COMP 250

Lecture 33

recurrences 1

March. 30, 2022

What’s left to do ?

- Lecture 33, 34 : Recurrences
- Lecture 35, 36, 37: Asymptotic Complexity

Let $t(n)$ be the time or number of instructions to execute an algorithm.

We have discussed how to determine $t(n)$ when our algorithms only have loops.

Let’s briefly review a few examples...

Grade School Addition

$carry = 0$ $\qquad c_1$

for $i = 0$ to $N - 1$ **do**

$$\left.\begin{array}{lcl} sum & \leftarrow & a[i] + b[i] + carry \\ r[i] & \leftarrow & sum \ \% \ 10 \\ carry & \leftarrow & sum \ / \ 10 \end{array}\right\} \quad c_2 N$$

end for

$r[N] \leftarrow carry$ $\qquad c_3$

The number of steps is $t(N) = (c_1 + c_3) + c_2 * N$.

When we analyze algorithms with $O(\ )$, we ignore the constants.

Grade School Multiplication

Step 1	**for** $j = 0$ to $N-1$ **do**	
	$\quad carry \leftarrow 0$	N
	$\quad$ **for** $i = 0$ to $N-1$ **do**	
	$\quad\quad prod \leftarrow (a[i] * b[j] + carry)$	
	$\quad\quad table[j][i+j] \leftarrow prod\%10$	N^2
	$\quad\quad carry \leftarrow prod/10$	
	$\quad$ **end for**	
	$\quad table[j][N+j] \leftarrow carry$	N
	end for	
Step 2	$carry \leftarrow 0$	1
	for $i = 0$ to $2 * N - 1$ **do**	
	$\quad sum \leftarrow carry$	N
	$\quad$ **for** $j = 0$ to $N-1$ **do**	
	$\quad\quad sum \leftarrow sum + table[j][i]$	N^2
	$\quad$ **end for**	
	$\quad r[i] \leftarrow sum\%10$	N
	$\quad carry \leftarrow sum/10$	
	end for	

Similarly, we could include constant factors for each of these terms.

The number of steps is $t(N) = c_0 + c_1 N + c_2 * N^2$.

Selection Sort

for $i = 0$ to $N - 2${

  index = i
  minValue = list[i] $\sim N$

  for k = $i + 1$ to $N - 1$ { // nested loop

    if (list[k] < minValue){
      minValue = list[k] $\sim N^2/2$
      index = k
    }
  }
  if (index != i)
    list.swap(i, index) $\sim N$
}

The number of steps is $t(N) = c_0 + c_1 N + c_2 * N^2$.

Let $t(n)$ be the time or number of instructions to execute an algorithm.

We have discussed how to determine $t(n)$ when our algorithms only have loops.

Q: How do we determine $t(n)$ for **recursive** algorithms ?

A: We use *recurrence relations*.

Recurrence Relation

A recurrence relation is an equation that defines a sequence of numbers whose n-th term depends on previous terms.

e.g. Fibonacci $F(n) = F(n-1) + F(n-2)$

We will consider recurrence relations for time complexity $t(n)$

e.g. the number of steps to execute a recursive algorithm as a function of the *input size* n.

The recurrence expresses $t(n)$ in terms of a smaller input size.

Example 1 :   Suppose a list has n elements.

What is $t(n)$ for reversing the list as follows ?

```
reverse( list ){
    if list.size > 1 {
        firstElement  =  list.removeFirst( )
        reverse( list )
        list.addLast( firstElement )
    }
}
```

Example 1 : Suppose a list has n elements.

What is $t(n)$ for reversing the list as follows ?

Constant time to do the base case check.

```
reverse( list ){
    if list.size > 1 {
        firstElement  =  list.removeFirst( )
        reverse( list )
        list.addLast( firstElement )
    }
}
```

If a linked list is used, then it is constant time for both removeFirst() and addLast(). But if an arraylist were used, then removeFirst() would not be constant time.

$$t(n) = c + t(n-1)$$

Solving a recurrence using "back substitution"

$$t(n) = c + t(n-1)$$

Solving a recurrence using back substitution

$$
\begin{aligned}
t(n) &= c + t(n-1) \\
&= c + c + t(n-2)
\end{aligned}
$$

Solving a recurrence using back substitution

$$
\begin{aligned}
t(n) &= c + t(n-1) \\
&= c + c + t(n-2) \\
&= c + c + c + t(n-3)
\end{aligned}
$$

Solving a recurrence using back substitution

$$
\begin{aligned}
t(n) &= c + t(n-1) \\
&= c + c + t(n-2) \\
&= c + c + c + t(n-3) \\
&= \ldots \\
&= c\ (n-1) + t(1)
\end{aligned}
$$

e.g. base case is $n = 1$
when reversing a list

Example 2 : Sorting a list

```
sort( list ) {
    if list.size == 1
        return list
    else{
        minElement = list.removeMin()
        sort(list)
        return list.addFirst( minElement )
    }
}
```

What is the recurrence relation?

