16.1 Analyzing the time complexity of recursive algorithms

This section has been set as optional by your instructor.

Deriving the recurrence relation

The ***time complexity*** of an algorithm is a function T(n) indicating the number of atomic operations performed by the algorithm on an input of size n. For a recursive algorithm, the function T(n) is defined recursively by a recurrence relation and initial values. For example, suppose that the function T(n) indicates the number of atomic operations performed by the factorial algorithm given below on input n.

```
Factorial(n)

If (n = 0), Return(1)

r := Factorial(n-1)

Return( r*n )
```

- Initial value: The initial value for the function T is the number of operations performed during the base case. The base case occurs for the factorial algorithm when n = 0, so the initial value would be the value for T(0).
- Recurrence relation: For $n \geq 1$, the value of T(n) is defined by a recurrence relation that includes the running time for the recursive calls plus the additional operations performed outside the recursive calls.

PARTICIPATION ACTIVITY 16.1.1: Deriving the recurrence relation that defines the time complexity of a recursive algorithm.

Animation captions:

1. T(n) = number of atomic operations performed by the factorial algorithm on input n.
2. If n = 0, two atomic operations are performed: one to test n = 0 and one to return the value of 1. Therefore T(0) = 2.

3. The number of operations performed in the recursive call to Factorial on input n-1 is T(n-1).
4. There are 5 additional operations, so the recurrence relation is T(n) = T(n-1) + 5.

PARTICIPATION ACTIVITY | 16.1.2: Deriving a recurrence relation to determine the asymptotic complexity of a recursive algorithm.

The function PowerRecursive below takes inputs r and n, where r is any real number and n is a non-negative integer. The output is r^n.

```
PowerRecursive(r, n)

If (n = 0), Return (1)

y := PowerRecursive(r, n-1)

Return (r · y)
```

The number of atomic operations performed by PowerRecursive does not depend on the value of r. The function T(n) is the number of atomic operations performed by the PowerRecursive on input n. The questions below derive the recurrence relation for the time complexity of the algorithm PowerRecursive.

1) What is the number of atomic operations performed by the recursive call in PowerRecursive?
   - T(n)
   - T(n-1)
   - T(r-1)

2) How many atomic operations are performed by PowerRecursive outside of the recursive call if the exponent n is greater than 0?
   - 1
   - 5
   - n-1

3) What is the correct recurrence relation for T(n)?
   - $T(n) = T(n - 1)$

- $T(n) = n + 5$
- $T(n) = T(n - 1) + 5$

Overview of analyzing recursive algorithms: A two step process

The animation above shows that the time complexity of the factorial function T(n) is described by the initial value and recurrence relation:

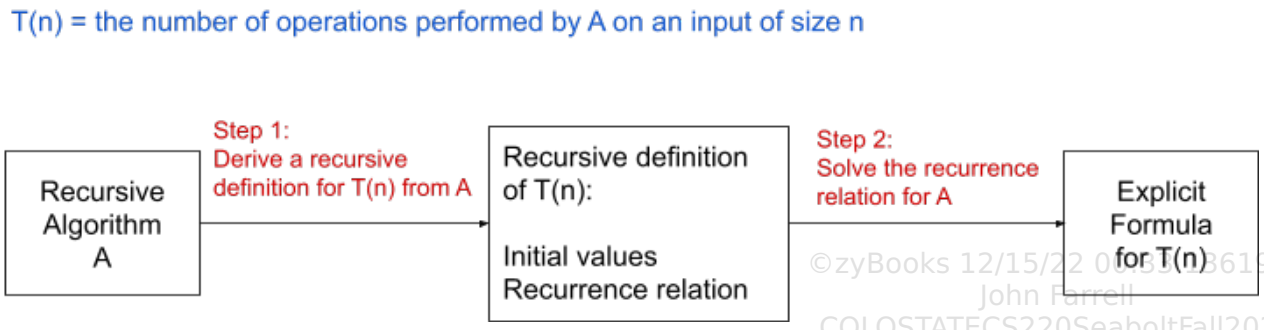
- T(0) = 2
- T(n) = T(n-1) + 5, for $n \geq 1$

The recursive definition for T is not useful in determining the actual number of operations performed on a particular input. For example, the recursive definition does not provide a convenient way to determine T(100), the number of operations performed by the algorithm on an input of size 100.

An explicit formula for T(n), that does not depend on the value of T for smaller inputs, is more useful in understanding the time complexity of the algorithm. It turns out that the explicit formula for the factorial algorithm is T(n) = 5n + 2.

After deriving the recurrence relation for T(n) from a recursive algorithm, the next step is to solve the recurrence relation by finding an explicit formula for T(n). This section focuses on deriving a recurrence relation from a recursive algorithm. Solving recurrence relations is covered elsewhere in this material.

Figure 16.1.1: The two steps in analyzing the time complexity of a recursive algorithm.



PARTICIPATION ACTIVITY 16.1.3: Recursively defined functions and explicit formulas.

Identify whether the definition of the function T is a recursive definition or an explicit

formula.

1) $T(n) = 3n - 2$
   - Recursive definition
   - Explicit formula

2) $T(n) = 3 \cdot T(\lceil n/3 \rceil) + n^2 + 2$, for $n \geq 2$ and $T(1) = 3$
   - Recursive definition
   - Explicit formula

3) $T(n) = 3^n + n^2 + 7$
   - Recursive definition
   - Explicit formula

Simplifying recurrence relations for asymptotic analysis

The exact number of operations performed by an algorithm is less important than the asymptotic complexity of the algorithm, described using Θ notation. For example, it is more important to know that the time complexity of the factorial function is Θ(n), rather than 5n + 2.

Some of the constants in a recurrence relation do not affect the asymptotic growth of the final explicit formula. For example, the recurrences T(n) = T(n-1) + 5n and T(n) = T(n-1) + 7n both result in explicit formulas that are $\Theta(n^2)$. Therefore, both recurrence relations can be simplified as T(n) = T(n-1) + Θ(n).

The initial value for T, such as the value of T(0), is omitted in determining the asymptotic time complexity of a recursive algorithm because the number of operations performed in the base case is almost always Θ(1). The exact constant does not affect that asymptotic complexity of the algorithm.

PARTICIPATION ACTIVITY 16.1.4: Rules for simplifying recurrence relations for asymptotic analysis.

Animation content:

undefined

Animation captions:

1. Additive functions of n can be replaced with Θ notation.

2. For example, any recurrence of the form $T(n) = T(n-1) + \Theta(n^2)$ will have explicit formula $\Theta(n^3)$.
3. Terms such as $T(\lceil n/c \rceil)$ and $T(\lfloor n/c \rfloor)$ can be replaced with $T(n/c)$ for any constant c.
4. Recurrences $T(n) = 2 \cdot T(\lceil n/2 \rceil) + \Theta(n)$ and $T(n) = 2 \cdot T(\lfloor n/2 \rfloor) + \Theta(n)$ can be simplified as $T(n) = 2 \cdot T(n/2) + \Theta(n)$.
5. Constant factors in front of T(·) are important in the asymptotic growth of T(n) and can not be changed.
6. The two recurrences $T(n) = 4 \cdot T(n/2) + \Theta(n)$ and $T(n) = 2 \cdot T(n/2) + \Theta(n)$ result in explicit. formulas with different asymptotic growth.

PARTICIPATION ACTIVITY 16.1.5: Simplifying recurrence relations in determining the asymptotic growth of functions.

Indicate whether one recurrence relation can replace the other in determining the asymptotic growth of T(n).

1) $T(n) = 3 \cdot T(n/2) + 7n$
$T(n) = T(n/2) + \Theta(n)$

- Yes
- No

2) $T(n) = T(\lfloor n/2 \rfloor) + 5n^2 + 10$
$T(n) = T(n/2) + \Theta(n^2)$

- Yes
- No

3) $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$
$T(n) = 2 \cdot T(n/2) + \Theta(n)$

- Yes
- No

4) $T(n) = T(n-1) + 3n^2 + 8n$
$T(n) = T(n-1) + \Theta(n^3)$

- Yes
- No

Deriving a simplified recurrence relation for asymptotic analysis

The specific constants that arise in counting each atomic operation in a recursive algorithm do not

affect the algorithm's asymptotic complexity. The animation below shows how the derivation of a recurrence relation can be simplified if the goal is to determine the algorithm's asymptotic complexity instead of determining the exact number of operations performed.

PARTICIPATION ACTIVITY | 16.1.6: Deriving the recurrence relation that defines the asymptotic time complexity of a recursive algorithm.

Animation captions:

1. $T(n)$ = number of atomic operations performed by the factorial algorithm on input n.
2. The number of operations performed in the recursive call to Factorial on input $n - 1$ is $T(n - 1)$.
3. There are $\Theta(1)$ additional operations, so the recurrence relation is $T(n) = T(n - 1) + \Theta(1)$.

PARTICIPATION ACTIVITY | 16.1.7: Deriving a recurrence relation to determine the asymptotic complexity of a recursive algorithm.

