Exam Preparation

Written exam

Tobias Andersson Gidlund

tobias.andersson.gidlund@lnu.se

Exam information

- First exam:
  - Friday March 22
- Second exam:
  - Saturday May 18
- Third exam:
  - In August, will be announced later
- Exact time and place will be published (one week in advance) at:
  - `https://lnu.se/student` (follow the links)
- Register for the exam at the same URL.

Excercise 1 (Recursion)

- In the Fibonacci sequence the first two numbers are 0 and 1 and the others are the sum of the two previous numbers.

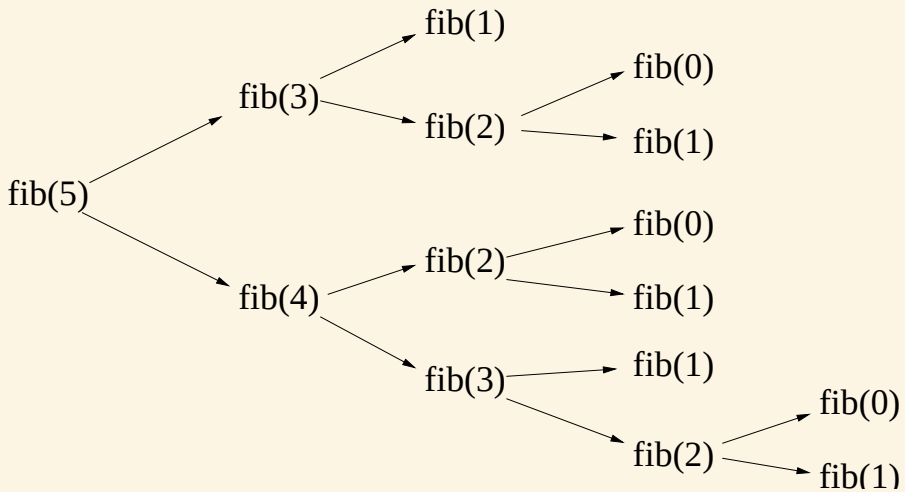
  `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...`

  Write a recursive Java method `int fib(int n)` that computes the *n*:th number in the Fibonacci sequence.
- Why is the above recursive method bad if you would like compute the first 50 numbers in the fibonacci sequence? Motivate your answer using an example. Also, present Java code for a much better *non-recursive* approach to compute and print the the first 50 numbers in the fibonacci sequence.
- Write a recursive method int mult(int a, int b) that computes the multiplication a $\times$ b with the use of addition. You can assume that both a and b are positive.

Answer excercise 1a

```
public int fib(int N) {
   if (N==0)
      return 0;
   else if(N==1)
      return 1;
   else
      return fib(N-1) + fib(N-2);
}
```

- Bad (but beautiful) since it gices an exponential number of calls $O(2^n)$.



Answer excercise 1b

- A non-recursive version:

```
public static void main(String[] args) {
   int N = 90;                    // N = 100 does not work, why?
   long fm2 = 0, fm1 = 1, f;
   for (int i = 2;i<N;i++) {
      f = fm1 + fm2;
      System.out.println(i+"\t"+f);
      fm2 = fm1;
      fm1 = f;
   }
}
```

- Calculates the fibonacci seqence in $O(N)$.

Answer excercise 1c

- A method for recursively calculating the multiplication of a and b.
- Works by knowing that $a \times b = b + (a-1) \times b$
- Base case is $0 \times b = 0$

```
public int mult(int a, int b) {
    if (a == 0)
        return 0;
    else
        return b + mult(a - 1,b);
}
```

Excercise 2 (Algorithms)

- What is an algorithm? What properties do we expect from an algorithm? Why do we use algorithms?
- Write an algorithm in pseudo code that is searching for an integer *N* in a sorted list (lowest first) using the method *binary search*.

