Recurrence Relations

Class 40

Definitions

- you are very familiar with function definitions in math
- a function is defined with an algebraic rule

$$f(x) = x^2 - 3x + 2$$

- it can be translated directly into a C++ function

```
double f(double x)
{
  return x * x - 3 * x + 2;
}
```

Sequences

- there is another type of definition
- commonly used to define sequences of values
- the Fibonacci sequence can be listed as $\{1, 1, 2, 3, 5, \ldots\}$
- and it can also be defined by a rule

$$f(n) = f(n-1) + f(n-2) \text{ given } f(0) = f(1) = 1$$

- this type of rule is called a recurrence relation
- with initial conditions

Recurrence Relations

- a recurrence relation is an equation or inequality
- defines an arbitrary element in a sequence in terms of one or more of its predecessors

Recurrence Relations

- a recurrence relation is an equation or inequality
- defines an arbitrary element in a sequence in terms of one or more of its predecessors

- a recursive algorithm implements a recurrence relation
- a recurrence relation describes a recursive algorithm

Recurrence to Recursion

- recurrence relations translate into code
- the initial conditions turn into base cases
- the code has recursive calls

```
unsigned fib(unsigned n)
{
  if (n == 0 || n == 1)
  {
    return 1;
  }
  return fib(n - 1) + fib(n - 2);
}
```

Solving Recurrence Relations

- to solve a recurrence relation means to give a formulation for an arbitrary element in a sequence in terms that does not use any other elements in the sequence
- the solution is a closed form
- there are many techniques for solving recurrence relations
- we will only look at a couple

Solving by Substitution

Let

$$T(n) = T(n-1) + n \text{ given } T(0) = 0$$

- off to the side, replace every occurrence of n with $n-1$
- we can do this because n is arbitrary
- this substitution gives us

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

Solving by Substitution

Let

$$T(n) = T(n-1) + n \text{ given } T(0) = 0$$

- off to the side, replace every occurrence of n with $n-1$
- we can do this because n is arbitrary
- this substitution gives us

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

- now substitute this expression for $T(n-1)$ back into the original formulation, to give

Solving by Substitution

Let

$$T(n) = T(n-1) + n \text{ given } T(0) = 0$$

- off to the side, replace every occurrence of n with $n-1$
- we can do this because n is arbitrary
- this substitution gives us

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

- now substitute this expression for $T(n-1)$ back into the original formulation, to give

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

Solving by Substitution

- using the original formulation, off to the side substitute every occurrence of n by $n-2$ to get

$$T(n-2) = T(n-3) + (n-2)$$

- and use this expression for $T(n-2)$ in the last expression of the previous slide

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

Solving by Substitution

- continuing the series, we have

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

- how long can this process go on?

Solving by Substitution

- the series ends at the initial condition (base case) $T(0) = 0$

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&= T(n-2) + (n-1) + n \\
&= T(n-3) + (n-2) + (n-1) + n \\
&\vdots \\
&= T(n-(n-1)) + (n-(n-2)) + (n-(n-3)) + \cdots \\
&\qquad + (n-1) + n \\
&= T(n-n) + (n-(n-1)) + (n-(n-2)) + (n-(n-3)) \\
&\qquad + \cdots + (n-1) + n \\
&= 0 + 1 + 2 + \cdots + n \\
&= \frac{n(n+1)}{2}
\end{aligned}
$$

Analysis

- thus we have the closed form

$$\begin{aligned} T(n) &= T(n-1) + n \text{ given } T(0) = 0 \\ &= \frac{n(n+1)}{2} \end{aligned}$$

- the solution of a recurrence relation is identical to the analysis of its matching recursive algorithm
- we analyze recursive algorithms by
  - writing the recurrence relation for the algorithm
  - solving that recurrence relation

Solving by Cancellation

- a second technique for solving recurrence relations is cancellation
- I find this much more confusing that substitution, and no more enlightening
- if you like it, feel free to use it

- I have used:

$$T(n) = T(n-1) + n \text{ given } T(0) = 0$$

your author instead writes:

$$\begin{aligned} r_0 &= 0, \\ r_n &= r_{n-1} + n. \end{aligned}$$

Analyzing Recursive Functions

- substitution only works when there is a single recursive term, and it differs in position by exactly one $n \to n - 1$
- substitution, for example cannot be used to find a closed form for the nth Fibonacci number
- many recurrence relations in computer science can be solved by substitution
- but many are of a different form

Binary Search

- binary search is a classic recursive algorithm
- a recursive algorithm consists of
    1. one or more checks for base case(s)
    2. some amount of local work
    3. one or more recursive calls

Binary Search

2	3	5	11	17	23	29

1. if the range of elements is **empty**, return not-found sentinel

Binary Search

2	3	5	11	17	23	29

1. if the range of elements is **empty**, return not-found sentinel
2. divide the range of elements to search into 3:
   2.1 the very middle element