The function FastPowerRecursive below takes inputs r and n, where r is any real number and n is a non-negative integer. The output is r^n.

```
FastPowerRecursive(r, n)

If (n = 0), Return (1)

x := ⌊n/2⌋
y := FastPowerRecursive(r, x)

If (n is even), Return (y · y)
If (n is odd), Return (y · y · r)
```

The number of atomic operations performed by FastPowerRecursive does not depend on the value of r. The function T(n) is the number of atomic operations performed by the FastPowerRecursive on input n. The questions below derive the recurrence relation for the asymptotic complexity of the algorithm FastPowerRecursive.

1) What is the correct $T(\cdot)$ term in the recurrence relation for T(n)?
   - T(n)
   - T(n-1)
   - T(n/2)

2) How many atomic operations are performed by FastPowerRecursive outside of the recursive call? You can assume that the test to determine whether n is even or odd requires Θ(1) operations.

- $\Theta(1)$
- $\Theta(r)$
- $\Theta(n)$

3) What is the correct recurrence relation for T(n)?

- $T(n) = T(n/2)$
- $T(n) = T(n/2) + \Theta(1)$
- $T(n) = 2 \cdot T(n/2) + \Theta(1)$

Additional exercises

EXERCISE 16.1.1: Simplifying recurrence relations for asymptotic analysis.

Simplify each recurrence relation as much as possible. The simplified formula for the function should have the same asymptotic growth as the original recurrence relation.

(a) $T(n) = T(n-1) + 5n^3 + 4n$

(b) $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 7n$

(c) $T(n) = 3 \cdot T(\lfloor n/2 \rfloor) + 14$

(d) $T(n) = 3 \cdot T(\lceil n/3 \rceil) + 4n + 6n\log n$

EXERCISE 16.1.2: Recursively computing sums of cubes, cont.

(a) Give the recurrence relation to describe the asymptotic time complexity of your algorithm to compute the sum of the cubes of the first n positive integers.

EXERCISE 16.1.3: Recursively computing the product of two non-negative integers, cont.

(a) Give the recurrence relation to describe the asymptotic time complexity of your algorithm that recursively computes the product of x and y, where x and y are non-negative integers.

EXERCISE 16.1.4: Recursively computing a number raised to an exponent that is a power of 2, cont.

(a) Give the recurrence relation to describe the asymptotic time complexity of your algorithm to compute r^{2^n}.

EXERCISE 16.1.5: Recurrence relation for RecursiveFibonacci.

(a) The algorithm given below is a recursive algorithm to compute the n^{th} Fibonacci number. Give a recurrence relation to describe the asymptotic complexity of the algorithm as a function of n, the value of the input integer.

```
RecursiveFibonacci(n)

If (n = 0), Return( 0 )
If (n = 1), Return( 1 )

f := RecursiveFibonacci(n-1) + RecursiveFibonacci(n-2)

Return( f )
```

©zyBooks 12/15/22 00:33 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

EXERCISE | 16.1.6: A fast algorithm to multiply two non-negative integers, cont.

(a) The following algorithm takes as input two non-negative integers, x and y, and returns the product of x and y.

```
FastMult(x, y)

Input: Two non-negative integers, x and y
Output: The product of x and y

If (y = 0), Return( 0 )

z := ⌊y/2⌋
p := FastMult(x, z) // The recursive call

If (y is even)
    Return( 2 · p )
Else
    Return( 2 · p + x )
End-if
```

Give a recurrence relation to describe the asymptotic complexity of the algorithm.

16.2 Divide-and-conquer algorithms: Introduction and mergesort

This section has been set as optional by your instructor.

Divide-and-conquer algorithms

A ***divide-and-conquer*** algorithm solves a problem recursively by breaking the original input into smaller subproblems of roughly equal size. There are three basic steps in a divide-and-conquer algorithm:

Break the input into smaller subproblems of the same type on smaller inputs

Solve each subproblem recursively

Combine the solutions of the subproblems to obtain a solution to the original problem

For example, a divide-and-conquer algorithm to find the smallest number in a list would break the list into two halves, recursively find the smallest number in each half, compare the two smallest numbers from each half, and return the smaller of the two. The base case occurs when the list only has one item, in which case the smallest number is the only number in the list.

```
FindMin(L)

If L has only one item x, Return(x)

Break list L into two lists, A and B

a := FindMin(A)
b := FindMin(B)

If ( a ≤ b ), then Return(a)
Else Return(b)
```


PARTICIPATION ACTIVITY 16.2.1: Illustration of divide-and-conquer FindMin.

Animation captions:

1. Divide the input list (17, 3, 9, 11, 21, 4, 7, 2) into two lists: (17, 3, 9, 11) and (21, 4, 7, 2).
2. Recursively find the smallest item in the first list (17, 3, 9, 11). The base case occurs when the list has only one item (17). FindMin returns 17.
3. When the two recursive calls are complete, FindMin returns the smaller of the two values.
4. The recursive call to (17, 3, 9, 11) returns 3.
5. Now FindMin makes a recursive call on the second half of the list (21, 4, 7, 2) which returns 2.
6. The recursive call on (17, 3, 9, 11) returns 3. The recursive call on (21, 4, 7, 2) returns 2. FindMin returns the smaller of 2 and 3, which is 2.

PARTICIPATION ACTIVITY 16.2.2: Divide-and-conquer compute sum.

The instructions below give the basic steps in a divide-and-conquer algorithm that computes the sum of all the numbers in a list:

- Break the list into two lists, each with half the numbers: List1 and List2
- R1 := the value returned from a recursive call to the algorithm with input List1.
- R2 := the value returned from a recursive call to the algorithm with input List2.

- Return (?)

1) What value should the algorithm return?

- ○ 2 · R1
- ○ R2
- ○ R1 + R2
- ○ min{R1, R2}

2) Suppose the base case occurs when there is only one number in the list (x). What value should the algorithm return for the base case?

- ○ 0
- ○ 1
- ○ x

Mergesort

A ***sorting algorithm*** takes as input a list of items and returns the same list sorted in ascending order. On input (12, 4, 7, 9), for example, the correct output is (4, 7, 9, 12). The items to be sorted must have a well-defined ordering. For example, integers can be sorted by value and names can be sorted according to alphabetical order.

Mergesort is a divide-and-conquer algorithm that sorts a list of items recursively by dividing the list into two sub-lists of roughly half the size, recursively sorting each sub-list and merging the sorted sub-lists. Mergesort would sort the list (17, 3, 9, 11, 21, 4, 7, 1) in four basic steps:

Break the list (17, 3, 9, 11, 21, 4, 7, 1) into two sub-lists: (17, 3, 9, 11) and (21, 4, 7, 1).

Make a recursive call to mergesort with input (17, 3, 9, 11), which returns (3, 9, 11, 17).

Make a recursive call to mergesort with input (21, 4, 7, 1), which returns (1, 4, 7, 21).

Merge the sorted sub-lists to obtain the final sorted list (1, 3, 4, 7, 9, 11, 17, 21).

The base case occurs when the input list only has one item, in which case the list is already sorted.

PARTICIPATION ACTIVITY 16.2.3: Recursion tree for mergesort.

Animation content:

```
undefined
```

Animation captions:

1. On input (17, 2, 9, 11, 21, 4, 7, 1), recursively call mergesort on (17, 2, 9, 11). Continue to call mergesort on left half of the list until base case with input (17) is reached.
2. Return (17) as the list is already sorted. In each call to mergesort, when both sub-lists have been sorted, merge the two sub-lists.
3. The list (2, 9, 11, 17) is returned from the recursive call to (17, 2, 9, 11). Next make a recursive call on the right half of the input list: (21, 4, 7, 1).
4. The recursive call on the right half of the list returns (1, 4, 7, 12). Now that both halves have been sorted, merge the two sorted sub-lists to get (1, 2, 4, 7, 9, 11, 17, 21), the final sorted list which is returned.

PARTICIPATION ACTIVITY 16.2.4: Recursive calls in mergesort.

1) If mergesort is called an input (5, 4, 3, 2, 9, 8, 7, 6), then what would be the output of the two recursive calls?
   - (5, 4, 3, 2) and (9, 8, 7, 6)
   - (2, 3, 4, 5) and (6, 7, 8, 9)
   - (2, 3, 4, 5, 6, 7, 8, 9)

The queue data structure

A ***data structure*** is a way of organizing data in a computer so that the data can be accessed and updated efficiently. A programmer selects a data structure for an application depending on what particular operations she would like to perform on the data. The version of MergeSort presented in this material uses a data structure called a queue. A ***queue*** maintains items in an ordered list. New items can be added to one end of the queue called the ***back***. Items can be removed from the other end of the queue called the ***front***. A queue operates like a line at the grocery store in which the customers are items. Customers join the line at the back and leave the line from the front. The details of how to implement a queue are beyond the scope of this material. The important fact for analyzing MergeSort is that the following operations can be implemented in $\Theta(1)$ time:

- A := createEmptyQueue(). Creates a queue with no items and assigns the queue to variable A.

- x := front(A). Returns the item at the front of queue A but does not change A.
- add(x, A). Adds item x to the back of queue A.
- x := remove(A). Removes the item x at the front of queue A and returns x.
- n := size(A). Returns a non-negative integer whose value is the number of items in queue A.

The operations front(A) and remove(A) result in an error if A is empty.

The version of MergeSort presented in this material sorts by moving items between different queues. It is more efficient to implement MergeSort by rearranging the items within a single list or array. However, using queues makes the pseudo-code simpler. All versions of MergeSort run in time $\Theta(n \log n)$ on a list of n items, so the efficiency of different versions only differ by constant factors.