Answer excercise 2a

- An algorithm is a step-by-step description of how to solve a problem.
- It should:
  1. Give an unambiguous result
     $\rightarrow$ only one result for each input
  2. Be unambiguously presented
     $\rightarrow$ a precise formulation that can't be misinterpreted
  3. Terminate after a finite number of steps
     $\rightarrow$ no infinite computations.
- Why use algorithms?
  - Document problem solutions
  - Communicate problem solutions
  - Compare problem solutions (time complexity!)
  - Sketchy algorithms are a good preparation before programming.

Answer excercise 1b

- A recursive version of binary search initially called as:

  `BinSearch(L,N,1,MAX)`

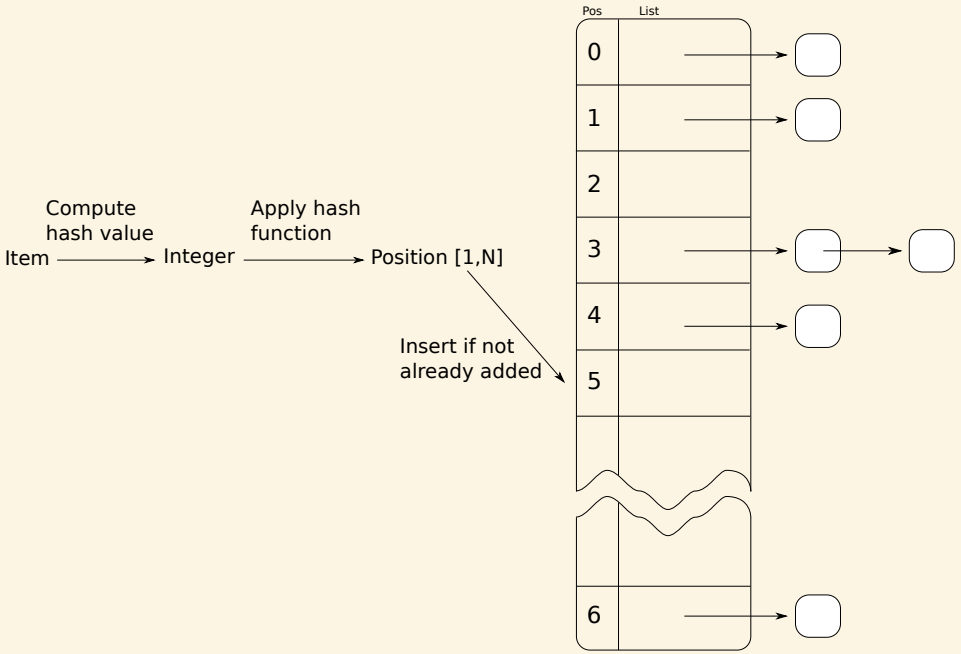
  where *L* is the list to be searched (sorted, lowest first), *N* is the value to search for, 1 is the first index in the list, and *MAX* is the last index in the list.

```
if Max < Min then
   return false
else
   MidPos = (Min + Max)/2
   MidValue = L.get(MidPos)
   if N = MidValue then
       return true
   else if N < MidValue then
       return BinSearch(L,N,Min,Mid-1)
   else
       return BinSearch(L,N,Mid+1,Max)
   end if
end if
```

Excercise 3 (Data Structures)

- *Hashing* is an implementation technique often used when implementing certain data structures. Describe how hashing works and why it is used.
- The Java interface `java.util.Map` describes a *map* (or *table*) data structure. What is a map and what operations do we associate with a map? Show with a Java example how maps can be used in Java.

