| COS 126 | General Computer Science | Spring 2007 |
|---|---|---|

# Exam 2 Solutions

1. **Data types.**

   (a) A *data type* is a set of values and a set of operations on those values. This definition was given repeatedly both in lecture and in the textbook.

   (b)

   C A data type whose representation is hidden from the client.

   D After constructing an object of this type, you cannot change its value.

   B `int`

   A When an object of this type is passed to a function, the function can, in general, change its value.

   A. Reference type.

   B. Primitive type.

   C. Abstract (encapsulated) data type.

   D. Immutable data type.

   (c) Programmers use data types to:

   - Make programs easier to read and understand.
   - Make programs easier to debug (since debugging is restricted to smaller pieces of code).
   - Make programs easier to maintain and improve.
   - Make code easier to reuse (without having to copy and re-implement it).
   - Make designing programs easier to manage (since each programmer can work independently on their own part).

2. **Algorithms and data structures.**

   A Implement recursion.

   E Parse an NCBI genome data file.

   A Evaluate an arithmetic expression.

   A Implement the back button in a browser.

   D Model pairwise relationships in facebook.

   C Search for an IP address given a domain name.

   B Model the buffer in the Karplus-Strong algorithm for simulating the pluck of a guitar string.

   A. Stack

   B. Queue

   C. Symbol table

   D. Graph

   E. Regular expression

3. **Floating point precision.**

   Catastrophic cancellation. This is the same example from lecture.

   The underlying reason is that only a finite number of real numbers can be exactly represented in floating point number. So calculations involving floating point numbers are subject to roundoff error. This problem is magnified when subtracting two nearly equal numbers, resulting in a devastating loss of precision. For example, if x = `1.1e-8`, then `Math.cos(x)` is about 0.999999999999999889, which is accurate to 16 decimal places. However, when we compute `1.0 - Math.cos(x)`, the result is about `1.1102e-16`, which isn't even accurate to one decimal place!

4. **Creating data types.**

   There are 4 errors.

   - Constructors have no return type, not even `void`.
   - The constructor should not declare local variables `x`, `y`, and `z` since this hides the instances variables with the same name.
   - The signature of `distanceTo()` should not include the modifier `static`.
   - The `toString()` method must return a `String`.

```
public class Point3D {
    private double x, y, z;

    // create a point (x0, y0, z0)
    public Point3D(double x0, double y0, double z0) {
       x = x0;
       y = y0;
       z = z0;
    }

    // return the Euclidean distance between this point and q
    public double distanceTo(Point3D q) {
        double dx = x - q.x;
        double dy = y - q.y;
        double dz = z - q.z;
        return Math.sqrt(dx*dx + dy*dy + dz*dz);
    }

    // return a string representation of this point
    public String toString() {
        String s = "(" + x + ", " + y + ", " + z + ")";
        return s;
    }
}
```

5. **Analysis of algorithms.**

   (a) 14050 seconds. (almost 4 hours!)
   The key observation is that (for large values of `N`) the running time *quadruples* when the input size increases by a factor of 2. This strongly suggests the running time grows proportional to $N^2$. Solving an instance of size 800,000 will take roughly $10^2$ times as long as one of size 80,000.

   (b) 215
   With the faster algorithm, the running time *doubles* when the input size increases by a factor of 2. This strongly suggests that the running time grows proportional to $N$. Solving an instance of size 800,000 will take roughly 10 times as long as one of size 80,000.

6. **Linked structures.**

```
public double distance() {
    double sum = 0.0;
    for (Node x = first; x.next != null; x = x.next) {
        sum = sum + x.p.distanceTo(x.next.p);
    }
    return sum;
}
```

7. **Modular programming.**

```
public class Ball3D {
    private Point3D center;
    private double radius;

    // construct a ball centered at c, with radius r
    public Ball(Point3D c, double r) {
        center = c;
        radius = r;
    }

    // does the ball contain the point p?
    public boolean contains(Point3D p) {
        return p.distanceTo(center) <= radius;
    }

    // return the ball's volume = 4/3 pi r^3
    public double volume() {
        return 4.0 / 3.0 * Math.PI * radius * radius * radius;
    }
}
```

8. **Theory of computation.**

E Universal

C Undecidable

G Duality

H Church-Turing thesis

I Turing machine

D P

A NP

F NP-complete

A. The set of all search problems (i.e., solution can be checked in polynomial time).

B. A set of search problems that are believed to have no polynomial time solution.

C. A problem that cannot be solved by a Turing machine.

D. The set of all search problems that can be solved in polynomial time.

E. One machine can do any computational task.

F. If you can solve a problem in this class in polynomial time, then P = NP.

G. Programs and data are each encoded as sequences of bits and can be used interchangeably.

H. Anything computable in this universe can be computed by a Turing machine.

I. A simple, universal, model of computation.

9. **DNA analyzer.**

```
public class AnalyzeDNA {
    public static void main(String[] args) {

        // read in the command-line argument k
        int k = Integer.parseInt(args[0]);

        // read in the DNA string from standard input
        String dna = StdIn.readString();
        int N = dna.length();

        // create a symbol table with String keys and Integer values
        ST<String, Integer> st = new ST<String, Integer>();

        // populate symbol table with kgrams and their frequencies
        for (int i = 0; i <= N - k; i++) {
            String kgram = dna.substring(i, i + k);
            if (st.contains(kgram)) st.put(kgram, st.get(kgram) + 1);
            else                    st.put(kgram, 1);
        }

        // print out kgrams and their frequencies
        for (String kgram : st) {
            System.out.println(kgram + " " + st.get(kgram));
        }

    }
}
```

10. **Circuits**

| $x$ | $y$ | output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

(a) $xy + x'y'$

(b) There are $2n = 128$ inputs, so the truth table has $2^{128}$ rows.

(c) We connect each pair of bits $x_i$ and $y_i$ to an *xnor* gate. The output of the $i$th *xnor* gates is 1 if and only if $x_i = y_i$.