PARTICIPATION ACTIVITY 16.2.5: Simulating operations on a queue.

Animation captions:

1. The operations create an empty list called A and add three items, 6, 7, and 5, to A. The list A is now (6, 7, 5).
2. The operation front(A) returns 6, the item at the front of the list. A is still (6, 7, 5).
3. The operation remove(A) returns 6, the item at the front of the list. 6 is no longer in A, so A is now (7, 5).
4. Size(A) returns 2 because there are two items in A.
5. add(3) adds 3 to the back of A. A is now (7, 5, 3).

PARTICIPATION ACTIVITY 16.2.6: Identifying the output of operations on queues.

Match each object to its description. The queues A and B are:

A: (9, 5, 8, 7)
B :(5, 8, 7)

If unable to drag and drop, refresh the page.

(9, 5, 8, 7) **3** **5** **(5, 8, 7, 9)** **(5, 8, 7)**

The state of A after operation front(A)

The state of A after operation remove(A)

The value returned from front(B) operation

The value returned by size(B)

The state of B after add(9)

Reset

The merge operation

The input to the merge operation are two sorted lists, A and B. The output is a single list C that contains all the items from A and B in sorted order. The first loop continues to iterate as long as A and B are both non-empty. Each iteration proceeds by selecting the smaller of the two items at the front of A and B, removing that item, and adding the item to the back of C. Loop 1 ends when either A or B becomes empty.

Immediately after Loop 1, only one of A or B has any items left. If A still has items, then Loop 2 removes each item from A and adds the item to the back of C. If B still has items, then Loop 3 removes each item from B and adds the item to the back of C.

In each iteration of the three loops, one item is removed from A or B and added to C. By the end of Loop 3, both A and B are empty. Therefore the total number of iterations in all three loops is exactly n, where n is the total number of items in A and B at the beginning of the merge. Each iteration performs $\Theta(1)$ operations. Therefore the running time of the Merge operations is $\Theta(n)$.

PARTICIPATION ACTIVITY 16.2.7: Simulating the merge operation.

Animation content:

undefined

Animation captions:

1. The input to the Merge operation is A: (3, 5, 9, 15, 17) and B: (7, 11, 12, 20, 21). Start by creating a new empty queue C.
2. In each iteration of Loop 1, the smaller of the items at the front of A and B is removed and added to C. Loop 1 ends when A is empty and B is (20, 21)
3. Since A is empty, there are no iterations in Loop 2.
4. In Loop 3, each item is removed from B and added to C until B is empty.
5. Return(C). The output of the Merge operation is the merged list C.

PARTICIPATION ACTIVITY 16.2.8: Simulating the merge operation.

The input to the merge operation is:
A: (3, 7, 17, 19, 25)
B: (5, 6, 8, 10, 15)

1) How many times does Loop 1 iterate?

Check Show answer

2) How many times does Loop 2 iterate?

Check Show answer

3) How many times does Loop 3 iterate?

Check Show answer

Pseudo-code for mergesort

The pseudo-code for mergesort is given below. The base case occurs when the size of the input list X is 1. If size(X) = n and $n > 1$, then the items in X are split between two sub-lists, L and R. The first $\lceil n/2 \rceil$ items from X are placed in L, and the remaining $\lfloor n/2 \rfloor$ items from X are placed in R. Then L and R are sorted recursively by calls to mergesort. The two sorted sub-lists that are returned from the recursive calls are sent to the merge operation which returns the final sorted list.

PARTICIPATION ACTIVITY 16.2.9: Simulating the pseudo-code for mergesort.

Animation captions:

1. The input to MergeSort is the list X = (17, 5, 13, 23, 9, 21, 7, 1, 14). Since X has 9 items, the

conditions for the base case are not met.
2. Empty list L and R are created. size(X) = n = 9. m = $\lceil 9/2 \rceil$ = 5.
3. Move m = 5 items from X to L. Afterwards, L is (17, 5, 13, 23, 9) and X is (21, 7, 1, 14).
4. Move n - m = 4 items from X to R. Afterwards, R is (21, 7, 1, 14) and X is empty.
5. Make recursive calls to MergeSort on input L and R. The output are sorted sub-lists L: (5, 9, 13, 17, 23) and R: (1, 7, 14, 21).
6. Call Merge on input A and B. Merge returns the merged list C (1, 5, 7, 9, 13, 14, 17, 21, 23). The sorted list C is returned.

PARTICIPATION ACTIVITY 16.2.10: The recurrence relation for the asymptotic complexity of mergesort.

T(n) is defined to be the number of operations performed by MergeSort on an input list of n items.

1) For input list X with n items, how many operations are performed in the first five lines of MergeSort?
   - $\Theta(1)$
   - $\Theta(\log n)$
   - $\Theta(n)$

2) For input list X with n items, how many operations are performed in the two For-loops?
   - $\Theta(1)$
   - $\Theta(n)$
   - $T(n/2)$

3) Which expression best describes the number of operations that are performed during the two recursive calls made by MergeSort?
   - $\Theta(n)$
   - $T(n/2)$
   - $2 \cdot T(n/2)$

4) The running time for the call to Merge is $\Theta(n)$ for an input list with n items.

Select the recurrence relation that describes the asymptotic complexity of MergeSort.

- $T(n) = T(n/2) + \Theta(1)$
- $T(n) = 2 \cdot T(n/2) + \Theta(1)$
- $T(n) = T(n/2) + \Theta(n)$
- $T(n) = 2 \cdot T(n/2) + \Theta(n)$

Additional exercises

EXERCISE 16.2.1: Recurrence relation for divide-and-conquer FindMin.

(a) The pseudo-code for the divide-and-conquer FindMin is given below. Give a recurrence relation that describes the number of operations performed on an input list with n items.

FindMin(n, $(a_1, a_2, \ldots, a_n)$)

If (n = 1), Return(a_1)

m = $\lceil n/2 \rceil$
List1 := $(a_1, a_2, \ldots, a_m)$
List2 := $(a_{m+1}, a_{m+2}, \ldots, a_n)$
min1 := FindMin(m, List1)
min2 := FindMin(n-m, List2)

If (min1 < min2), Return(min1)
Return(min2)

EXERCISE 16.2.2: Recursion tree for mergesort.

(a) Give the recursion tree for a call to Mergesort with input list (6, 2, 21, 5, 17, 1, 15). For each call to Mergesort, give the input list to that call, the input and output for the recursive calls made and the final output list for that call.

EXERCISE 16.2.3: Adapting mergesort to remove duplicates.

(a) Change mergesort so that the algorithm removes duplicate items in the list as well as sorts the remaining items. For example, on input (15, 2, 21, 5, 15, 2, 15), the output should be (2, 5, 15, 21).

EXERCISE 16.2.4: Simplifying mergesort with a maximum item.

(a) Suppose that there is a constant named MaxItem that is the same type as the items to be sorted and is guaranteed to be larger than any of the items in the input list. Use MaxItem to write a simplified merge operation.
Hint: Add MaxItem to the back of the two lists to be sorted.

EXERCISE | 16.2.5: 3-way mergesort.

Here is a 3-way version of mergesort:

```
MergeSort(X)

If (??), Return(X)

L1 := createEmptyQueue()
L2 := createEmptyQueue()
L3 := createEmptyQueue()

n := size(X)
m1 := ⌈n/3⌉
m2 := ⌈(n − m1)/2⌉

For i := 1 to m1
    x := remove(X)
    add(L1, x)
End-for

For i := 1 to m2
    x := remove(X)
    add(L2, x)
End-for

For i := 1 to (n-m1-m2)
    x := remove(X)
    add(L3, x)
End-for

A := MergeSort(L1)
B := MergeSort(L2)
C := MergeSort(L3)

D := 3WayMerge(A, B, C)

Return(D)
```


©zyBooks 12/15/22 00:33 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

3wayMerge is an operations that takes as input 3 sorted lists and returns a single sorted lists that contains all the items in the 3 input lists.

(a) Give the sizes of the three sub-lists when the input list has n items, for n = 9, 10, and 11.

(b) What should the condition be for the base case?

(c) Show that if the input list has n items, then the sizes of the three sub-lists are $\lfloor n/3 \rfloor$ or $\lceil n/3 \rceil$.
Hint: consider three cases: n = 3k, or n = 3k+1, or n = 3k+2, for some integer k.

(d) Give a recurrence relation describing the asymptotic complexity of 3WayMergeSort. You can assume that 3WayMerge runs in $\Theta(n)$ time, where n is the sum of the sizes of the three input lists.

16.3 Divide-and-conquer algorithms: Binary search

This section has been set as optional by your instructor.

Binary search

The goal of a ***search algorithm*** is to find a target item in a list. For example, consider a student trying to find his name in a list of students enrolled in a class. Assume, for simplicity, that no two students have the same name. If the names in the list are in an arbitrary order, then he must start at the beginning of the list and scan through the names until he finds his name or gets to the end of the list. If the class list is sorted alphabetically, then he should be able to find his name more quickly. ***Binary search*** is an efficient algorithm to search for a target item in a sorted list.

The input to binary search is a sorted list of distinct items and a target item x. If x is in the list, then binary search returns the location of x in the list. If x is not in the list, then binary search returns -1.