Example 2 : Sorting a list

```
sort( list ) {
    if list.size == 1
        return list
    else{
        minElement = list.removeMin()
        sort(list)
        return list.addFirst( minElement )
    }
}
```

Base case $n = 1$.
(But we do this test everytime.)

Time proportional to n

Depends on list data structure (worst case is proportional to n)

$$t(n) = \boxed{c_1} + \boxed{c_2\, n} + \boxed{t(n-1)}$$

Let's solve the slightly simpler recurrence, by dropping the constant term.

$$t(n) = c\,n + t(n-1)$$

Let’s solve the slightly simpler recurrence.

$$\begin{aligned} t(n) &= c\,n + t(n-1) \\ &= c\,n + c\cdot(n-1) + t(n-2) \end{aligned}$$

Let’s solve the slightly simpler recurrence. (We are sorting a list. Base case is $n = 1$.)

$$
\begin{aligned}
t(n) &= c\,n + t(n-1) & k = 0 \\
&= c\,n + c\cdot(n-1) + t(n-2) & k = 1 \\
&= \cdots \\
&= c\,\{\, n + (n-1) + (n-2) \;+ \cdots + (n-k)\} + t(n-k-1)
\end{aligned}
$$

Let’s go all the way to the end, and use a cleaner base case $t(0)$ or $n - k = 1$.

Let’s solve the slightly simpler recurrence, **and cleaner base case $(\boldsymbol{n} = \mathbf{0})$.**

$$
\begin{aligned}
t(n) &= c\,n + t(n-1) \\
&= c\,n + c \cdot (n-1) + t(n-2) \\
&= \ldots \\
&= c\,\{\, n + (n-1) + (n-2) \;+ \cdots + (n-k)\} + t(n-k-1) \\
&= c\,\{\, n + (n-1) + (n-2) \;+ \cdots + 2 + 1\} + t(0)
\end{aligned}
$$

Let's solve the slightly simpler recurrence, **and cleaner base case** $(n = 0)$.

$$
\begin{aligned}
t(n) &= c\,n + t(n-1) \\
&= c\,n + c \cdot (n-1) + t(n-2) \\
&= \dots \\
&= c\,\{\, n + (n-1) + (n-2) \;+ \cdots + (n-k)\} + t(n-k-1) \\
&= c\,\{\, n + (n-1) + (n-2) \;+ \cdots + 2 + 1\} + t(0) \\
&= \frac{cn(n+1)}{2} + t(0)
\end{aligned}
$$

Example 3: Tower of Hanoi

```
tower(n, start, finish, other){      // base case is n=1, so t(1) = 1
    if  n == 1
          move from start to finish
    else {
          tower( n-1, start, other,  finish)
          move from start to finish
          tower( n-1, other, finish, start)
     }
}
```

Q: How many moves are needed for a tower with n disks ?

$$t(n) = 1 + 2\,t(n-1)$$

What do you think the solution will be ?

$$t(n) \sim \quad n^2 \,? \quad n^3 \;? \qquad 2^n \;?$$

Tower of Hanoi recurrence

$$t(n) \quad = \quad 1 + 2\ t(n-1)$$

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= 1 + 2\ t(n-1) \\
&= 1 + 2(1 + 2\ t(n-2))
\end{aligned}
$$

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= 1 + 2\ t(n-1) \\
&= 1 + 2(1 + 2\ t(n-2)) \\
&= (1+2) + 4\ t(n-2)
\end{aligned}
$$

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= 1 + 2\ t(n-1) \\
&= 1 + 2(1 + 2\ t(n-2)) \\
&= (1+2) + 4\ t(n-2) \\
&= (1+2) + 4\ (1 + 2\ t(n-3))
\end{aligned}
$$

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= 1 + 2\ t(n-1) \\
&= 1 + 2(1 + 2\ t(n-2)) \\
&= (1+2) + 4\ t(n-2) \\
&= (1+2) + 4\ (1 + 2\ t(n-3)) \\
&= (1+2+4)\ +\ 8\ t(n-3)
\end{aligned}
$$

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= 1 + 2\ t(n-1) && k = 1 \\
&= 1 + 2(1 + 2\ t(n-2)) \\
&= (1+2) + 4\ t(n-2) && k = 2 \\
&= (1+2) + 4\ (1 + 2\ t(n-3)) \\
&= (1+2+4)\ +\ 8\ t(n-3) && k = 3 \\
&= \ldots \\
&= (1+2+4+8+\cdots+2^{k-1}) + 2^k\ \ t(n-k)
\end{aligned}
$$

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= 1 + 2\ t(n-1) \\
&= 1 + 2(1 + 2\ t(n-2)) \\
&= (1+2) + 4\ t(n-2) \\
&= (1+2) + 4\ (1 + 2\ t(n-3)) \\
&= (1+2+4)\ +\ 8\ t(n-3) \\
&= \ldots \\
&= (1+2+4+8+\cdots+2^{k-1}) + 2^{k}\ \ t(n-k) \\
&= (1+2+4+8+\cdots+2^{n-2}) + 2^{n-1}\ t(1)
\end{aligned}
$$

