UIL Computer Science for Dummies™

by
Jake Warren
and works from Mr. Fleming

Foreword

First of all, this book isn't really for dummies. I wrote it for myself and other kids who are on the team. Everything here is stuff that either nabbed me on tests or is very obscure information. Hopefully this book will help you in your studies.

Jake Warren

P.S.

Don't forget to send you feedback to either jakewarren74@msn.com or jnw0050@unt.edu !!!

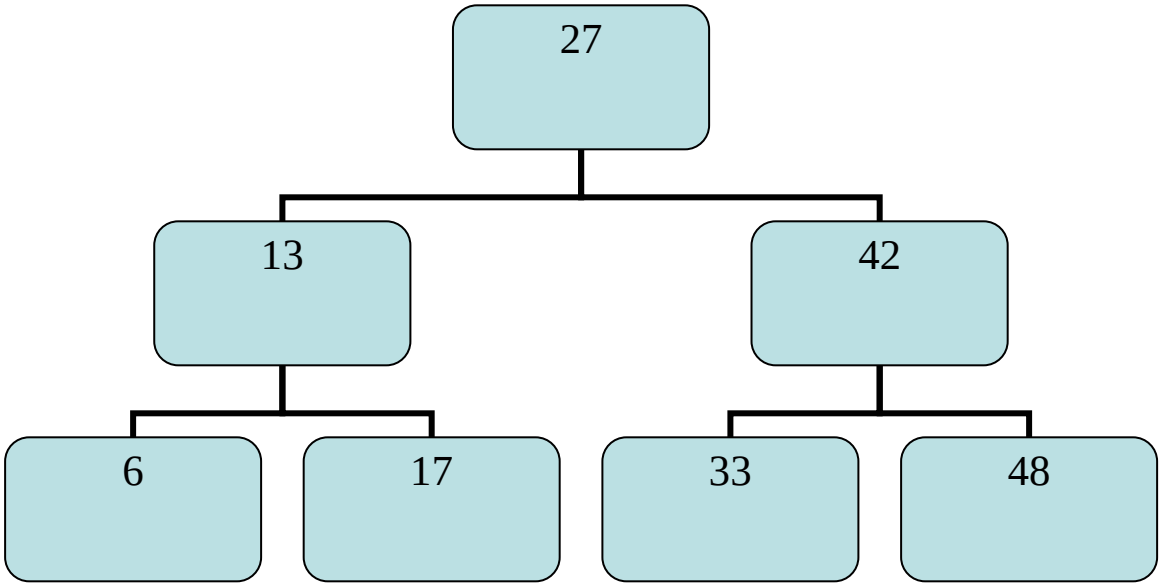
Table of Contents

Binary Search Trees

- A value that is less than the value in the parent node goes to the left.

The following values are inserted into the tree:

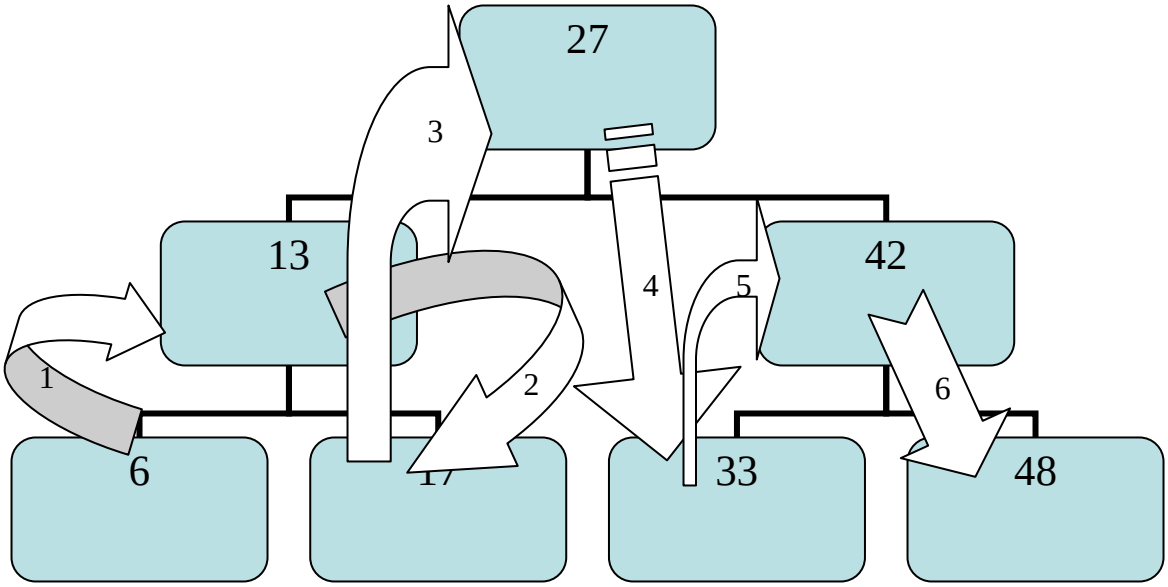
27 6 13 17 33 42 48



Inorder Traversal

- Returns values in ascending order.

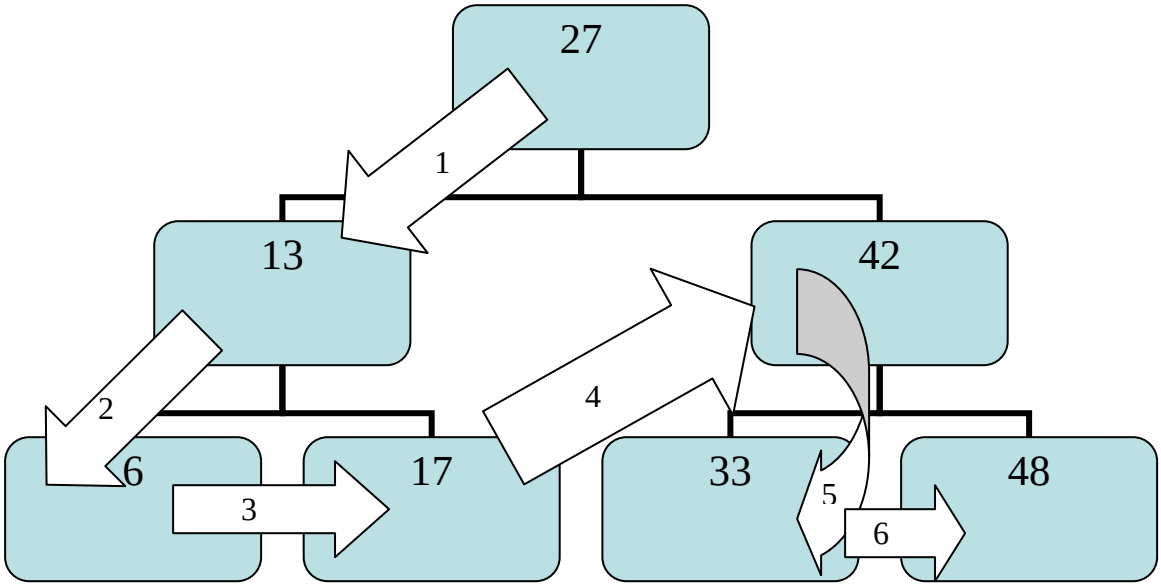
6 13 17 27 33 42 48



Pre-Order Traversal

- Process the left sub-tree first
- Process the right sub-tree next
- Start at the top
- Work your way down