PARTICIPATION ACTIVITY 16.3.1: Binary search input and output.

Animation captions:

1. On input list (2, 3, 7, 10, 21, 32) and target item x = 18, the target item is not in the list, so the correct output is -1.
2. On input list (2, 3, 7, 10, 21, 32) and target item x = 10, the target item is the 4th item in the list, so the correct output is 4.

PARTICIPATION ACTIVITY 16.3.2: Input and output for binary search.

The input to binary search is the list:

(2, 6, 9, 13, 17, 23, 29, 31, 34, 45)

The locations in the list are numbered from 1 to 10.

1) What is the correct output for target x = 18?

Check Show answer

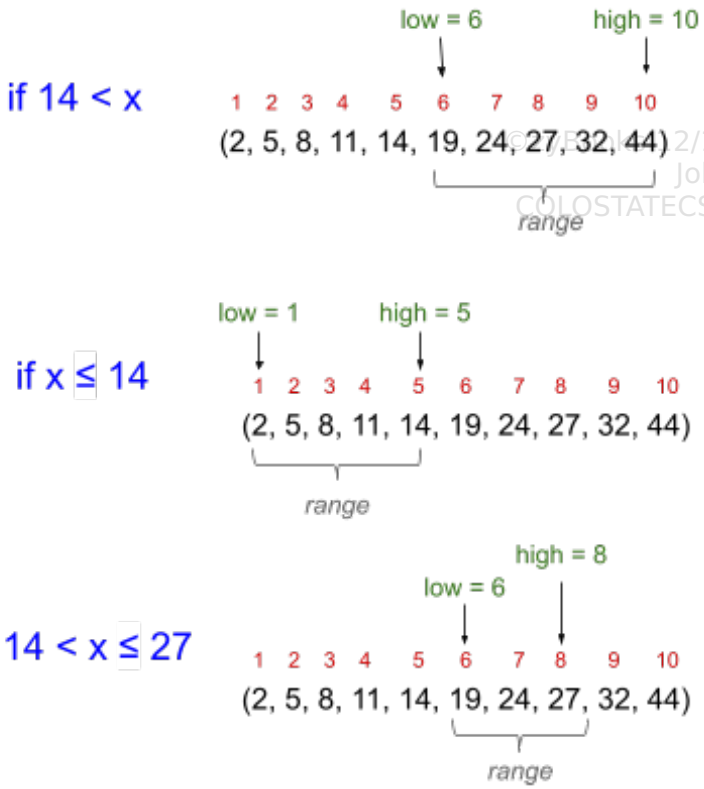
2) What is the correct output for target x = 9?

Check Show answer

Recursive binary search

A recursive version of binary search takes two additional input values, low and high. If n is the number of items in the list, then variables low and high must lie in the range 1 through n and must satisfy the inequality low $\leq$ high. The variables low and high denote a range of candidate locations {low, low+1, …, high} so that if x is in the list at all, then x must be reside in one of the locations in the range. For the example L = (2, 4, 7, 10, 21, 32), if $10 < x$, then x must occur after the 10 in the list. Since the 10 is in location 4, x must occur in location 5 or later in the list and low can be set to 5. If $x \leq 10$, then x must occur in location 4 or earlier in the list, and high can be set to 4. Initially, the algorithm is called with low = 1 and high = n because the range of candidate locations is the entire list.

Figure 16.3.1: Setting variable low and high in binary search.



PARTICIPATION ACTIVITY 16.3.3: Setting variables low and high in binary search.

The input to a recursive binary search algorithm is target x and list L:

(3, 5, 8, 11, 14, 18, 23, 31, 37, 40)

The algorithm sets the variables low and high so that if x occurs in the list, then x is guaranteed to occur somewhere in locations low through high.

1) If $11 < x$, which assignment is guaranteed to maintain the correct properties of low and high?

- low := 5
- low := 6
- high := 4
- high := 5

2) If $x \leq 23$, which assignment is guaranteed to maintain the correct properties of low and high?

- low := 7
- low := 8
- high := 6
- high := 7

3) If $8 < x \leq 31$, which assignments are guaranteed to maintain the correct properties of low and high?

- low := 4
  high := 8
- low := 3
  high := 7
- low := 5
  high := 9

Pseudo-code

In each recursive call, binary search compares the target item x to the middle item in the range of candidate locations and reduces the size of the range by a half. The base case occurs when the range of candidate locations is down to one (low = high). Then x can be compared to the single item in the range to determine whether x is in the list.

©zyBooks 12/15/22 00:33 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

Figure 16.3.2: Pseudo-code for recursive binary search.

```
RecBinarySearch(low, high, A, x)

If (low = high AND a_low = x),
Return(low)
If (low = high AND a_low ≠ x), Return(-1)

mid := ⌊(low + high)/2⌋

If (x ≤ a_mid), then high := mid
If (x > a_mid), then low := mid + 1

Return( RecBinarySearch(low, high, A,
x) )
```


PARTICIPATION ACTIVITY 16.3.4: Recursive binary search simulation.

Animation captions:

1. The input is a sorted list of 7 numbers. low =1 and high = 7. Target x = 14. mid = $\lfloor(1 + 7)/2\rfloor$ = 4. Compare x to the number in location 4, which is 11.
2. 11 < 14, so low becomes mid+1 = 5.
3. mid = $\lfloor(5 + 7)/2\rfloor$ = 6. The number in location 5 is 19. Since x ≤ 19, high is reset to mid = 6.
4. mid = $\lfloor(5 + 6)/2\rfloor$ = 5. The number in location 5 is 14. Since x ≤ 14, high is reset to mid = 5.
5. Since low = high, the conditions for the base case are met. Since x = 14 equals the number in location low, return low. The target number has been found.

PARTICIPATION ACTIVITY 16.3.5: Simulating binary search.

Binary search is called with an input list:

(2, 5, 8, 11, 14, 19, 24)

The target item is x = 6.

For the questions below, enter the values for low and high separated by a comma. For

example, for answer low = 3 and high = 7, enter: 3, 7.

RecBinarySearch(low, high, A, x)

If (low = high AND $a_{low} = x$), Return(low)
If (low = high AND $a_{low} \neq x$), Return(-1)

mid := $\lfloor (low + high)/2 \rfloor$

If ($x \leq a_{mid}$), then high := mid
If ($x > a_{mid}$), then low := mid + 1

Return(RecBinarySearch(low, high, A, x))

1) What are the values of low and high for the initial call to RecBinarySearch?

Check Show answer

2) What are the values of low and high for the first recursive call to RecBinarySearch?

Check Show answer

3) What are the values of low and high for the second recursive call to RecBinarySearch?

Check Show answer

4) What are the values of low and high for the last recursive call to RecBinarySearch?

Check Show answer

5) What is the final return value of RecBinarySearch?

Check Show answer

Analyzing recursive binary search

The time complexity of an algorithm is a function of the size of the input. For binary search, the size of the input is the number of candidate locations in the range {low, ..., high}, which is (high - low +1). Since binary search is initially called with low = 1 and high = n, the size of the input is (n - 1 + 1) = n, the number of items in the input list.

Binary search assigns mid := $\lfloor (high + low)/2 \rfloor$ and breaks the range of locations into two sub-ranges: {low, low+1, ..., mid} and {mid+1, mid+2, ..., high}. The size of the first sub-range is (mid - low + 1). The size of the second sub-range is (high - (mid+1) + 1) = (high - mid). If the variable m denotes the size of the original range (high - low + 1), the fact below indicates that the sizes of the two sub-ranges are m/2 and m/2, in the case that m is even. The sizes of the two sub-ranges are $\lceil m/2 \rceil$ and $\lfloor m/2 \rfloor$, in the case that m is odd.

Theorem 16.3.1: Sizes of the subranges in binary search.

Suppose that high and low are two positive integers such that low $\leq$ high, and define the variable m = (high - low + 1) and mid = $\lfloor (high + low)/2 \rfloor$.

- If m is even, then the sizes of the two subranges are both m/2: (mid - low + 1) = (high - mid) = m/2.
- If m is odd, then the size of the first subrange is (mid - low + 1) = $\lceil m/2 \rceil$, and the size of the second subrange is (high - mid) = $\lfloor m/2 \rfloor$.

PARTICIPATION ACTIVITY 16.3.6: Sizes of the subranges in binary search.

Animation captions:

1. When high = 20 and low = 15, the number of locations in the range is 6 which is even. mid = $\lfloor (20 + 15)/2 \rfloor$ = 17.
2. The two sub-ranges are 15 through 17 and 18 through 20. The size of each sub-range is 3 = 6/2.
3. When high = 21 and low = 15, the number of locations in the range is 7 which is odd. mid = $\lfloor (21 + 15)/2 \rfloor$ = 18.
4. The two sub-ranges are 15 through 18 and 19 through 21. The size the first sub-range is 4 = $\lceil 7/2 \rceil$. The size of the other is 3 = $\lfloor 7/2 \rfloor$.

PARTICIPATION ACTIVITY 16.3.7: Recurrence relation for binary search.

1) What is the number of recursive calls made in one call of RecBinarySearch for the case that low < high?

- 0
- 1
- 2

2) The size of the input to a call to RecBinarySearch is the number of locations in the range {low, ...,high}. If the size of the input to RecBinarySearch is n, then which expression is closest to the size of the input for the recursive call?

