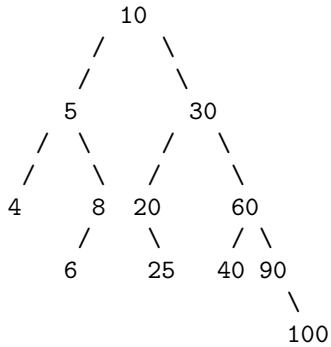


Problem 1 (a) Consider the following tree.



The height of the tree is: **Answer:** 4

The depth of the node 90 is: **Answer:** 3

The preOrder traversal of the tree is: **Answer:** 10 5 4 8 6 30 20 25 60 40 90 100

(b) Give a useful O -estimate of the run time of the following method:

```

double f(int n) {
    if (n <= 0) return 1.0;
    double ans = f(n/2) * 2;
    for (int i = 1; i <= (n + 3); i++)
        for (int j = 1; j <= (n + 2); j++)
            ans += i / j;
    for (int k = 1; k <= 3; k++)
        ans -= f(n/2 - k);
    return ans;
}
  
```

Answer:

The non-recursive instructions run in time $O(n^2)$. There are 4 recursive calls to problems with input size $n/2$. This gives an effective recursive cost of $O(n^{\log_2(4)}) = O(n^2)$. According to the Master theorem, the method has a run time of $O(n^2 \log(n))$.

Problem 2 The interface Deque can have array based and linked implementations. Partial versions of implementation classes follow. Supply implementations for the method addRear for both.

```

class ArrayDeque implements Deque // circular array based
{
    private Object data[];    private int front, rear, size, capacity;
    public ArrayDeque() {
        capacity = 1000; data = new Object[capacity];
        front = size = 0; rear = 1; }
    // method code omitted here
}

class DNode { private Object data; private DNode prev, next;
    // standard constructors, get and set method code omitted here
}

class LinkedDeque implements Deque
{
    private DNode front, rear;    private int size;
  
```

```

    public LinkedDeque() { front = rear = null; size = 0; }
    // method code omitted here
}

```

(a) Implement the method addRear for the array based class.

Answer:

```

public void addRear(Object x) {
    if (size == capacity) throw new RuntimeException("Full");
    data[rear++] = x;
    if (rear == capacity) rear = 0;
    size++;
}

```

(b) Implement the method addRear for the linked class.

Answer:

```

public void addRear(Object x) {
    DNode newNode = new DNode(x, rear, null);
    if (rear == null) front = newNode;
    else rear.setNext(newNode);
    rear = newNode;
    size++;
}

```

Problem 3 Write a Java implementation for the following problem. You can make use of whichever of the ADTs Stack, Queue, and Deque that you need. Assume that these ADTs are already implemented, compiled and available in files Stack.class, Queue.class, Deque.class.

Input is read as a sequence of strings from the terminal (*System.in*). The program continues until the special string * is found on the input. Input data is stored until one of the special strings +, -, 0 is encountered. These special strings cause the stored data to be printed forwards and removed, printed backwards and removed, or removed without printing.

For example, if the input is:

```

q w e r + z x c v 0 t y
u i p - a s d 0 q w e *

```

the output would be:

```

q w e r
p i u y t

```

Answer:

```

import java.util.*;
class Prob3 {
    public static void main(String args[]) {
        try {
            Scanner s = new Scanner(System.in);
            Deque d = new Deque();
            while (true) {
                String x = s.next();
                if (x.equals("+")) {
                    while(!d.empty()) System.out.print(d.removeFront() + " ");
                    System.out.println();
                }
                else if (x.equals("-")) {
                    while(!d.empty()) System.out.print(d.removeRear() + " ");
                    System.out.println();
                }
            }
        }
    }
}

```

```
    }
    else if (x.equals("0")) {
        while(!d.empty()) d.removeRear();
    }
    else if (x.equals("*")) {
        throw new RuntimeException("Done");
    }
    else d.addRear(x);
}
} catch (Exception e) { }
}
```