$k = n - 1$

verify

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= 1 + 2\ t(n-1) \\
&= 1 + 2(1 + 2\ t(n-2)) \\
&= (1+2) + 4\ t(n-2) \\
&= (1+2) + 4\ (1 + 2\ t(n-3)) \\
&= (1+2+4)\ +\ 8\ t(n-3) \\
&= \ldots \\
&= (1+2+4+8+\cdots+2^{k-1}) + 2^k\ \ t(n-k) \\
&= (1+2+4+8+\cdots+2^{n-2}) + 2^{n-1}\ t(1) \\
&= (2^{n-1}-1) + 2^{n-1}\ t(1)
\end{aligned}
$$

Tower of Hanoi recurrence

$$
\begin{aligned}
t(n) &= 1 + 2\ t(n-1) \\
&= 1 + 2(1 + 2\ t(n-2)) \\
&= (1+2) + 4\ t(n-2) \\
&= (1+2) + 4\ (1 + 2\ t(n-3)) \\
&= (1+2+4)\ +\ 8\ t(n-3) \\
&= \ldots \\
&= (1+2+4+8+\cdots+2^{k-1}) + 2^k\ \ t(n-k) \\
&= (1+2+4+8+\cdots+2^{n-2}) + 2^{n-1}\ t(1) \\
&= (2^{n-1}-1) + 2^{n-1}\ t(1) \\
&= 2^n - 1,\ \text{since}\ \ t(1) = 1 \longleftarrow \text{Base case for Tower of Hanoi}
\end{aligned}
$$

You should know by now….

$$1 + 2 + 3 + \ \dots + \ \ k \ = \ ?$$

$$1 + 2 + 4 + 8 + \ \dots + \ \ 2^k \ = \ ?$$

$$1 + x + x^2 + x^3 + \ \dots + x^k = \ ?$$

Example 4:   Binary Search

```
binarySearch( list, value, low, high ){        //    n = high − low + 1
    if  low <= high {
        mid = (low + high) / 2
        if value == list[mid]
            return value
        else   if value < list[mid]
            return  binarySearch(list, value,  low,     mid - 1 )
        else
            return  binarySearch(list, value,  mid+1,  high)
        }
    else
        return   -1
}
```

$$t(n) = c + t(\frac{n}{2})$$

The list sizes might not be be exactly n/2, but will be at most off by 1.

Example 4: Binary Search

```
binarySearch( list, value, low, high ){        //     n = high − low + 1
    if  low <= high {
        mid = (low + high) / 2
        if value == list[mid]
            return value                    ←      Base case :  found value
        else   if value < list[mid]
            return  binarySearch(list, value,  low,     mid - 1 )
        else
            return  binarySearch(list, value,  mid+1,  high)
        }
    else
        return   -1                   ←         Base case : high < low,     t(1) = 1
}
```

$$t(n) = c + t\left(\frac{n}{2}\right), \qquad n > 1$$

Binary search recurrence

$$t(n) = c + t(\frac{n}{2})$$

Note this doesn't quite capture the situation, since mid is excluded. But close enough!

Binary search recurrence

$$
\begin{aligned}
t(n) &= c + t(\frac{n}{2}) \\
&= c + c + t(\frac{n}{4})
\end{aligned}
$$

Binary search recurrence

$$
\begin{aligned}
t(n) &= c + t(\frac{n}{2}) \\
&= c + c + t(\frac{n}{4}) \\
&= c + c + \cdots + t(\frac{n}{2^k})
\end{aligned}
$$

Binary search recurrence

$$
\begin{aligned}
t(n) &= c + t(\frac{n}{2}) \\
&= c + c + t(\frac{n}{4}) \\
&= c + c + \cdots + t(\frac{n}{2^k}) \\
&= c + c + \cdots + c + t(\frac{n}{n})
\end{aligned}
$$

For the purpose of solving the recurrence, we assume that n is a power of 2.

$n = 2^k$

Binary search recurrence

$$
\begin{aligned}
t(n) &= c + t(\frac{n}{2}) \\
&= c + c + t(\frac{n}{4}) \\
&= c + c + \cdots + t(\frac{n}{2^k}) \\
&= c + c + \cdots + c + t(\frac{n}{n}) \\
&= c \log_2 n + t(1)
\end{aligned}
$$

For the purpose of solving the recurrence, we assume that n is a power of 2.

$n = 2^k$

Base case

Today's Recurrences

$$t(n) = c + t(n-1)$$

$$t(n) = c\,n + t(n-1)$$

$$t(n) = c + 2\,t(n-1)$$

$$t(n) = c + t\left(\frac{n}{2}\right)$$

Coming up…

Lectures

Fri, April 1

more recurrences

Mon, Wed, Fri : April 4, 6, 8

big O, big Theta, big Omega

best and worst cases

Assessments

Quiz 5 is in Mon. April 4

Practice quizzes are available for Quiz 5 and for lectures after that (33-37)

Assignment 4 due Wed. April 6.