- 1
- n/2
- n-1

3) Let T(n) denote the number of atomic operations performed by RecBinarySearch on a list of n items. What is the recurrence relation that describes the asymptotic complexity of RecBinarySearch ?

Additional exercises

EXERCISE 16.3.1: Simulating binary search.

The input list of binary search is:

(2, 5, 8, 10, 13, 19, 21, 32, 37, 52)

For each target value x given below, give the values for variables low and high for each call to BinarySearch. Then give the final value return value.

(a) x = 13

(b) x = 1

(c) x = 35

(d) x = 52

EXERCISE 16.3.2: Proving Theorem 18.14.1.

(a) Prove the theorem:
Theorem: Suppose that high and low are two positive integers such that low $\leq$ high, and define the variable m = (high - low + 1) and mid = $\lfloor (high + low)/2 \rfloor$.

- If m is even, then the sizes of the two subranges are both m/2: (mid - low + 1) = (high - mid) = m/2.
- If m is odd, then the size of the first subrange is (mid - low + 1) = $\lceil m/2 \rceil$, and the size of the second subrange is (high - mid) = $\lfloor m/2 \rfloor$.

Hint: consider two cases depending on whether the value (high - low + 1) is even or odd. Then, based on whether (high - low + 1) is even or odd, determine whether (high + low) is even or odd. The exact value for mid can then be determined (without the ceiling or floor function) using the following fact:
Fact: If n is even, then $\lceil n/2 \rceil = \lfloor n/2 \rfloor = n/2$. If n is odd, then $\lceil n/2 \rceil = n/2 + (1/2)$ and $\lfloor n/2 \rfloor = n/2 - (1/2)$.

EXERCISE 16.3.3: Worst and best case running time for binary search.

(a) Consider running binary search on input list:
(2, 5, 8, 11, 15, 18, 23, 29, 31, 35)
In searching for a target item x that is in the list, the running time of the algorithm may differ depending on which item from the list is the target. Give a target item x, selected from the list, that results in the best running time. Given a target item, selected from the list, that results in the worst running time.

(b) For some input lists, the time to search for an item in the list under binary search is the same for all items in the list. Characterize the input lists that have this property. Assume the items in the list are all different.

EXERCISE 16.3.4: Three-way binary search.

The psuedo-code below gives a slightly different version of binary search:

```
RecBinarySearch2(low, high, A, x)

If (??), Return(-1)

mid := ⌊(low + high)/2⌋

If (x = a_mid), then Return( mid )
If (x < a_mid), then high := mid - 1
If (x > a_mid), then low := mid + 1

Return( RecBinarySearch2(low, high, A, x) )
```

(a) What is the right condition for the base case?

(b) Consider running RecBinarySearch2 on input list:
(2, 5, 8, 11, 15, 18, 23, 29, 31, 35)
In searching for a target item x that is in the list, the running time of the algorithm may differ depending on which item from the list is the target item. Give a target item x, selected from the list, that results in the best running time. Given a target item, selected from the list, that results in the worst running time.

16.4 Solving linear non-homogeneous recurrence relations

This section has been set as optional by your instructor.

The associated homogeneous recurrence relation

A ***non-homogeneous linear recurrence relation*** is a linear recurrence relation with additional terms that are either a constant or a function of n. The recurrence relations below are all examples of non-homogeneous linear recurrence relations.

$$f_n = 3f_{n-1} + 10f_{n-2} + 2$$
$$f_n = 3f_{n-1} + 10f_{n-2} + 24n$$
$$f_n = 3f_{n-1} + 10f_{n-2} + 2^n + 3n$$

The ***associated homogeneous recurrence relation*** is the recurrence relation with the additional non-homogeneous terms dropped. For example, the associated homogeneous recurrence relation for all of the recurrence relations given above is $f_n = 3f_{n-1} + 10f_{n-2}$.

Table 16.4.1: Examples of non-homogeneous linear recurrence relations and their associated homogeneous recurrence relation.

Non-homogeneous recurrence relation	Associated homogeneous recurrence relation
$f_n = 2f_{n-1} + 12f_{n-3} + 3 \cdot 7^n + 21$	$f_n = 2f_{n-1} + 12f_{n-3}$
$g_n = 3g_{n-2} + \sqrt{n} + \log n$	$g_n = 3g_{n-2}$
$h_n = -h_{h-1} - 4h_{n-4} + n!$	$h_n = -h_{n-1} - 4h_{n-4}$

PARTICIPATION ACTIVITY 16.4.1: Associated homogeneous recurrence relations.

1) Select the homogeneous recurrence relation associated with

$f_n = 5f_{n-1} - 3f_{n-2} + 3n^2 + 7n - 5$

- $f_n = 5f_{n-1}$
- $f_n = 5f_{n-1} - 3f_{n-2}$
- $f_n = 5f_{n-1} - 3f_{n-2} - 5$
- $f_n = 5f_{n-1} - 3f_{n-2} + 7n - 5$

Particular and homogeneous solutions

The solution to a non-homogeneous linear recurrence relation is the sum of two parts: a ***homogeneous solution*** plus a ***particular solution***. If the sequence is $\{f_n\}$, then the homogeneous solution is denoted by $f_n^{(h)}$ and the particular solution is denoted by $f_n^{(p)}$.

- The homogeneous solution $f_n^{(h)}$ is the general solution to the associated homogeneous recurrence relation. For recurrence relation $f_n = 3f_{n-1} + 10f_{n-2} + 24n$, the homogeneous solution $f_n^{(h)}$ is the general solution to $f_n = 3f_{n-1} + 10f_{n-2}$, whose characteristic equation is $x^2 - 3x - 10 = (x - 5)(x + 2) = 0$. Therefore, $f_n^{(h)} = a_1 5^n + a_2(-2)^n$. The details of finding a solution to a linear homogeneous recurrence relation are covered elsewhere.
- The method for finding the particular solution $f_n^{(p)}$ is to guess the form of the particular solution and then check the guess. For recurrence $f_n = 3f_{n-1} + 10f_{n-2} + 24n$, the additional term 24n is linear in n. Therefore, a reasonable guess is that the particular solution is also linear. The form of a linear solution is $(an + b)$ for some constants a and b. The form $(an + b)$ is valid if it is possible to solve for constants a and b so that $f_n = (an + b)$ satisfies the recurrence relation for every value of n.

PARTICIPATION ACTIVITY 16.4.2: Checking a particular solution.

Animation captions:

1. For recurrence $f_n = 3f_{n-1} + 10f_{n-2} + 24n$, guess particular solution $f_n^{(p)} = an + b$.
2. Plug the particular solution into the recurrence to solve for a and b. The equation must be true for all n if $f_n^{(p)} = an + b$ is a valid particular solution.
3. Collect linear terms in n to solve for a.
4. Collect the constant terms (which do not have a factor of n). Plug in the value for a in order

to solve for b.

5. Plug values a = -2 and b = -23/6 back into the particular solution $f_n^{(p)} = an + b = -2n - (23/6)$.

PARTICIPATION ACTIVITY 16.4.3: Finding the constants in the form of a particular solution.

Consider the recurrence relation

$f_n = 5f_{n-1} - 6f_{n-2} + 7^n$.

The non-homogeneous term is 7^n so the guess for the form of the particular solution will be $c7^n$, for some constant c.

1) Select the equation that results from plugging in

$f_n = c7^n$
$f_{n-1} = c7^{n-1}$
$f_{n-2} = c7^{n-2}$

into the recurrence relation.

- $c7^n = 5c7^{n-1} - 6c7^{n-2} + c7^n$
- $c7^n = 5c7^n - 6c7^n + c7^n$
- $c7^n = 5 \cdot 7^{n-1} - 6 \cdot 7^{n-2} + 7^n$
- $c7^n = 5c7^{n-1} - 6c7^{n-2} + 7^n$

2) Select the equation that results from taking the answer to the previous question and dividing both side by 7^{n-2}.

- $49c = 35c - 6c + 1$
- $49c = 7c - 6c + 49$
- $49c = 35c - 6c + 49$
- $49c = 35c - 6c + 49c$

3) The answer to the previous question can be simplified as $20c = 49$. What is the particular solution for the

recurrence relation?

- $f_n = \frac{49}{20} \cdot 7^n$
- $f_n = 7^{49n/20}$
- $f_n = \frac{20}{49} \cdot 7^n$

Using the initial conditions to derive the final solution

The recurrence relation $f_n = 3f_{n-1} + 10f_{n-2} + 24n$ has homogeneous solution $f_n^{(h)} = a_1 5^n + a_2(-2)^n$ and particular solution $f_n^{(p)} = -2n - \frac{23}{6}$. The general solution to the recurrence relation is:

$$f_n = f_n^{(h)} + f_n^{(p)} = a_1 5^n + a_2(-2)^n - 2n - \frac{23}{6}$$

The final step is to use the initial conditions to solve for the constants a_1 and a_2. Suppose that the initial conditions are:

$$f_0 = \frac{7}{6} \quad f_1 = \frac{-11}{6}$$

Each of the initial values results in a linear equation. The first equation is determined by plugging in n = 0 into the general solution for f_n and setting the resulting expression equal to $f_0 = 7/6$. The second equation is determined by plugging in n = 1 into the general solution for f_n and setting the resulting expression equal to $f_1 = -11/6$. The two equations are linear in a_1 and a_2 and can be used to solve for a_1 and a_2.

