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

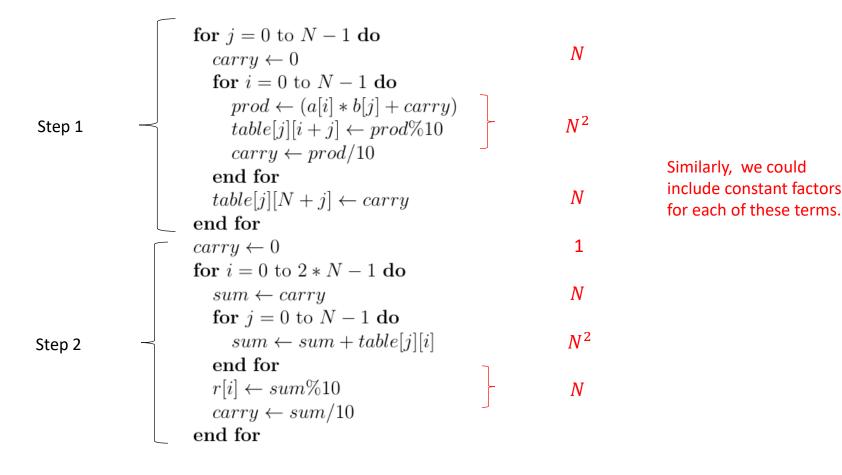
$$sum \leftarrow a[i] + b[i] + carry$$

 $r[i] \leftarrow sum \% 10$
 $carry \leftarrow sum / 10$
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



