CS112: Lecture 5

Lecturer: Hongwei Xi

January 28, 2010
A recurrence equation and its solution

Assume that we have the following recurrence equation for some integer constants $M, N \geq 2$:

$$T(n) = M \cdot T(n/N) + O(1) \text{ if } n > 1.$$  

The solution to this recurrence equation is

$$T(n) = O(n^\alpha) \text{ where } \alpha = \log_N(M).$$
Some examples

For $M = 2$ and $N = 2$, we have

$$T_1(n) = 2 \times T_1(n/2) + O(1) \quad \text{for } n > 1$$

and $T_1(n)$ is $O(n)$.

For $M = 4$ and $N = 2$, we have

$$T_2(n) = 4 \times T_2(n/2) + O(1) \quad \text{for } n > 1$$

and $T_2(n)$ is $O(n^2)$.

For $M = 8$ and $N = 2$, we have

$$T_3(n) = 8 \times T_3(n/2) + O(1) \quad \text{for } n > 1$$

and $T_3(n)$ is $O(n^3)$. 
private static
void merge_ (Node res, Node xs1, Node xs2) {
    if (xs1 == null) {
        res.next = xs2; return ;
    }
    if (xs2 == null) {
        res.next = xs1; return ;
    }
    if (xs1.item <= xs2.item) {
        res.next = xs1; merge_ (xs1, xs1.next, xs2);
    } else {
        res.next = xs2; merge_ (xs2, xs1, xs2.next);
    }
    return ;
} // end of [merge_]
public static
Node merge (Node xs1, Node xs2) {
    if (xs1 == null) return xs2;
    if (xs2 == null) return xs1;
    if (xs1.item <= xs2.item) {
        merge_ (xs1, xs1.next, xs2) ; return xs1;
    } else {
        merge_ (xs2, xs1, xs2.next) ; return xs2;
    } // end of [if]
} // end of [merge]
public static Node mergesort (Node xs) {
    int n = length (xs);
    if (n <= 1) return xs;
    int n2 = (n-1)/2;
    Node xs1 = xs;
    for (int i = 0; i < n2; i += 1) xs = xs.next;
    Node xs2 = xs.next;
    xs.next = null;
    xs1 = mergesort (xs1); xs2 = mergesort (xs2);
    return merge (xs1, xs2);
} // end of [mergesort]
private static
Node mergesort2_ (Node xs, int n) { // length(xs) = n
  if (n <= 1) return xs ;
  int n2 = (n-1)/2;
  Node xs1 = xs;
  for (int i = 0; i < n2; i += 1) {
    xs = xs.next;
  }
  Node xs2 = xs.next;
  xs.next = null;
  xs1 = mergesort2_ (xs1, n2+1);
  xs2 = mergesort2_ (xs2, n-n2-1);
  return merge (xs1, xs2);
} // end of [mergesort2_]
public static Node mergesort2 (Node xs) {
    return mergesort2_ (xs, length (xs)) ;
} // end of [mergesort2]
What is a stack?

A stack is just a stack. The elements in a stack are inserted and removed according to the LIFO discipline: first-in-last-out.
A list-based stack implementation

class intStack {
    static class StackEmptyException extends Exception { };
    intStack() { } // constructor
    private Node theRoot = null;
    public boolean isEmpty () { return (theRoot == null) ; }
    public void push (int item) {
        theRoot = new Node (item, theRoot) ; return ;
    }
    public int pop () throws StackEmptyException {
        if (theRoot == null) throw new StackEmptyException () ;
        int item = theRoot.item ;
        theRoot = theRoot.next ; return item ;
    }
} /* end of [intStack] */