PARTICIPATION ACTIVITY 16.4.4: Solving for the constants in a general solution.

Animation content:

undefined

Animation captions:

1. The general solution is $f_n = a_1 5^n + a_2(-2)^n - 2n - \frac{23}{6}$. The initial conditions are $f_0 = 7/6$ and $f_1 = -11/6$.
2. Plug in $n = 0$ to the general solution. The resulting expression $a_1 5^0 + a_2(-2)^0 - 2 \cdot 0 - \frac{23}{6}$ is equal to the initial value of $f_0 = \frac{7}{6}$.

3. Simplifying the equation for $n = 1$ results in the equation $-\frac{11}{6} = a_1 + a_2 - \frac{23}{6}$.
4. Plug in $n = 1$ into the general solution. The resulting expression $5a_1 - 2a_2 - 2 - \frac{23}{6}$ is equal to the initial value of $f_1 = -\frac{11}{6}$.
5. Solving the two linear equations results in $a_1 = 2$ and $a_2 = 3$. Plugging the values for a_1 and a_2 back into the general solution results in the final solution: $f_n = 2 \cdot 5^n + 3(-2)^n - 2n - \frac{23}{6}$.

PARTICIPATION ACTIVITY 16.4.5: Linear equations to find constants in the general solution.

The general solution to

$f_n = 5f_{n-1} - 6f_{n-2} + 7^n$

is

$f_n = a_1 2^n + a_2 3^n + \frac{49}{20} \cdot 7^n$

1) What is the linear equation that results from the initial value $f_0 = 11$?

- $11 = a_1 + a_2 + \frac{49}{20}$
- $11 = 2a_1 + 3a_2 + \frac{49}{20}$
- $11 = a_1 + a_2 + \frac{49}{20} \cdot 7$

2) What is the linear equation that results from the initial value $f_1 = 13$?

- $13 = a_1 + a_2 + \frac{49}{20}$
- $13 = 2a_1 + 3a_2 + \frac{49}{20}$
- $13 = a_1 + a_2 + \frac{49}{20} \cdot 7$
- $13 = 2a_1 + 3a_2 + \frac{49}{20} \cdot 7$

Summary of all the steps

The table below summarizes the steps for solving a linear non-homogeneous recurrence relation. The column on the right shows the result of each step on the example $f_n = 3f_{n-1} + 10f_{n-2} + 24n$ with initial values $f_0 = 7/6$ and $f_1 = -11/6$.

©zyBooks 12/15/22 00:33 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

Table 16.4.2: Solving linear non-homogeneous recurrence relations.

General step	Example: $f_n = 3f_{n-1} + 10f_{n-2} + 24n$
Find the homogeneous part of the solution which is the general solution to the associated homogeneous recurrence relation.	The homogeneous part is $f_n^{(h)} = a_1 5^n + a_2(-2)^n$ which is the general solution to $f_n = 3f_{n-1} + 10f_{n-2}$
Guess the correct form for the particular solution.	Guess $f_n^{(p)} = an + b$ for some constants a and b.
Verify the guess by solving for constants so that the guess satisfies the recurrence relation for all n.	Find constants a and b so that $an + b = 3(a(n-1) + b) + 10(a(n-2) + b) + 24n$ for every n. $a = -2 \quad b = \frac{-23}{6}$
Add the homogeneous and particular solutions to get the general solution.	$f_n = f_n^{(h)} + f_n^{(p)} = a_1 5^n + a_2(-2)^n - 2n - \frac{23}{6}$
For a degree d linear recurrence relation, there must be d initial values to specify the sequence. Each initial value gives a value of f_n for a specific value for n. Plug the values for n and f_n into the general solution to get a linear equation with variables $a_1, \ldots, a_d$. There are d linear equations (from the d initial values) and d variables.	$n = 0: f_0 = \frac{7}{6} = a_1 + a_2 - \frac{23}{6}$ $n = 1: f_1 = \frac{-11}{6} = 5a_1 - 2a_2 - \frac{35}{6}$

Solve for $a_1, \ldots, a_d$.	$a_1 = 2, \quad a_2 = 3$
Plug values for $a_1, \ldots, a_d$ back in to the general solution for f_n to get the final closed form expression for f_n.	$f_n = 2 \cdot 5^n + 3(-2)^n - 2n - \frac{23}{6}$

PARTICIPATION ACTIVITY 16.4.6: Solving linear non-homogeneous recurrence relations.

The homogeneous solution to

$g_n = 6g_{n-1} - 9g_{n-2} + 16$

is

$g_n^{(h)} = a_1 3^n + a_2 n 3^n$

1) Since the non-homogeneous term in the recurrence relation is a constant (+16), the guess for the form of the particular solution will be that $g_n^{(p)} = c$, for some constant c. Select the equation to use to solve for c.
   - $c = 6c - 9c + 16$
   - $c = 6c - 9c + 16c$
   - $c = 6(c-1) - 9(c-2) + 16$

2) What is the general solution to the recurrence relation $g_n = 6g_{n-1} - 9g_{n-2} + 16$?
   - $g_n = a_1 3^n + a_2 n 3^n + 16$
   - $g_n = a_1 3^n + a_2 n 3^n + 4$
   - $g_n = a_1 3^n + a_2 n 3^n$
   - $g_n = 4$.

3) What is the linear equation that

results from the initial value $g_0 = 5$?

- $5 = a_1 + 4$
- $5 = a_1 + a_2 + 4$
- $5 = 3a_1 + 3a_2 + 4$

4) What is the linear equation that results from the initial value $g_1 = 13$?

- $13 = a_1 + a_2 + 4$
- $13 = 3a_1 + 9a_2 + 4$
- $13 = 3a_1 + 3a_2 + 4$

Determining the form of the particular solution

There is no rule for determining the form of the particular solution for a linear non-homogeneous recurrence relation that works in every situation. However, there is a rule that works when the non-homogeneous terms have a particular common form. Suppose the recurrence relation is

$$f_n = c_1 f_{n-1} + c_2 f_{n-2} + \cdots c_d f_{n-d} + F(n)$$

The function F(n) includes all the non-homogeneous terms. The theorem below provides a way to determine the form of the specific solution when F(n) is a polynomial times an exponential: $p(n) \cdot s^n$ for some constant s. The polynomial p(n) can be a degree 0 polynomial, which is just a constant. Also, the value of s can be 1, in which case F(n) is a polynomial.

Theorem 16.4.1: The form of particular solutions to certain linear non-homogeneous recurrence relations.

Suppose the sequence $\{f_n\}$ is described by the linear non-homogeneous recurrence relation

$$f_n = c_1 f_{n-1} + c_2 f_{n-2} + \cdots c_d f_{n-d} + F(n)$$

and suppose that the function F(n) has the form $F(n) = p(n)s^n$, where p(n) is a polynomial of degree t and s is a constant.

- If s is not a root of the characteristic equation for the associated homogeneous recurrence relation, then there is a particular solution of the form

$$f_n = (d_t n^t + d_{t-1} n^{t-1} + \cdots + d_1 n + d_0)s^n$$

- If s is a root of the characteristic equation for the associated homogeneous recurrence relation of multiplicity m, then there is a particular solution of the form

$$f_n = n^m(d_t n^t + d_{t-1} n^{t-1} + \cdots + d_1 n + d_0)s^n$$

PARTICIPATION ACTIVITY 16.4.7: Determining the form of the particular solution.

Animation captions:

1. For recurrence $f_n = 5f_{n-1} - 6f_{n-2} + 3n^2 + 6$, the associated homogeneous recurrence has characteristic equation $x^2 - 5x + 6 = (x - 2)(x - 3) = 0$.
2. F(n) is a degree 2 polynomial times 5^n, and 5 is not a root of the characteristic polynomial.
3. The form of the particular solution is a degree 2 polynomial $(an^2 + bn + c)5^n$. The recurrence relation can be used to solve for constants a, b, and c.
4. For recurrence $f_n = 8f_{n-1} - 21f_{n-2} + 18f_{n-3} + 3^n$, the associated homogeneous recurrence has characteristic equation $x^3 - 8x^2 + 21x - 18 = (x - 2)(x - 3)^2 = 0$.
5. F(n) is a degree 0 polynomial (which is a constant) times 3^n. 3 is a root of the characteristic polynomial with multiplicity 2.
6. The form of the particular solution is $f_n^{(p)} = cn^2 3^n$. The recurrence relation can be used to solve for the constant c.

PARTICIPATION ACTIVITY 16.4.8: Determining the form of the particular solution.

Animation captions:

1. For recurrence $f_n = -6f_{n-1} + 7f_{n-2} + 3n^2 + 6$, the associated homogeneous recurrence has characteristic equation $x^2 + 6x - 7 = (x + 7)(x - 1) = 0$.
2. F(n) is a degree 2 polynomial times 1^n, and 1 is a root of the characteristic polynomial with multiplicity 1.
3. The form of the particular solution is n^1 times a degree 2 polynomial $an^2 + bn + c$. The recurrence relation can be used to solve for constants a, b, and c.
4. For recurrence $f_n = 6f_{n-1} - 8f_{n-2} + 6$, the associated homogeneous recurrence has characteristic equation $x^2 - 6x + 8 = (x - 2)(x - 4) = 0$.
5. F(n) is a degree 0 polynomial (which is a constant) times 1^n, and 1 is not a root of the characteristic polynomial.
6. The form of the particular solution is $f_n^{(p)} = c$. The recurrence relation can be used to solve for the constant c.