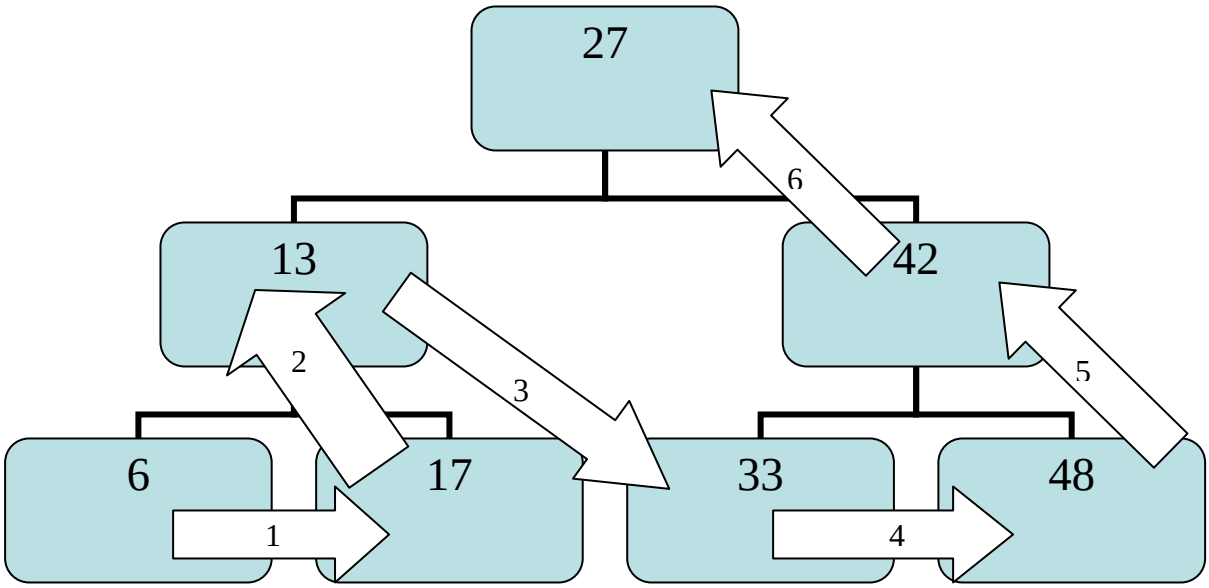
27 13 6 17 42 33 48



Post-Order Traversal

- Start with the children from the bottom up.
- Process the left nodes first
- Parent node last!!!

6 17 13 33 48 42 27



Binary Search Tree Notes

- If a duplicate value is inserted, it will be discarded.

Big-O Notation

- Extremely Fast

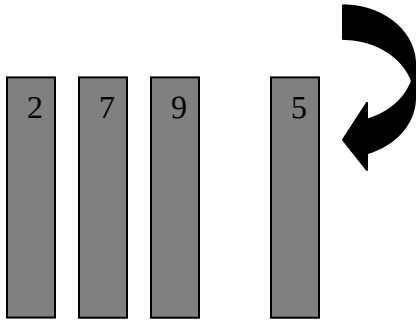
  Common: $\log_2 n$
  Worst-case: $\log_2 n$

Insertion Sort

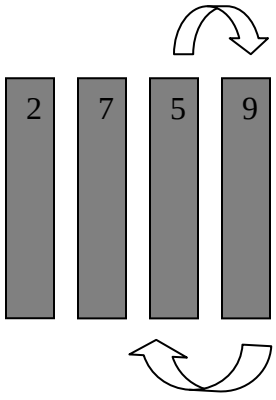
Here is how the insertion sort works graphically:

You have a collection of CDs; each CD has an ID#, and you want to sort them in ascending order.

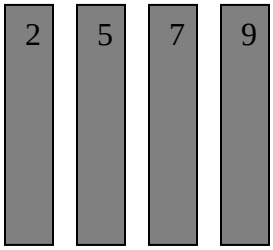
1. Add the new CD.



