CS109A Notes for Lecture 2/5/96

Programs with Function Calls

- Establish a size measure n for each function.
- Let $T_f(n)$ be the running time of function f.
- When evaluating a simple or compound statement, include the running times of any function calls.
- Do not put big-oh around the functions' running times, because we must keep track carefully of how many calls there are.

Nonrecursive Functions

If no recursion, we can order the function evaluations so each function calls only previously evaluated functions.

- Example pp. 128–130, FCS.

Recursions

Define running time $T_f(n)$ recursively in terms of itself (with arguments smaller than n).

- Solve the resulting *recurrence relation* by one of several "tricks": repeated expansion or "guess-and-check."
- When combining times of function calls like $T_f(n)$ with big-oh expressions, remember that each call takes at least $O(1)$.
  - Thus, $T_f(n)$ subsumes $O(1)$; e.g., $T_f(n)+O(1)$ can be replaced by $T_f(n)$.
- However, in general, big-oh and $T_f(n)$ expressions cannot be combined.

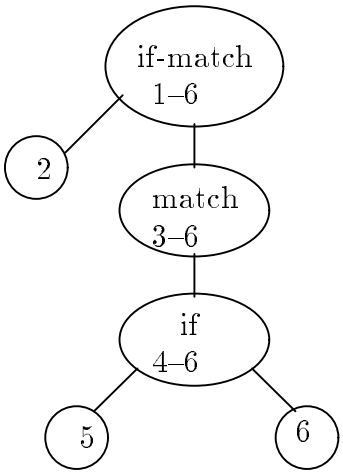
Example: Insertion-sort in ML:

```
(1) fun insert(x:int, nil) =
(2)         [x]
(3) |   insert(x, y::ys) =
(4)         if x>y
(5)         then y::insert(x,ys)
(6)         else x::y::ys;
```

```
(7)  fun isort(nil) =
(8)          nil
(9)  |    isort(x::xs) =
(10)     insert(x, isort(xs));
```

- Hint for ML function analysis: A pattern is like nested if's:

  if(first pattern matches)
  evaluate first expression;
  else if(second pattern matches)
  evaluate second expression;
  . . .



- Let $T_{ins}(n)$ and $T_{isort}(n)$ be the two functions' running times.
  - n is the length of the input list in each case.

Analysis of `insert`

- Lines 1, 2, 3, 4, 6 take $O(1)$ time.
- Line 5 takes $O(1) + T_{ins}(n-1)$.
  - $O(1)$ for the cons.
  - $T_{ins}(n-1)$ for the recursive call on the tail of the list.
- If-expression 4–6 takes

$$O(1) + \max\big(O(1),\ O(1) + T_{ins}(n-1)\big)$$

  - Simplifies to $O(1) + T_{ins}(n-1)$.

- Pattern match of 3–6 takes $O(1)+T_{ins}(n-1)$.
- Pattern match and test of lines 1–6 takes $O(1) + \max\big(O(1),\ O(1) + T_{ins}(n-1)\big)$.
  - Simplifies to $O(1) + T_{ins}(n-1)$.

The Recurrence Relation for `insert`

Basis: If $n = 0$, only lines 1 and 2 execute.

$$T_{ins}(0) = O(1)$$

Induction: If $n > 0$:

$$T_{ins}(n) = O(1) + T_{ins}(n-1)$$

- Replace $O(1)$'s by concrete constants:

$$T_{ins}(0) = a$$
$$T_{ins}(n) = b + T_{ins}(n-1)$$

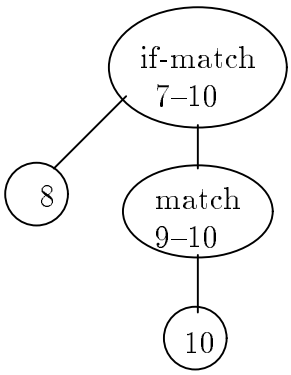
- Repeated expansion gives:

$$T_{ins}(n) = b + \big(b + T_{ins}(n-2)\big)$$
$$T_{ins}(n) = 2b + \big(b + T_{ins}(n-3)\big)$$
$$\vdots$$
$$T_{ins}(n) = (n-1)b + \big(b + T_{ins}(0)\big)$$
$$T_{ins}(n) = nb + a$$

- Restore the big-oh instead of unknown constants a and b: $T_{ins}(n) = O(n)$.

Analysis of `isort`

```
(7)  fun isort(nil) =
(8)          nil
(9)  |   isort(x::xs) =
(10)     insert(x, isort(xs));
```



- Lines 7, 8, 9 are $O(1)$.

- For line 10:
  - `isort(x,xs)` takes time $T_{isort}(n-1)$.
  - Then applying `insert` to a list of length n takes time $O(n)$ by previous example.
  - Thus, line 10 takes $O(n) + T_{isort}(n)$.
- Match of 9–10 and if-match of 7–10 also take $O(n) + T_{isort}(n)$.

Basis: If $n = 0$, only lines 7–8 execute.

$$T_{isort}(0) = O(1)$$

Induction: If $n > 0$:

$$T_{isort}(n) = O(n) + T_{isort}(n-1)$$

- Replace $O(1)$ by a and $O(n)$ by bn.

$$T_{isort}(0) = a$$
$$T_{isort}(n) = bn + T_{isort}(n-1)$$

- Repeated expansion gives:

$$T_{isort}(n) = bn + \big(b(n-1) + T_{isort}(n-2)\big)$$
$$T_{isort}(n) = b(2n-1) + \big(b(n-2) + T_{isort}(n-3)\big)$$
$$\vdots$$
$$T_{isort}(n) = b\textstyle\sum_{i=0}^{n-1}(n-i) + T_{isort}(0)$$
$$T_{isort}(n) = n(n-1)b/2 + a$$

- Restore the big-oh instead of unknown constants a and b: $T_{isort}(n) = O(n^2)$.

General rule: $T(0) = a$, $T(n) = f(n) + T(n-1)$ yields $T(n) = nf(n)$.

- Warning: $f(n)$ must be at most a polynomial and increasing in n.