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
                                            \sim N
   minValue = list[i]
   for k = i + 1 to N - 1 { // nested loop
      if ( list[k] < minValue ){</pre>
         minValue = list[k]
                                        \sim N^2/2
         index = k
       }
   }
   if (index != i)
                                          ~N
        list.swap(i, index )
}
```

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.

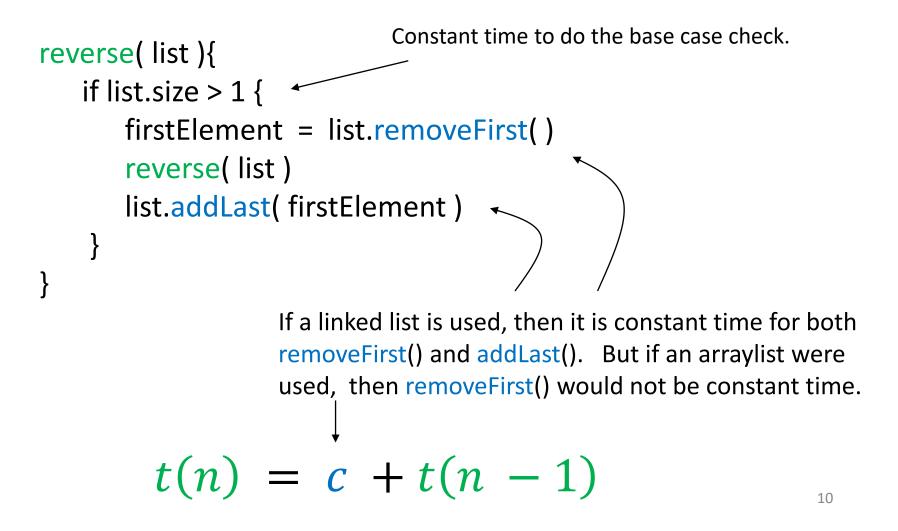
Example 1 : Suppose a list has *n* elements.

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 ?



Solving a recurrence using "back substitution"

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

Solving a recurrence using back substitution

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



Solving a recurrence using back substitution

$$t(n) = c + t(n-1)$$
$$= c + c + t(n-2)$$
$$= c + c + c + t(n-3)$$

Solving a recurrence using back substitution

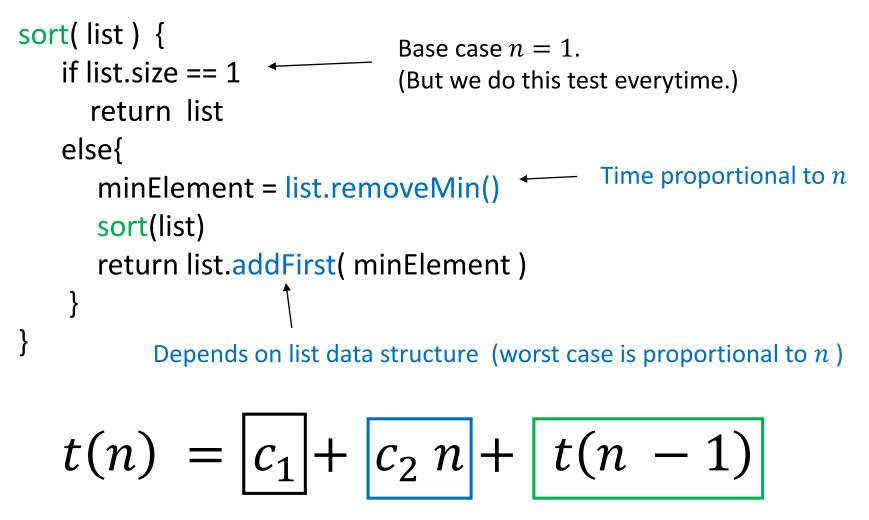
t(n) = c + t(n-1)= c + c + t(n - 2)= c + c + c + t(n - 3)= c (n-1) + t(1)e.g. base case is n = 1when reversing a list

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



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.

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

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



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

$$t(n) = c n + t(n-1)$$
 $k = 0$

$$= c n + c \cdot (n-1) + t(n-2) \qquad k = 1$$

$$= c \{ n + (n - 1) + (n - 2) + \dots + (n - k) \} + t(n - k - 1)$$

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 (n = 0).

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

= $c n + c \cdot (n - 1) + t(n - 2)$
= ...
= $c \{ n + (n - 1) + (n - 2) + \dots + (n - k) \} + t(n - k - 1)$
= $c \{ n + (n - 1) + (n - 2) + \dots + 2 + 1 \} + t(0)$

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

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

. . .

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

$$= c \{ n + (n-1) + (n-2) + \dots + (n-k) \} + t(n-k-1)$$

$$= c \{ n + (n-1) + (n-2) + \dots + 2 + 1 \} + t(0)$$

$$= \frac{cn(n+1)}{2} + t(0)$$

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 ?

$$t(n) \sim n^2$$
? n^3 ? 2^n ?

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

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

= 1 + 2(1 + 2 t(n-2))

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

= 1 + 2(1 + 2 t(n - 2))
= (1 + 2) + 4 t(n - 2)

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

= 1 + 2(1 + 2 t(n - 2))
= (1 + 2) + 4 t(n - 2)
= (1 + 2) + 4 (1 + 2 t(n - 3))

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

= 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)

$$t(n) = 1 + 2 t(n-1) \qquad k = 1$$

= 1+2(1+2 t(n-2))
= (1+2) + 4 t(n-2) \qquad k = 2
= (1+2) + 4 (1+2 t(n-3))
= (1+2+4) + 8 t(n-3) \qquad k = 3
= ...
= (1+2+4+8+...+2^{k-1}) + 2^k t(n-k)

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

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

$$= (1 + 2) + 4 t(n - 2)$$

$$= (1 + 2) + 4 (1 + 2 t(n - 3))$$

$$= (1 + 2 + 4) + 8 t(n - 3)$$

$$= ...$$

$$= (1 + 2 + 4 + 8 + \dots + 2^{k-1}) + 2^{k} t(n - k)$$

$$= (1 + 2 + 4 + 8 + \dots + 2^{n-2}) + 2^{n-1} t(1)$$

verify

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

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

$$= (1 + 2) + 4 t(n - 2)$$

$$= (1 + 2) + 4 (1 + 2 t(n - 3))$$

$$= (1 + 2 + 4) + 8 t(n - 3)$$

$$= ...$$

$$= (1 + 2 + 4 + 8 + \dots + 2^{k-1}) + 2^{k} t(n - k)$$

$$= (1 + 2 + 4 + 8 + \dots + 2^{n-2}) + 2^{n-1} t(1)$$

$$= (2^{n-1} - 1) + 2^{n-1} t(1)$$

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

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]
                            Base case : found value
        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
                                  Base case : high < low, t(1) = 1
}
  t(n) = c + t\left(\frac{n}{2}\right),
                                                  n > 1
```

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

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

$$t(n) = c + t(\frac{n}{2})$$
$$= c + c + t(\frac{n}{4})$$



$$t(n) = c + t(\frac{n}{2})$$
$$= c + c + t(\frac{n}{4})$$
$$= c + c + \dots + t(\frac{n}{2^k})$$

$$\begin{split} t(n) &= c + t(\frac{n}{2}) & \text{For the purpose of solving the recurrence, we assume that n is a power of 2.} \\ &= c + c + t(\frac{n}{4}) & \\ &= c + c + \dots + t(\frac{n}{2^k}) & \\ &= c + c + \dots + c + t(\frac{n}{n}) & n = 2^k \end{split}$$

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

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

$$= c + c + t(\frac{n}{4})$$

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

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

$$= c \log_2 n + t(1)$$

Base case

Today's Recurrences

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

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

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

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

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.