2. Compare the new CD(CD5) with the rightmost CD(CD9). Since 5 is less than 10, swap them.



3. Repeat Step 2 until the new CD is greater than the CD to the left.



Here is the recursive function that does the sort:

```
int[5] mList = new int [] {1,2,4,5,3};
public void insertItem(int ItemToInsert)
{
    *swap( itemToInsert,itemToInsert -1 );
    --itemToInsert;
    If( (itemToInsert >= 1 ) &&
    ( mList[ itemToInsert ] <= mList[ itemToInsert -1] ) )
    {
            insertItem( itemToInsert ); //recurse
    }
}
```

Here's a breakdown of the code:

```
public void insertItem(int ItemToInsert)
{
    *swap( itemToInsert,itemToInsert -1 );
```

1. This method takes an integer(*itemToInsert*) representing the offset of the item to insert. Then the function swaps it immediately to the left.

```
--itemToInsert;
```

2. For example, if the *itemToInsert* variable has the value 5, you will need to swap the items at the offset 5 and 4. Having made that swap you now want to examine item 4 to see if it is bigger or smaller than item 3. To do so, you must decrement the *itemToInsert* variable.

```
    If( (itemToInsert >= 1 ) &&
    ( mList[ itemToInsert ] <= mList[ itemToInsert -1] ) )
    {
            insertItem( itemToInsert ); //recurse
    }
```

3. Now you must compare again, if you find that item 3 is smaller, you must recurse the method.

* The swap function is not included here, assuming that you understand the logic of a swap function.

4. You continue to recurse until you find the right spot or the list runs out.

Insertion Sort Performance

Big-O Notation: $N^2/4$

Selection Sort

- The Selection Sort compares the first index *j* through the list and if it finds a smaller value than *minimum* it swaps it.
- The j is incremented and the process starts all over again.

```
public void SelectionSort()
{
        int minimum; int nItems = mList.length()-1;

        for( int j = 0; j < nItems-1; ++j)
        {
                minimum = j;
                for( int w = j + 1; w < nItems; ++w)
                {
                        if( mList[w] < minimum)
                        {
                                minimum = j;
                        }
                }
                if( minimum != j)
                {
                        swap( minimum, j);
                }
        }
}
```

Here's a breakdown of the code:

This method works by using an outer loop to iterate through the list; and an inner loop to compare the items. The outer loop:

```
        for( int j = 0; j < nItems-1; ++j)
        {
```

Then you make an inner loop:

```
                for( int w = j + 1; w < nItems; ++w)
                {
```

If the loop finds a smaller value, then minimum is reset.

```
if( mList[w] < minimum)
{
        minimum = j;
}
```

Then if minimum doesn't equal the value of *j*, the two values are swapped:

```
if( minimum != j)
        {
                swap( minimum, j);
        }
```

Selection Sort Performance

Big-O Notation:

Average: (N * (N -1)) / 2
Worst-case: N – 1

Bubble Sort

- Bad choice for a sort, but shows up on a few tests.
- The bubble sort works by comparing the first two elements in the list. If the second element is larger, then they are swapped. Then it compares the second and third item, and so on.

Here is the Bubble Sort code:

```
Public void BubbleSort()
{
For(int j = nItems -1; j >= 1; j++)
{
        For(int w =1; w <= j; w++)
        {
                If(mList[w] < mList[w-1])
                {
                        Swap( j, j-1);
                }
        }
}
}
```

Bubble Sort Performance

Big-O Notation:

$N^2 / 2$

System.out.println()

- Memorize these!

  \n - new line
  \t - horizontal tab
  \r - carriage return; moves cursor back to start.(Not a new Line!!!)
  All characters will be overwritten!!!
  \\ - backslash
  \" - used to print a double quote

Tip:

If a println() statement starts out with a ("") character, then all output becomes strings.

