Exercises for lab 3 of CS3101b

Instructor: Marc Moreno Maza, TA: Xiaohui Chen

Thursday 16 January 2015

1 Exercise 1

The following is a `C` function for computing the sequence of the Fibonacci numbers (in a naive way).

```
double fib(int n){
  if(n<=2)
    return(1.0);
  else
    return(fib(n-2)+fib(n-1));
}
```

1. Write a Julia program that computes `fib(n)`
2. Using the `@time` macro, measure the running times of your Julia function `fib(n)` for n between `35` and `45.`
3. If you are a Matlab user, here's `fib(n)` in Matlab for you to perform the same measurement.

   ```
   function f=fib(n)
     if n <= 2
       f=1.0;
     else
       f=fib(n-1)+fib(n-2);
     end
   end
   ```

2 Exercise 2

The following is a `C` function for computing the product of two square matrices (in a naive and inefficient way).

```
#define M 500
void mmult(double A[M][M],double B[M][M], double C[M][M]){
```

```
  //double C[M][M];
  int i,j,k;
  for(i=0; i<M; i++)
    for(j=0; j<M; j++){
      C[i][j] = 0;
      for(k=0; k<M; k++)
C[i][j] += A[i][k]*B[k][j];
    }
}
```

1. Write a Julia program that computes `mmult(A,B)` where `A` and `B` are two square matrices of the same order `M` (using the same naive and inefficient algorithm as in C).

2. Using the `@time` macro, measure the running times of your Julia function `mmult(A,B)` for `M` equal to `500`, `1000`, `1500`, `2000`. Your input matrices will be randomly generated using `rand(M,M)`.

3. If you are a Matlab user, here's `mmult(A,B,C)` in Matlab for you to perform the same measurement.

   ```
   function C=mmult(A,B,C)
     [M,N] = size(A);
     for i=1:M
       for j=1:M
         for k=1:M
           C(i,j) = C(i,j) + A(i,k)*B(k,j);
         end
       end
     end
   end
   ```

3 Exercise 3

The following Julia session implements a famous algorithm for sorting called *quicksort*. Look at its wikipedia page to learn how this algorithm works!

`http://en.wikipedia.org/wiki/Quicksort`

```
function qsort!(a,lo,hi)
    i, j = lo, hi
    while i < hi
        pivot = a[(lo+hi)>>>1]
        while i <= j
            while a[i] < pivot; i = i+1; end
            while a[j] > pivot; j = j-1; end
            if i <= j
```

```
                a[i], a[j] = a[j], a[i]
                i, j = i+1, j-1
            end
        end
        if lo < j; qsort!(a,lo,j); end
        lo, j = i, hi
    end
    return a
end

function sortperf(n)
    qsort!(rand(n), 1, n)
end
@time sortperf(5000)
```

1. Go through the code and make sure you agree that it is an implementation of the algorithm presented in the wikipedia page.

2. Record the running time of `sortperf`(2^e∗1000000) for $e = 0, 1, 2, 3, 4, 5, 6, 7, 8$.

3. Are your results coherent with the theoretical prediction (see the section *Formal analysis* in the wikipedia page) that sorting of an array of size n with *quicksort* runs in a time asymptotically proportional to $O(n\log(n))$?

4 Exercise 4

Read the wikipedia page dedicated to the merge-sort algorithm:

`http://en.wikipedia.org/wiki/Merge sort`

1. Write a Julia implementing the merge-sort algorithm and following the style and presentation done for *quicksort*.

2. Compare the running times of both sorting algorithms