PARTICIPATION ACTIVITY 16.4.9: The form for the particular solution.

Match the form for a particular solution to its corresponding recurrence relation.

If unable to drag and drop, refresh the page.

$f_n^{(p)} = cn$ $f_n^{(p)} = n^2(an^2 + bn + c)$ $f_n^{(p)} = cn(-3)^n$ $f_n^{(p)} = (an + b)$

$f_n^{(p)} = (an + b)3^n$

$f_n = -f_{n-1} + 5f_{n-2} - 3f_{n-3} + n^2$
Characteristic equation:
$(x - 1)^2(x + 3) = 0$

$f_n = -2f_{n-1} + 3f_{n-2} + 7$
Characteristic equation:
$(x - 1)(x + 3) = 0$

$f_n = -2f_{n-1} + 3f_{n-2} + 5n \cdot 3^n$

Characteristic equation:
$(x - 1)(x + 3) = 0$

$f_n = -2f_{n-1} + 3f_{n-2} + (-3)^n$
Characteristic equation:
$(x - 1)(x + 3) = 0$

$f_n = 7f_{n-1} - 12f_{n-2} + 4n$
Characteristic equation:
$(x - 3)(x - 4) = 0$

Reset

Additional exercises

EXERCISE | 16.4.1: The form of the particular solution.

Give the form for the particular solution for each recurrence relation. You do not have to solve for the coefficients. For example, your answer can have the form $3^n(an^2 + bn + c)$.

(a) $f_n = 8f_{n-1} - 5f_{n-2} - 50f_{n-3} + 7^n n^2$

(b) $f_n = 8f_{n-1} - 5f_{n-2} - 50f_{n-3} + 5^n n$

(c) $f_n = 8f_{n-1} - 5f_{n-2} - 50f_{n-3} + 2^n$

(d) $f_n = 8f_{n-1} - 5f_{n-2} - 50f_{n-3} + 3n^2 - 4$

(e) $f_n = 8f_{n-1} - 5f_{n-2} - 50f_{n-3} + 17n$

(f) $f_n = -6f_{n-1} + 7f_{n-2} + 3^n$

(g) $f_n = -6f_{n-1} + 7f_{n-2} + 5n^2$

(h) $f_n = -6f_{n-1} + 7f_{n-2} + 2$

EXERCISE 16.4.2: Solving linear non-homogeneous recurrence relations.

Solve the recurrence relation.

(a) $f_n = 8f_{n-1} - 15f_{n-2} + 2$
$f_0 = 21/4$
$f_1 = 85/4$

(b) $f_n = -f_{n-1} + 6f_{n-2} + 4n$
$f_0 = \frac{1}{4}$
$f_1 = -\frac{11}{4}$

(c) $f_n = 8f_{n-1} - 15f_{n-2} + 3n^2 - 15/8$
$f_0 = 4$
$f_1 = 7/16$

(d) $f_n = -2f_{n-1} + 3f_{n-2} + 5n - 3$
$f_0 = 3$
$f_1 = -57/16$

(e) $f_n = -f_{n-1} + 5f_{n-2} - 3f_{n-3} + 8$
$f_0 = 5$
$f_1 = -7$
$f_2 = 15$

(f) $f_n = 5f_{n-1} + 14f_{n-2} + 3^n$
$f_0 = 71/20$
$f_1 = -7/20$

(g) $f_n = f_{n-1} + 6f_{n-2} + 3 \cdot (-2)^n$
$f_0 = 10$
$f_1 = -12/5$

(h) $f_n = 4f_{n-1} - 4f_{n-2} + 3 \cdot 2^n$
$f_0 = 3$
$f_1 = 31/4$

(i) $f_n = 2f_{n-1} + 3f_{n-2} + 4n(-1)^n$
$f_0 = 5$

$f_1 = 5/4$

(j) $f_n = -f_{n-1} + 6f_{n-2} + 3n(-2)^n$
$f_0 = -11/2$
$f_1 = 22$

(k) $f_n = f_{n-1} + 2f_{n-2} + n^2(-2)^n$
$f_0 = 10$
$f_1 = -3$

(l) $f_n = -f_{n-1} + 2f_{n-2} + (4n^2 + 4n) \cdot 2^n$
$f_0 = 13$
$f_1 = 15$

16.5 Divide-and-conquer recurrence relations

This section has been set as optional by your instructor.

Deriving solutions to divide-and-conquer recurrence relations

Many recurrence relations that describe the time complexity of divide-and-conquer algorithms have the form

$$T(n) = aT(n/b) + \Theta(n^d)$$

On a call to the algorithm with input of size n, $\Theta(n^d)$ operations are performed outside the recursive calls. In addition, there are a recursive calls made on inputs of size n/b. For example, the recurrence relation for mergesort is $T(n) = 2T(n/2) + \Theta(n)$. Mergesort makes 2 recursive calls to sub-lists of size n/2 and then performs $\Theta(n)$ work in merging the two sorted sublists.

The animation below illustrates the total amount work done by a divide-and-conquer algorithm whose recurrence relation is $T(n) = 3T(n/2) + n^d$ for some constant d. The initial value for T is T(1) = 1. The analysis ignores the complication of having to round n/2 to $\lfloor n/2 \rfloor$ or $\lceil n/2 \rceil$ and simply uses the term n/2 to represent either possibility. The simplifications does not affect the asymptotic growth of the explicit formula for T(n).

The analysis proceeds by expanding the terms using the recurrence relation: Every expression T(m) is replaced by three $T(m/2)$ terms and one m^d term because $T(m) = 3T(m/2) + m^d$. The expansion continues until the initial value T(1) is reached.

PARTICIPATION ACTIVITY | 16.5.1: Expanding the terms in a divide-and-conquer recurrence relation: Number of levels.

Animation captions:

1. The value of T(n) is the sum of four terms: n^d + T(n/2) + T(n/2) + T(n/2). At level 0, replace T(n) by the sum of the four terms.
2. At level 1, the fact that $T(n/2) = 3T(n/4) + (n/2)^d$, can be used to replace each of the T(n/2) terms in the expression for T(n) with four terms: $(n/2)^d$ + T(n/4) + T(n/4) + T(n/4).
3. Keep using the recurrence relation to replace each $T(\cdot)$ term until the last level, when the input to T is $n/2^L = 1$. Then apply the initial condition T(1) = 1.
4. The number of levels is L+1, where $n/2^L = 1$. Solving for L gives that $L = \log_2 n$.

PARTICIPATION ACTIVITY | 16.5.2: Expanding the terms in a divide-and-conquer recurrence relation: Evaluating the sum.

Animation captions:

1. The value of T(n) is the sum of all the terms in the boxes.
2. There is $1 = 3^0$ term at level 0. The value of the term is $n^d = (n/2^0)^d$. Add $3^0 \cdot (n/2^0)^d$ to the sum for T(n).
3. There are $3 = 3^1$ terms at level 1. The value of each term is $(n/2)^d = (n/2^1)^d$. Add $3^1 \cdot (n/2^1)^d$ to the sum for T(n).
4. There are $9 = 3^2$ terms at level 2. The value of each term is $(n/4)^d = (n/2^2)^d$. Add $3^2 \cdot (n/2^2)^d$ to the sum for T(n).
5. Continue until the last level L, where $n/2^L = 1$. There are 3^L terms at level L. The value of each term is $1 = (n/2^L)^d$. Therefore, $3^L \cdot (n/2^L)^d$ is the last term in the sum for T(n).

PARTICIPATION ACTIVITY | 16.5.3: The sum of terms at level j in the expansion.

The questions below refer to the expansion of the recurrence relation $T(n) = 3T(n/2) + n^d$ shown in the animations above.

1) If the number of terms at level j is m, then how many terms are at level j+1?

- m/2
- m/3
- 2m
- $3n^2$

2) If there is 1 term at level 0, then how many terms are at level j?

- a^j
- 2^j
- 3^j

3) If the value of each term at level j is s^d, then what is the value of each term at level j+1?

- $(s/2)^d$
- $(s/3)^d$
- $(2s)^d$
- $(3s)^d$

4) If the term at level 0 is n^d, then what is the value of each term at level j?

- $(n/2^j)^d$
- $(n/3^j)^d$
- $(3^j)^d$

5) What is the sum of the values of all the terms at level j?

