balances the tree after each update without recreating tree (e.g., AVL, red-black trees)

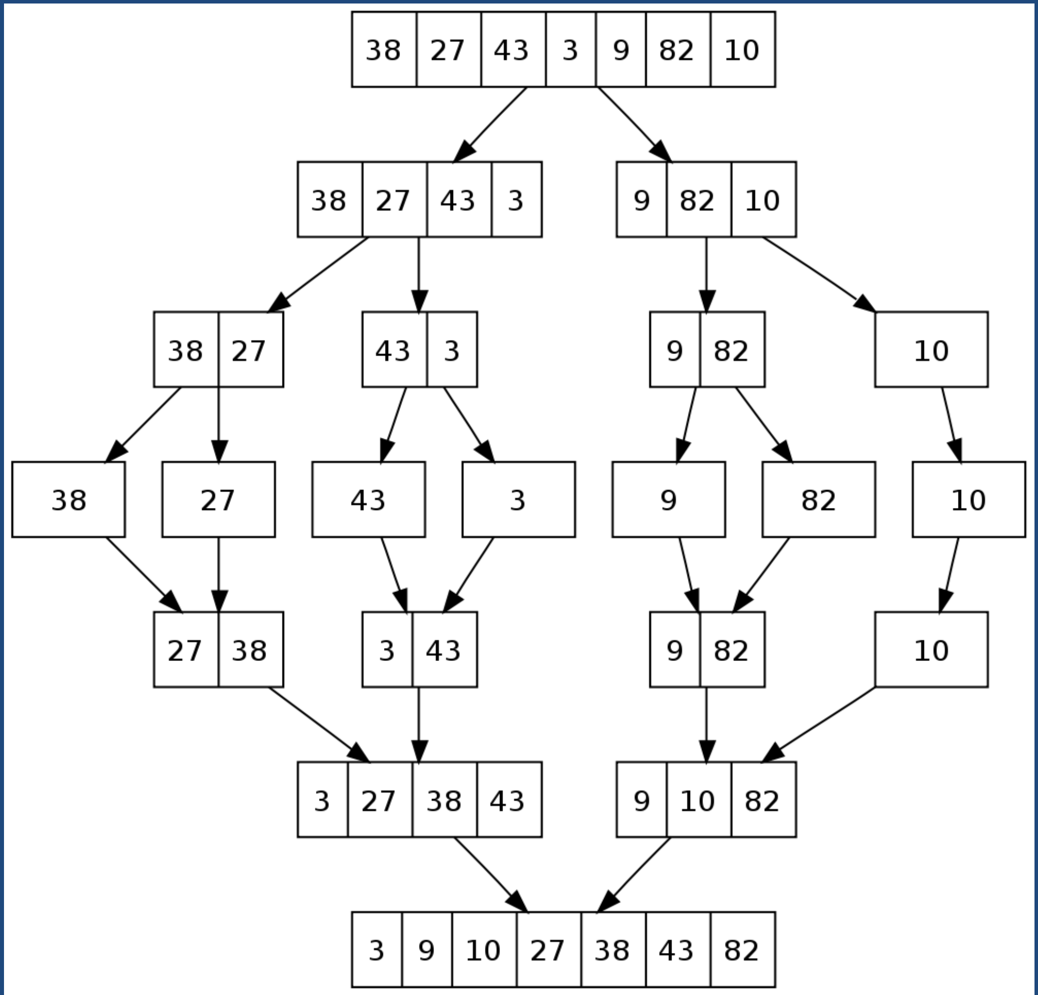
Sorting

- Naïve sorting algorithm:
  - Quadratic time
  - Simplest: bubble sort, selection sort
- **Smarter algorithm:**
  - Merge sort (tree-based)

Merge Sort

- Simple Algorithm:
  1. Divide array into two halves
  2. Recursively sort each half
  3. Merge two halves to make sorted whole

Merge Sort (example)



Analysis:

1) Splitting:

+

2) Merging per level:

x Number of levels:

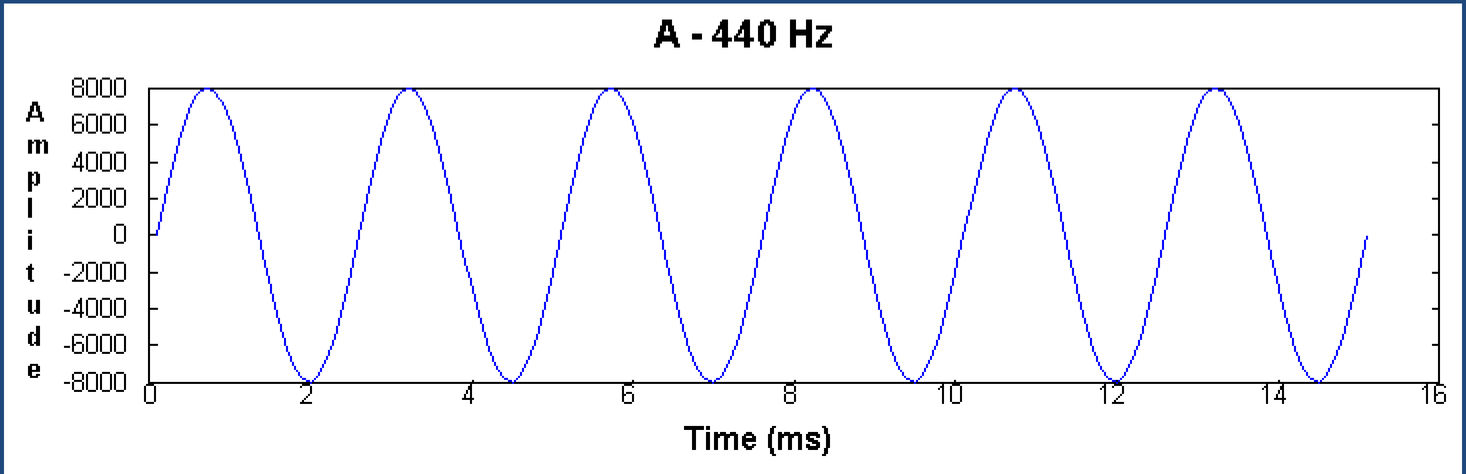
Total:

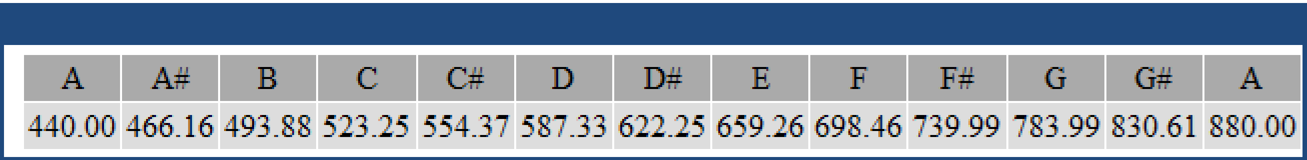
Assignment 2

- You can work in pairs.

- Be able to understand how sound wave is stored and manipulated in Java. (`Wave.java`)
- Be able to add an echo effect to the sound. (`EchoFilter.java`)

What are sound waves?

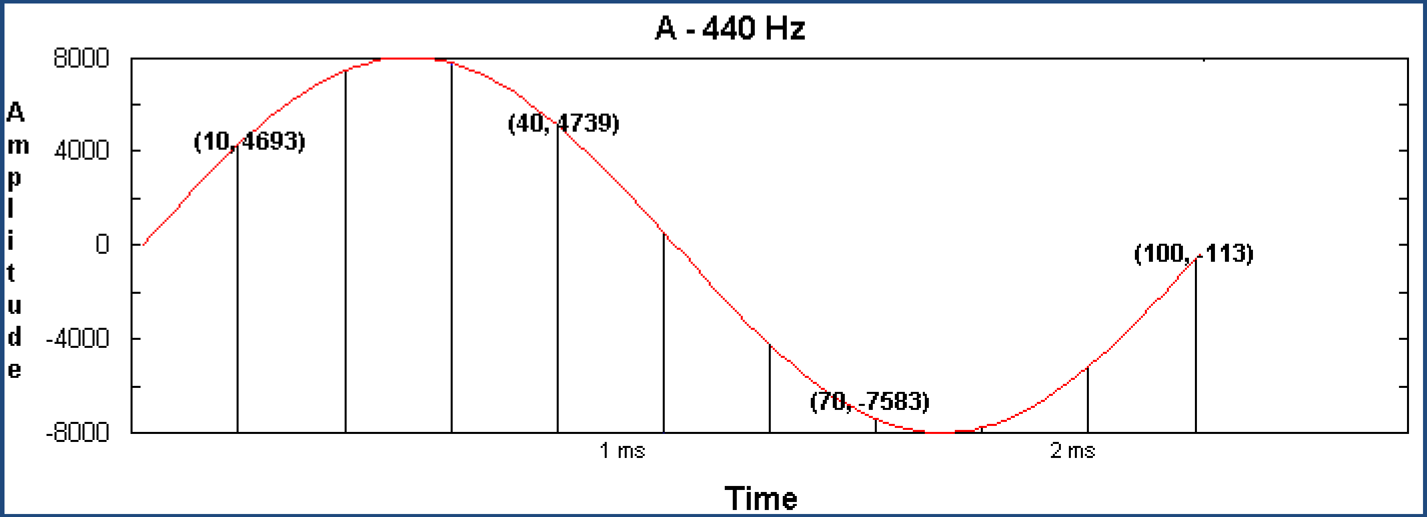




A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A
440.00	466.16	493.88	523.25	554.37	587.33	622.25	659.26	698.46	739.99	783.99	830.61	880.00

- A music note can be characterized by its frequency
  – E.g., A = 440 Hz, C = 523.35 Hz
- Two components: frequency and maximum amplitude

- How to store sound waves?



- A sound wave is continuous. Can’t store this. Must sample it at some regular time intervals.
- We sample the instantaneous amplitude of the continuous wave at a certain frequency
  - CDs uses 44 100 Hz, two channels (take 44.1 thousand samples per second)
  - Each sample is a 16-bit integer (Java `short`)

- Amplitude$_i$ = max_amp * sin (2 * pi * freq * I * sampling_rate)

- **Writing Wave.java**
    - Store left and right channels using two `short` arrays
    - Declare your constants (SAMPLING_RATE)

- **`public Wave plus(Wave a)`**
    - Add samples from left channels together, repeat for right channel.
    - Returns a new Wave
    - Must cast to short

- **Writing EchoFilter.java**
    - Load the sound from an MP3 file, and add echo effect to it.
    - Maintain a queue of last 10 waves (use the provided `Queue.java` library)
    - Add wave at time t-10 to wave at time t.
    - Use enqueue and dequeue mechanisms to fix queue size at 10

Tips

- Follow directions on using JAR file
  - `javac -classpath .:player.jar A.java Wave.java`
  - `java -classpath .:player.jar A`

- `import javazoom.jl.player.Player;`

- Remember to declare constants.