Design and Analysis of Algorithms — Lab

R S Milton, C Aravindan, T T Mirnalinee
Department of CSE, SSN College of Engineering

Session 2: Recursion

1 Exponentiation

1. Construct a recursive algorithm `power (x, n)`, to compute x^n, x raised to a non-negative exponent $n \geq 0$. Use the recurrence relation:

$$x^n = \begin{cases} 1 & n = 0 \\ x \times x^{n-1} & \text{otherwise} \end{cases}$$

2. Construct an iterative algorithm for Question 1.

3. Construct a faster recursive algortithm `fast_power (x, n)` to compute the power of x raised to n, using the recurrence relation

$$x^n = \begin{cases} 1 & n = 0 \\ x \times x^{n-1} & n \text{ is odd} \\ x^{n/2} \times x^{n/2} & n \text{ is even} \end{cases}$$

4. Construct an iterative version of `fast_power (x, n)` algorithm of Question 3.

5. Construct a function `mat_mul (a, b)` that multiples two matrices `a` and `b` and returns the product matrix. Represent a matrix by a list of lists. The function should check for valid input, and raise execption if the input is invalid.

6. Modify `fast_power (x, n)` to `mat_power (x, n)` for computing the power of a matrix `x` raised to `n`.

7. Fibonacci numbers can be expressed using matrix notation:

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

   Using this idea, implement a function `fib (n)` to compute F_n, the nth Fibonacci number. Use `mat_power` for matrix expoentiation.

2 Insertion sort

Develop functions useful for implemeting `insertion_sort`.

1. Algorithm, `ordered_insert`, should insert the target at its right position in a sorted list and create a new sorted list. Implement `ordered_insert (u, v)` as a recursive function. `u` is an item, and `v` is a sorted list. The algorithm will have the same outline as `linear_locate`, and differ only in the way the solution is constructed from the subsolution.

```
ordered_insert (15, [5, 10, 20, 35, 50])
[5, 10, 15, 20, 35, 50]
ordered_insert (35, [5, 10, 20, 35, 50])
[5, 10, 20, 35, 35, 50]
ordered_insert (2, [5, 10, 20, 35, 50])
[2, 5, 10, 20, 35, 50]
ordered_insert (25, [])
[25]
```

3 Mergesort

Develop the following functions useful for implemeting `mergesort`.

1. Two sorted lists `u` and `v` are given as input. Construct a recursive algorithm `ordered_merge (u, v)` which inserts each item of `u` in its right position in `v` and constructs a new sorted list as the output. Design `ordered_merge (u, v)` using `ordered_insert` (Section 2, Question 1).

```
ordered_merge ([15, 40, 45], [5, 10, 20, 35, 50])
[5, 10, 15, 20, 35, 40, 45, 50]
ordered_merge ([30, 60], [20, 35, 50])
[20, 30, 35, 50, 60]
```

```
ordered_merge ([], [20, 35, 50])
[20, 35, 50]
```

2. Operation merge takes two sorted lists as arguments and results in a sorted list of all the items in the argument lists. Let us denote prepend by the operator `:` and merge by the operator ++ (it is not a standard notation – we use it in this lecture only). Then

```
  [2,3,8,9] ++ [1,4,5,7] =  [1,2,3,4,5,7,8,9]
```

   Algorothm `merge` merges two sorted lists as illustrated below. Implement function `merge`.

```
    [2,3,8,9] ++ [1,4,5,7]
=   1: [2,3,8,9] ++ [4,5,7]
=   1: 2: [3,8,9] ++ [4,5,7]
=   1: 2: 3: [8,9] ++ [4,5,7]
=   1: 2: 3: 4: [8,9] ++ [5,7]
=   1: 2: 3: 4: 5: [8,9] ++ [7]
=   1: 2: 3: 4: 5: 7: [8,9] ++ []
=   1: 2: 3: 4: 5: 7: [8,9]
=   1: 2: 3: 4: 5: [7,8,9]
=   1: 2: 3: 4: [5,7,8,9]
=   1: 2: 3: [4,5,7,8,9]
=   1: 2: [3,4,5,7,8,9]
=   1: [2,3,4,5,7,8,9]
=   [1,2,3,4,5,7,8,9]
```

3. Implement mergesort using function `merge`.

   Algorithm: MergeSort (a)

   Input: a is a list of comparable items.
   Output: A sorted list of items in a.

   if *len(a) = 0 or len(a) = 1* **then** **return** a
   m ← $\lfloor$ len(a)/2 $\rfloor$
   return Merge (MergeSort a[:m]), (MergeSort a[m:])