Binary Search

2	3	5	11	17	23	29

1. if the range of elements is **empty**, return not-found sentinel
2. divide the range of elements to search into 3:
    2.1 the very middle element
    2.2 the elements to the **left** of middle

Binary Search

2	3	5	11	17	23	29

1. if the range of elements is **empty**, return not-found sentinel
2. divide the range of elements to search into 3:
   2.1 the very middle element
   2.2 the elements to the **left** of middle
   2.3 the elements to the right of middle

Binary Search

2	3	5	11	17	23	29

1. if the range of elements is **empty**, return not-found sentinel
2. divide the range of elements to search into 3:
   2.1 the very middle element
   2.2 the elements to the **left** of middle
   2.3 the elements to the right of middle
3. if the searched-for value is the middle element, you've found it and you're done

Binary Search

2	3	5	11	17	23	29

1. if the range of elements is empty, return not-found sentinel
2. divide the range of elements to search into 3:
   2.1 the very middle element
   2.2 the elements to the left of middle
   2.3 the elements to the right of middle
3. if the searched-for value is the middle element, you've found it and you're done
4. else if the searched-for value is smaller than the middle element, repeat step 1 on the left half

Binary Search

2	3	5	11	17	23	29

1. if the range of elements is **empty**, return not-found sentinel
2. divide the range of elements to search into 3:
   2.1 the very middle element
   2.2 the elements to the **left** of middle
   2.3 the elements to the right of middle
3. if the searched-for value is the middle element, you've found it and you're done
4. else if the searched-for value is **smaller** than the middle element, repeat step 1 on the left half
5. else repeat step 1 on the right half

Binary Search

look at code

Binary Search Analysis

base case determination

- line 5: 1 operation
- line 6: 3 operations

local work

- line 8: 3 operations
- lines 9 and 13: 1 operation
- line 11 or 15: 1 operation

total: 9 operations

Binary Search Analysis

- how many recursive calls?

- how big is the input for the recursive call?

- can the algorithm end early?

Binary Search Analysis

- how many recursive calls?
  - either line 9 or line 13, but never both
  - therefore one recursive call
- how big is the input for the recursive call?
- can the algorithm end early?

Binary Search Analysis

- how many recursive calls?
  - either line 9 or line 13, but never both
  - therefore one recursive call
- how big is the input for the recursive call?
  - the size of the range is half the size of the original
- can the algorithm end early?

Binary Search Analysis

- how many recursive calls?
  - either line 9 or line 13, but never both
  - therefore one recursive call
- how big is the input for the recursive call?
  - the size of the range is half the size of the original
- can the algorithm end early?
  - yes, because of line 19 `return mid;`

Binary Search Analysis

- running this algorithm thus involves
  - $\leq$ because the whole process might end early
  - 9 operations of local work
  - 1 recursive call on a range of size $\frac{n}{2}$

$$T(n) \leq aT\left(\frac{n}{b}\right) + kn^d$$

Binary Search Analysis

- running this algorithm thus involves
  - $\leq$ because the whole process might end early
  - 9 operations of local work
  - 1 recursive call on a range of size $\frac{n}{2}$

$$T(n) \leq aT\left(\frac{n}{b}\right) + kn^d$$

Binary Search Analysis

- running this algorithm thus involves
  - $\leq$ because the whole process might end early
  - 9 operations of local work
  - 1 recursive call on a range of size $\frac{n}{2}$

$$
\begin{aligned}
T(n) &\leq aT\left(\frac{n}{b}\right) + kn^d \\
&\leq 1T\left(\frac{n}{2}\right) + 9n^0
\end{aligned}
$$

Base Case

- what is the base case for binary search?

Base Case

- what is the base case for binary search?
- the size of the range is 0
- the comparison on line 6 fails

  line 5: 1 operation

  line 6: 3 operations

total: 4 operations

therefore, $T(0) = 4$

- using $T(0)$ instead of $T(1)$, on page 379, your textbook presents (using cancellation) this formula for $T(n)$:

$$T(n) = a^k T(0) + \sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right)$$

- in this case, we have $a = 1$, $b = 2$, $f(n) = 9n^0$, and $T(0) = 4$
- we also know that $2^k = n$ which means $k = \log_2 n$
- this gives

$$\begin{aligned} T(n) &= 4 \cdot 1^k + \sum_{i=0}^{k-1} 1^i 9 \\ &= 4 + 9 \log_2 n \end{aligned}$$

Best Case

- what is the best case?
- we find the searched-for element at the very first spot we check

  line 5: 1 operation
  line 6: 3 operations
  line 8: 3 operations
  lines 9 and 13: 1 operation

- total: 8 operations

Complete Analysis

- putting it all together, for recursive binary search, we have
  - best case: 8 operations
  - worst case: $4 + 9\log_2 n$ opeations