

Bioinformatics Programming “Answers”

Problem 1

Part 1:

Let's think of an algorithm for the following problem.

Input: a number

Output: the Fibonacci number corresponding to it

```
Fibonacci(n)
  if (n == 0) {
    return 0;
  } elseif (n == 1) {
    return 1;
  } else {
    x = Fibonacci(n-1)
    y = Fibonacci(n-2)
    return x + y
  }
```

or, to avoid repeated calculations:

```
Fibonacci2(n)
  initialize @myArray
  myArray[0] = 0
  myArray[1] = 1
  for (i = 3; i <= n; i++)
    myArray[i] = myArray[i-1] + myArray[i-2];
  return myArray[n];
```

or, even better, to avoid repeated calculations and using unnecessary memory:

```
Fibonacci2(n)
  initialize variables u and v
  u = 0
  v = 1
  for (i = 3; i <= n; i++) {
    t = u + v // store the fibonacci number for i in t
    u = v // save the value of (i-2) into u
    v = t // save the value of (i-1) into v
  }
  return v; // return the last Fibonacci number calculated (n)
```

Part 2:

Now write the program for computing the Fibonacci number for a given input value, in Perl. Feel free to make a copy of one of the programs in the notes from yesterday and modify it!

```
sub fibonacci {
    my $n = shift @_; // retrieve input
    if ($n == 0) {
        return 0;
    } elsif ($n == 1) {
        return 1;
    } else {
        return fibonacci($n-1) + fibonacci($n-2);
    }
}
```

or

```
sub fibonacci2 {
    my $n = shift @_; // retrieve input
    my @fibArray = ();
    push (@fibArray, 0);
    push (@fibArray, 1);
    for ($i = 2; $i <= $n; $i++) {
        push(@fibArray, $fibArray[$i-1] + $fibArray[$i-2]);
    }
    return $fibArray[$n];
}
```

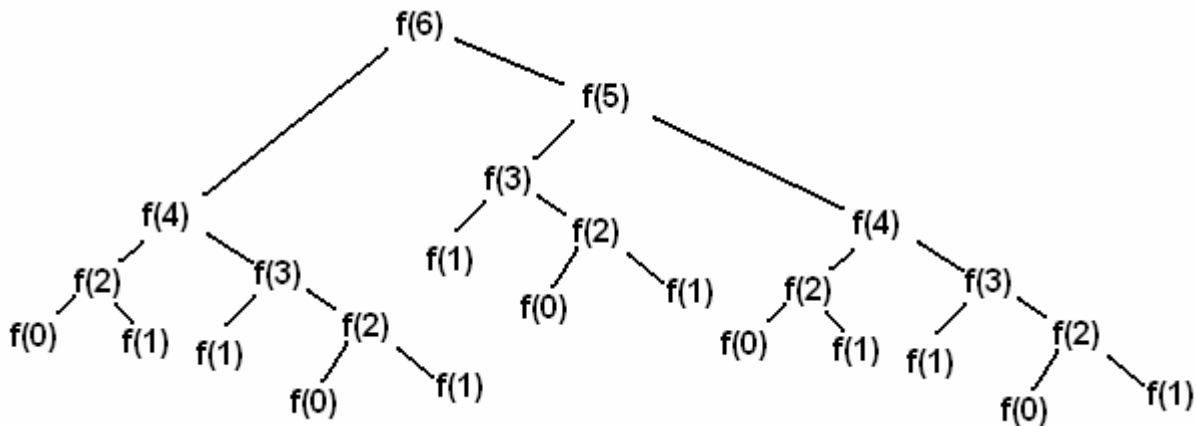
or

```
sub fibonacci3 {
    my $n = shift @_; // retrieve input
    my $u = 0;
    my $v = 1;
    for ($i = 2; $i <= $n; $i++) {
        $t = $u + $v;
        $u = $v;
        $v = $t;
    }
    return $v;
}
```

Part 3:

What is the running time of your algorithm?

Looking at the first algorithm, let us see the calculations for when $n = 6$;



This shows that there is an exponential growth in calculations in terms of n , making it very inefficient. Thus, we take a look at the second algorithm, which is actually not recursive, but iterative. Fibonacci2 actually runs in $O(n)$ time because of the saving of repeated calculations in `@fibArray`., however it uses an enormous amount of unneeded memory. So Fibonacci3, which also runs in $O(n)$ time, is the most efficient and accurate of the three.

Problem 2

Now let's try a bioinformatics problem. The GC content of a sequence is defined by the following formula, where each capital letter is the number of occurrences of that letter.

$$\frac{G + C}{A + T + G + C} \times 100$$

Input: a DNA sequence in FASTA format

Output: its GC content.

```
#!/usr/bin/perl
```

```
my %resHash; // initialize a hash to hold the residues and their numbers
```

```
my $i, $res;
```

```
// initialize the values of the hash
```

```
$resHash{A} = 0;
```

```
$resHash{G} = 0;
```

```
$resHash{C} = 0;
```

```
$resHash{T} = 0;
```

```

open(FILE, $ARGV[0]); // open the file

while (<FILE>) { // loop each line
    if (substr($_, 0, 1) neq ">") { // if the first character of the line does not begin with ">"
        chomp($_); // remove the newline character
        my $len = length $_; // get its length
        for ($i=0; $i<$len; $i++) { // loop for the number of residues in the line
            $res = substr($_, $i, 1); // get the residue at the $i-th position
            $resHash{$res}++; // add 1 to the value for the residue $res
        }
    }
}

// print the values for each residue
foreach (keys %resHash) {
    print $_, " ", $resHash{$_}, "¥n";
}

// compute the GC content
my $gc = 100*($resHash{"G"} + $resHash{"C"})/($resHash{"A"} + $resHash{"C"} +
$resHash{"G"} + $resHash{"T"});

// print the GC content
print "$gc¥n";

```

Problem 3

Ok, finally for a more challenging problem.

Input: two amino acid sequences in FASTA format

Output: its alignment using the dynamic programming algorithm taught yesterday.

Hint: use your own pre-defined symbols, numbers, or strings for up-arrow, left-arrow, etc.