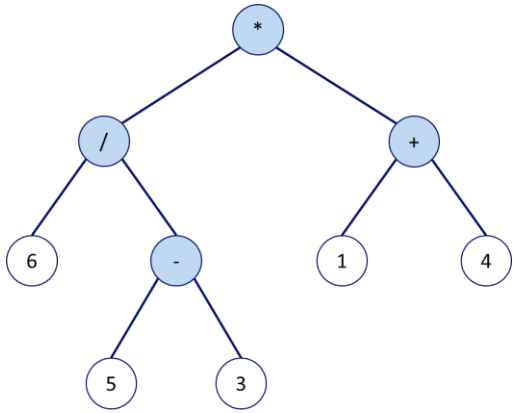
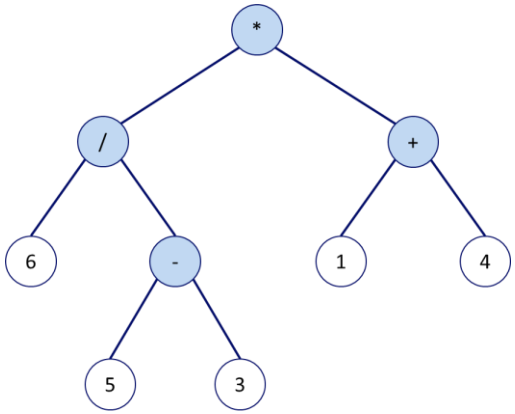
Trees

Traversals:

- In-Order:
- Pre-Order:
- Post-Order:
- Level-Order:





Implementation:

In-Order:	Pre-Order	Post-Order

A different type of traversal: Level-Order

Strategy:

Evaluating the tree expression:

1. Traverse through each item in the post-order expression

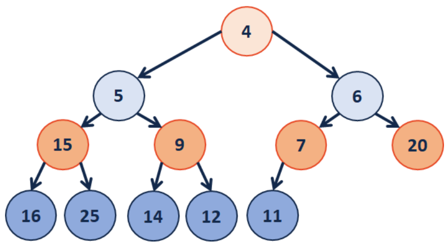
2. If item is an operand, push to stack. Otherwise, pop two elements.
   i. Let A be first popped element
   ii. Let B be second popped element
   iii. Evaluate B <operator> A
   iv. Push result to stack

3. Pop final item and return it.

Formally, a binary tree T is a minHeap if:

-
-

Heap Operation: insert / bubbleUp:



–	4	5	6	15	9	7	20	16	25	14	12	11			

–	4	5	6	15	9	7	20	16	25	14	12	11			

–	4	5	6	15	9	7	20	16	25	14	12	11			

Heap.java (partial)

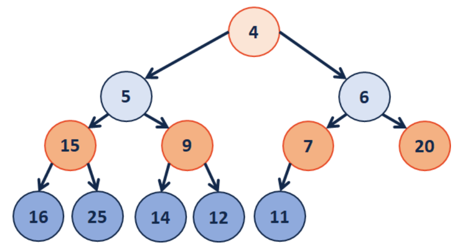
```
void insert(int key) {
  // Check to ensure there's space to insert an element
  if (size == CAPACITY) { throw new FullHeapException… }

  // Insert the new element at the end of the array
  data[++size] = key;

  bubbleUp(size); // Restore the heap property
}
```

```
void bubbleUp( ___________ ) {
  if ( ________________ ) {
    if ( data[index] < data [ ________ ] ) {
       swap(data, index, index/2);
       bubbleUp( ____________ );
    }
  }
}
```

Heap Operation: removeMin / bubbleDown



–	4	5	6	15	9	7	20	16	25	14	12	11			

–	4	5	6	15	9	7	20	16	25	14	12	11			

–	4	5	6	15	9	7	20	16	25	14	12	11			

–	4	5	6	15	9	7	20	16	25	14	12	11			

Heap.java (partial)

```
int removeMin() {
  // Swap with the last value
  int minValue = data[1];
  items[1] = data[size];
  data[size] = minValue;
  size--;

  bubbleDown(1); // Restore the heap property
  return minValue; // Return the minimum value
}
```

```
void bubbleDown(int index) {
  if ( !isLeaf(index) ) {
    int minChildIndex = minChild(index);
    if ( items[index] _____ items[minChildIndex] ) {
       swap(data, index, minChildIndex);
       bubbleDown( ___________________ );
    }
  }
}
```