Ex:

```
Int x = 3, y = 4;
String s = "UIL";
System.out.println("" + x + y + s);
```

Displays:

34UIL

However:

Ex:

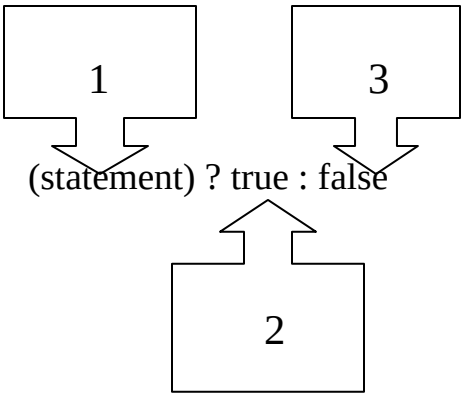
```
Int x = 3, y = 4;
String s = "UIL";
System.out.println(x + y + s);
```

Displays:

7UIL

?:

?: is a unary operator that is a condensed version of an if statement.



1. The statement that is evaluated to either true or false.
2. If the statement evaluates to true, then this value is returned.
3. If the statement evaluates to false, then this value is returned.

Ex:

System.out.println(StudentGrade >= 60 ? "Passed" : "Failed");

If the student has an 80 average, then "Passed" is printed. If he has a 50 average, then "Failed" is printed.

Switch

- Can **ONLY** accept: byte, char, short, int values
- Exits immediately if it hits a break statement. Continues if it doesn't; or "falls through", executing every statement below.
- Uses the default only if a case is not met.
- Characters must be in single quotes, ‘’.

Ex:

```
Int x = 1;
Switch(x)
{
Case 0: x=2;
        Break;
Case 1: x=3;
        Break;
Default: x=-1;
}
```

X = 2

Ex:

```
Int x = 0;
Switch(x)
{
Case 0: x=-1;
Case 1: x=32767;
Default: x=-1;
}
```

X = 32767!!!!

Ex:

```
char ch = ‘x’;
Switch(ch)
{
Case 0: ch=’j’; break;
Case 1: ch=’w’; break;
}
```

Ch still equals ‘x’!!!!

BASE ARITHMETIC

Converting decimal to binary

To convert decimal to binary is also very simple, you simply divide the decimal value by 2 and then write down the remainder, repeat this process until you cannot divide by 2 anymore, for example let's take the decimal value 157:

157 ÷ 2 = 78 with a remainder of **1**
78 ÷ 2 = 39 with a remainder of **0**
39 ÷ 2 = 19 with a remainder of **1**
19 ÷ 2 = 9 with a remainder of **1**
9 ÷ 2 = 4 with a remainder of **1**
4 ÷ 2 = 2 with a remainder of **0**
2 ÷ 2 = 1 with a remainder of **0** <--- to convert write this remainder
1 ÷ 2 = 0 with a remainder of **1** first.

Next write down the value of the remainders from bottom to top (in other words write down the bottom remainder first and work your way up the list) which gives:

10011101 = 157

To Convert Hexadecimal to Binary

Take the number $A7_{16}$ as an example (Hexadecimal numbering runs from (0-F))

Separate it into individual digits

A is the tenth value in the hexadecimal numbering system. Write ten in binary

A becomes 1010

Now write 7 in binary 7 becomes 0111

A + 7 = 1010 + 0111
So $A7_{16}$ = 10100111

To Convert Binary to Hexadecimal

Break Binary number into blocks of 4 digits

Example 10111101

broken in two becomes 1011 and 1101

Convert 1011 to decimal (1*8 + 0*4 + 1*2 + 1*1) =11 Convert 11 to Hexadecimal = B
Convert 1101 to decimal (1*8 + 1*4 + 0*2 + 1*1) =13 Convert 13 to Hexadecimal = D

That means that 10111101 is equal to BD_{16}

To Convert Decimal numbers to Hexadecimal

Method 1

2 Step Process

Convert Decimal to Binary then Convert Binary to Hexadecimal

Method 2

Determine largest possible power of 16 that is less than the decimal number.