Answer excercise 3a



Hashing: Assume table with *N buckets* (A pair position/list)

- Associate each element with a hash value (an integer): `element --> int`
- Apply hash function (maps hash value to a bucket): `int --> bucket`
- Add to the bucket (the list part) if not already added

If all elements are evenly distributed, `add`/`contains`/`remove` will execute in $O(1)$

Answer excercise 3b

- Maps is a set of key/value pairs. The two key operations are put and get

```
V put(K key, V value) // Associates the specified value with the specified
↪  key
V get(K key)          // The value to which the specified key is mapped
```

- Usage:

```
Map<Integer,String> map = new HashMap<Integer,String>();

map.put(64, "C64");
map.put(128, "C128");
map.put(512, "Atari ST");
map.put(1024, "Atari Mega ST");

System.out. println ("Value for 128: " + map.get(128)); // Prints C128
System.out. println ("Value for 512: " + map.get(512)); // Prints Atari ST
```

Excercise 4 (Inheritance)

- What is *binding* in object-oriented programming languages. What is the difference between *static* and *dynamic* binding? How does binding work in Java?
- What separates an *abstract* class from an *interface* and an *ordinary* class in Java?

Answer excercise 4a

- Binding $\rightarrow$ Process of associating a call `a.m(..)` to a concrete target method `m`.
- Static binding:
  - A call `a.m(...)` is resolved using the static type (the declared type) of `a`
  - Binding takes place at compile time
- Dynamic binding:
  - A call `a.m(...)` is resolved using the dynamic type (the current value type) of `a`
  - Binding takes place at run-time
  - Java uses dynamic binding in all calls to non-static methods
  - The target method in a call `a.m(...)` is determined by the type of the object referenced by `a`. Assume type `X`:
    1. If `m(..)` is declared in `X`, call `X.m()`
    2. If not, use `m(...)` in the first super class where `m()` is defined.

Answer excercise 4b

- An abstract method has no implementation
- If a class has an abstract method, it must be declared as `abstract`

```
public abstract class Shape { // An abstract super class
    protected int x, y;    // x- and y-position
    public Shape(int x_pos, int y_pos) {x = x_pos; y = y_pos;}
    public void move(int x_pos, int y_pos) {x = x_pos; y = y_pos;}
    public abstract double getArea(); // Abstract methods
    public abstract void print();
}
```

- Abstract methods must be implemented by its subclasses
- It is not possible to create objects of abstract classes
- Interface $\rightarrow$ a completely abstract class $\rightarrow$ just signatures, no implementation
- Subclasses must implement their versions of abstract classes if they want to be concrete classes.

Excercise 5 (Binary Search Tree)

- Consider the Java source code fragment below representing a binary search tree for integers where the methods add and contains are incomplete. Write down Java code for what these two methods should look like.

```
public class IntBST {
    private BST root = null;

    public void add(int n) {
        if (root == null)
            root = new BST(n);
        else
            root.add(n);
    }

    public boolean contains(int n) {
        if (root == null)
            return false;
        else
            return root.contains(n);
    }
```

```
    private class BST {
        int value;
        BST left = null;
        BST right = null;

        BST(int val) { value = val; }

        void add(int n) {
            ...
        }

        boolean contains(int n) {
            ...
        }
    }
}
```

Answer excercise 5

- Recursive solutions

```
void add(int n) {
    if (n<value) {
        // go left
        if (left == null)
            left = new BST(n);
        else
            left.add(n);
    }
    else if (n>value) { // go right
        if (right == null)
            right = new BST(n);
        else
            right.add(n);
    }
// Neither < nor > ==> Duplicate
}
```

```
boolean contains(int n) {
    if (n<value) {
        if (left == null)
            return false;
        else
            return left.contains(n);
    }
    else if (n>value) {
        if (right == null)
            return false;
        else
            return right.contains(n);
    }
    return true; // Found!
}
```

Exam Tactics

- Before the exam:
  - About 50% will be Java programming, the rest will be theory
  - Focus on understanding the assignments
  - Focus on the lecture slides
  - Use the book to learn things you don't understand
- At the exam:
  - Read the entire exercise carefully!
  - Answer our questions. Nothing else!
  - If you don't understand the question, ask teacher.
  - Still in doubt and the teacher has left?
    $\rightarrow$ explain your problem, make an assumption, and continue.
  - Relax and do it properly. You have plenty of time.