Examples 3899

Largest power of 16 less than 3899 is 256 (next power of 16 is 4096)

Divide 3899 by 256

Quotient is 15 Remainder is 59

Divide 59 by 16

Quotient is 3 Remainder is 11

Convert each quotient to hexadecimal 15 = F 11 = B

Results $F3B_{16}$

Obviously method one is preferred

To Convert Octal numbers to Binary numbers

Example 351_8

Separate into individual numbers and convert to binary (3 places only)

3 = 011 5 = 101 1 = 001

So 351_8 = 011101001

Example 765_8

7 = 111 6 = 110 5 = 101

So 765_8 = 111110101

To Convert Binary numbers to Octal numbers

Example 01011101 (special note – will be breaking into groups of three so you may need to add a zero in front)

Adding a zero in front of 01011101 making it 001011101

Breaking into groups of three

001 011 101

Convert each group to decimal

001 = 1 011 = 3 101 = 5

So 01011101 = 135_8

Example 111000101

111 000 101

111 = 7 000 = 0 101 = 5

So 111000101= 705_8

To find the sum of numbers in mixed format

3 Step Process

1. Convert all values to Binary

2. Line up and do binary arithmetic

3. Convert Binary answer to Octal and Hexadecimal values

Example
$6A_{16} + 10110110_2 + 74_8$

Convert
6 = 0110 A = 1010 = 01101010 7 = 111 4 = 100 = 111100

Line them up and do arithmetic

RULES FOR BINARY ARITHMETIC

1. Only 1 and 0 are allowed

2. When summing each column:

Sum of 2 – put down a 0 and carry a 1
Sum of 3 – put down a 1 and carry a 1
Sum of 4 – put down zero and carry a 2

3. Work as far to the left as needed

Original Number									
	256	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
6A (base 16)		**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**
10110110		**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**
74(base 8)				**1**	**1**	**1**	**1**	**0**	**0**
	---	---	---	---	---	---	---	---	---
	1	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**

So $6A_{16} + 10110110_2 + 74_8 = 101011100$

IF THAT ISNT AN ANSWER CHOICE – THEN YOU MUST CONVERT IT

Convert to Octal

101 = 5 011= 3 100=4 **so answer = 534_8**

Convert to Hexadecimal (must use groups of 4 so need to add 000 to front of 1010111000

0001 = 1 0101 = 5 1100 = 12 **so it equals $15C_{16}$**

Bitwise Operators

1. Analyze the problem.
2. Breakdown into binary.
3. Line up the binary.
4. Solve.
5. Translate back into decimal.

& - Bitwise AND:

The resulting bits are set to 1 if both bits are 1.

Ex:

```
        6 & 4          //Statement
6 -     110             //Broken into binary
4 -     100
        =
        100             //The resulting binary number
          4             // Translated back into decimal
```

| - Bitwise Inclusive OR:

The resulting bits are set to 1 if either bit is 1.

Ex:

```
        25 | 7         //Statement
25 -    11001           //Broken into binary
7 -     111
        =
        11111           //The resulting binary number
          31            // Translated back into decimal
```

^ - Bitwise Exclusive OR:

The resulting bits are set to 1 if only one bit is 1.

Ex:

```
        25 ^ 7          //Statement
25 -    11001           //Broken into binary
7 -     111
        =
        00110           //The resulting binary number
          6             // Translated back into decimal
```

<< - Left Shift:

Shift the bits of the first operand to the left by the number specified by the second operand; fill from right with 0s

Ex:

```
        6 << 2          //Statement
6 -     110             //Broken into binary
        =
        11000           //The resulting binary number
          24            // Translated back into decimal
```

>> - Signed Right Shift:

Shift the bits of the first operand to the right by the number specified by the second operand; if the operand is negative, fill from left with 1s, otherwise fill in with 0s.

Ex:

```
        6 >> 2          //Statement
        110             //Broken into binary
        =
        001             //The resulting binary number
          1             // Translated back into decimal
```

Ex:

```
        -6 >> 2         //Statement
        110             //Broken into binary
        =
        11110           //The resulting binary number
          30            // Translated back into decimal
```

>>> - Unsigned Right Shift:

Shift the bits of the first operand to the left by the number specified by the second operand; fill from left with 0s

Ex:

```
        -6 >>> 2        //Statement
        -110            //Broken into binary
        =
        -001            //The resulting binary number
          -1            // Translated back into decimal
```

~ - Bitwise Complement:

All 0s are set to 1, all 1 bits are set to 0.

Ex:

```
        ~25         //Statement
        11001       //Broken into binary
        00110       //The resulting binary number
          6         //Translated back into decimal
```

String Tokenizer

- StringTokenizer is used to parse sentences into strings
- It removes all punctuation marks and white space

.nextToken() : Returns the next token
.hasMoreTokens() : Returns true or false depending upon whether you have used up your tokens

Ex:

```
StringTokenizer tk = new StringTokenizer("Hello Everyone");
System.out.println(tk.nextToken());
System.out.println(tk.hasMoreTokens());
System.out.println(tk.nextToken());
```

Displays:

```
Hello
true
Everyone
```

Exceptions

- Exceptions are used to capture errors.

Ex:

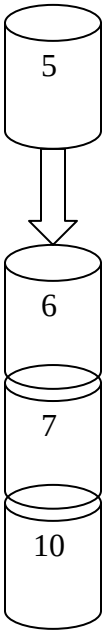
```
Try
{
        // code goes here
}
Catch( Exception e)
{
        // if an exception is thrown, this code is executed
}
Finally
{
        // runs whether or not an exception is thrown
}
```

Stacks

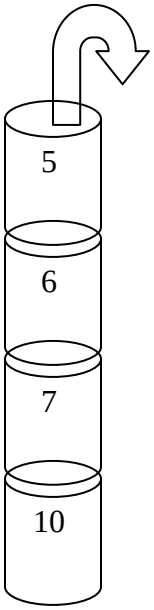
A stack operates like a stack of cups.

- Operates in LIFO order
- push() puts a new cup on top of the stack
- pop() pops a cup off of the top of the stack
- peek() looks at the bottom of the stack

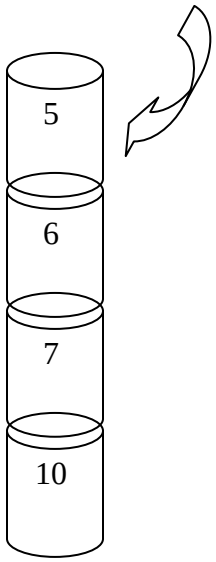
push(5)



pop()



peek()



Ex:

```
Stack s = new Stack();
Object obj;
s.push("a");
s.push("b");
s.push("c");
obj = s.peek();
s.push(obj + "d");
s.push(obj + "x" + obj);
while(!isEmpty())
{System.out.print(s.pop() + " ");
```

Displays:

cxc cd c b a

Queue

Think of a queue as a line for a movie. As a person come up (*enqueue*) the line (*queue*) increases. As a person is let into the movie (*dequeue*) the line gets shorter.

Appendix A: Java Keywords

Java has forty-eight keywords. You cannot declare the name of a variable, method, or class to be one of these keywords. If you try to do so, the compiler may give a rather puzzling error message.
The keywords come in the several categories, listed below. Where the keywords are listed on two lines, the second line consists of keywords defined in the text in one place and never used anywhere else in the text (thus you do not need to remember them).

8 primitive types of values:

```
boolean char int double long
byte short float
```

3 constant values (technically these are two boolean literals and one null literal instead of keywords):
`true false null`

14 statement indicators (the first word in the statement except for `else` and `catch`):
```
if else while do for return try catch throw
switch case default continue break
```

11 modifiers (can appear before the return type of a method):
```
public private static final abstract
protected synchronized native volatile transient strictfp
```

6 global (can be used outside of a class body):
```
class interface extends implements import
package
```

7 miscellaneous:
```
void new this super throws instanceof
finally
```

2 with no meaning, reserved so that people who accidentally put these C++ words in their programs will receive useful error messages:
`const goto`