- $2^j(n/3^j)^d$
- $3^j(n/2^j)^d$
- n^d

Expressing the formula as a geometric sum

The function T described by recurrence relation $T(n) = 3T(n/2) + n^d$ and initial value T(1) = 1 has an explicit formula that is a sum:

$$T(n) = \sum_{j=0}^{L} 3^j \left(\frac{n}{2^j}\right)^d \quad \text{where } L = \log_2 n$$

The sum can be simplified as

$$T(n) = \sum_{j=0}^{L} 3^j \left(\frac{n}{2^j}\right)^d = \sum_{j=0}^{L} 3^j \frac{n^d}{2^{jd}} = \sum_{j=0}^{L} n^d \left(\frac{3}{2^d}\right)^j = n^d \cdot \sum_{j=0}^{L} \left(\frac{3}{2^d}\right)^j$$

The explicit formula for T(n) can be generalized for a recurrence relation of the form $T(n) = aT(n/b) + n^d$ which has an explicit formula

$$T(n) = n^d \cdot \sum_{j=0}^{L} \left(\frac{a}{b^d}\right)^j \quad \text{where } L = \log_b n$$

The expression for T(n) is n^d times the sum of terms in a geometric sequence of the form $1 + r + r^2 + \cdots + r^L$, where the ratio $r = a/b^d$. The formula for the sum of a geometric sequence is

$$1 + r + r^2 + \cdots + r^L = \sum_{j=0}^{L} r^j = \frac{r^{L+1} - 1}{r - 1}$$

The asymptotic growth rate of T(n) depends on whether the ratio r is less than, greater than, or equal to 1.

PARTICIPATION ACTIVITY 16.5.4: Asymptotic growth of the sum of a geometric sequence.

Animation captions:

1. If $r > 1$, then the dominant term in the sum $1 + r + r^2 + \cdots + r^L$ is r^L. The value of the sum is $(r^{L+1} - 1)/(r - 1)$, which is $\Theta(r^L)$.
2. When r = 1, all the terms in the sum are equal to 1. There are L+1 terms, so the value of the sum is L+1, which is $\Theta(L)$.
3. If $r < 1$, then the dominant term in the sum $1 + r + r^2 + \cdots + r^L$ is 1. The value of the sum is $(r^{L+1} - 1)/(r - r)$, which is $\Theta(1)$.

PARTICIPATION ACTIVITY 16.5.5: Determining the growth rate of the sum of a geometric sequence.

Match each sum to the correct asymptotic growth rate.

If unable to drag and drop, refresh the page.

$\sum_{j=0}^{L} \left(\frac{1}{3}\right)^j$ $\sum_{j=0}^{L} 3^j$ $\sum_{j=0}^{L} 1^j$

$\Theta(3^L)$

$\Theta(L)$

$\Theta(1)$

Reset

PARTICIPATION ACTIVITY 16.5.6: The asymptotic growth of divide-and-conquer recurrence relations.

1) Select the correct asymptotic growth rate for the expression

$$n^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$

for a = 4, b = 2, and d = 2.

- $\Theta(\log n)$
- $\Theta(n^2)$
- $\Theta(n^3)$
- $\Theta(n^2 \log n)$

2) Select the correct asymptotic growth rate for the expression

$$n^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$

for a = 4, b = 2, and d = 3.

- $\Theta(n^3)$
- $\Theta(n^3 \log n)$
- $\Theta(n^4)$
- $\Theta(n^4 \log n)$

3) Select the correct asymptotic growth

rate for the expression

$$n^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$

for a = 8, b = 2, and d = 1.

- $n^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$
- $\Theta\left(n \cdot \left(\frac{3}{2}\right)^{\log_2 n}\right)$
- $\Theta\left(n \cdot \left(\frac{3}{2}\right)^{n}\right)$
- $\Theta(n)$
- $\Theta(n \log n)$

4) Select the expression that is equivalent to

$$n\left(\frac{3}{2}\right)^{\log_2 n}$$

You will need the following two laws of logarithms:
For $a > 0$ and $b > 1$,
$a^{\log_b n} = n^{\log_b a}$
$\log_b b = 1$

- $n^{\log_2 3}$
- $n^{\log_2 3 - 1}$
- $n^{\log_2 3 + 1}$

The Master Theorem

The final closed form solution to a divide-and-conquer recurrence relation of the form $T(n) = aT(n/b) + \Theta(n^d)$ is summarized in the Master Theorem which can be used to find the asymptotic time complexity for most divide-and-conquer algorithms. The initial values for the recurrence are always assumed to be $\Theta(1)$.

Although the analysis above used the simpler recurrence $T(n) = aT(n/b) + n^d$ instead of $T(n) = aT(n/b) + \Theta(n^d)$ and the simpler initial value T(1) = 1, instead of T(1) = O(1), the asymptotic

growth rate of T is the same.

Theorem 16.5.1: The Master Theorem.

If $T(n) = aT(n/b) + \Theta(n^d)$ for constants $a > 0$, $b > 1$ and $d \geq 0$, then

- If $(a/b^d) < 1$, then $T(n) = \Theta(n^d)$
- If $(a/b^d) = 1$, then $T(n) = \Theta(n^d \log n)$
- If $(a/b^d) > 1$, then $T(n) = \Theta(n^{\log_b a})$

PARTICIPATION ACTIVITY 16.5.7: Applying the Master Theorem: An example for each case.

Animation captions:

1. The recurrence relation for binary search is $T(n) = T(n/2) + \Theta(1)$. a = 1, b = 2, and d = 0. Since $(a/b^d) = 1$, $T(n) = \Theta(n^d \log n) = \Theta(\log n)$.
2. For recurrence relation, $T(n) = 4T(n/2) + \Theta(n)$. a = 4, b = 2, and d = 1. Since $(a/b^d) > 1$, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 4}) = \Theta(n^2)$.
3. For recurrence relation, $T(n) = 4T(n/3) + \Theta(n^2)$. a = 4, b = 3, and d = 2. Since $(a/b^d) < 1$, $T(n) = \Theta(n^d) = \Theta(n^2)$.

PARTICIPATION ACTIVITY 16.5.8: Applying the Master Theorem to divide-and-conquer recurrence relations.

Match each recurrence relation with its corresponding growth rate.

If unable to drag and drop, refresh the page.

$\Theta(n^{\log_3 2})$ $\Theta(n^{\log_2 7})$ $\Theta(n)$ $\Theta(n^3)$ $\Theta(n \log n)$

$T(n) = 3T(n/3) + \Theta(n)$

$T(n) = T(n/3) + \Theta(n)$

$T(n) = 7T(n/2) + \Theta(n^2)$

$$T(n) = 7T(n/2) + \Theta(n^3)$$

$$T(n) = 2T(n/3) + \Theta(1)$$

Reset

Additional exercises

EXERCISE | 16.5.1: Exact solutions for divide-and-conquer recurrence relations.

Expand the terms of each recurrence relation in order to obtain an exact solution for T(n). Your solution should include all the constants in the expression for T(n), and not just the asymptotic growth of the function T(n). You can assume that the value of n, the input to the function T, is a power of 3. That is, $n = 3^k$ for some integer k.

(a) $T(n) = 3T(n/3) + 5n$

$T(1) = 5$

(b) $T(n) = 3T(n/3) + 5n^2$

$T(1) = 5$

(c) $T(n) = 9T(n/3) + 5n$

$T(1) = 5$

EXERCISE | 16.5.2: Applying the Master Theorem.

Give the asymptotic growth of T(n) using Θ notation.

(a) $T(n) = 4T(n/3) + \Theta(n)$

(b) $T(n) = 4T(n/4) + \Theta(\sqrt{n})$

(c) $T(n) = 4T(n/2) + \Theta(n^2)$

(d) $T(n) = 4T(n/2) + \Theta(n^3)$

(e) $T(n) = 2T(n/3) + \Theta(n)$

(f) $T(n) = 2T(n/3) + \Theta(1)$

(g) $T(n) = 7T(n/4) + \Theta(n^2)$

(h) $T(n) = 7T(n/4) + \Theta(n)$

(i) $T(n) = 2T(n/4) + \Theta(\sqrt{n})$

(j) $T(n) = 3T(n/3) + \Theta(1)$

©zyBooks 12/15/22 00:33 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

EXERCISE 16.5.3: Calculating the asymptotic growth rate - step by step.

Consider the recurrence relation $T(n) = 5\ T(n/2) + 3n^3$. Assume that n is a power of 2, that is, $n = 2^k$ for some integer k.

(a) If the levels of an expansion of the recurrence relation are numbered 0 through L, then what is the value of L? Express your answer in terms of n.

(b) If the levels of an expansion of the recurrence relation are numbered 0 through L, then how many terms are there at level j?

(c) If the levels of an expansion of the recurrence relation are numbered 0 through L, then what is the value of each term at level j? Express your answer in terms of j and n.

(d) If the levels of an expansion of the recurrence relation are numbered 0 through L, then what is the sum of all the values at level j? Express your answer in terms of j and n.

(e) Give a summation expression that is equivalent to the sum of all the terms in the expansion of T(n).

(f) Express the asymptotic growth rate for the value of the sum in the previous answer.

©zyBooks 12/15/22 00:33 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

EXERCISE 16.5.4: Applying the Master Theorem to expressions.

Give the asymptotic growth growth rate of the expression. You should express your answer in the form $\Theta(n^c)$ or $\Theta(n^c \log n)$ for some constant c

(a) $\sum_{j=0}^{\log_b n} a^j \left(\frac{n}{b^j}\right)^d$ for a = 8, b = 2 and d = 3

(b) $\sum_{j=0}^{\log_b n} a^j \left(\frac{n}{b^j}\right)^d$ for a = 7, b = 2 and d = 3

(c) $\sum_{j=0}^{\log_b n} a^j \left(\frac{n}{b^j}\right)^d$ for a = 7, b = 2 and d = 5

(d) $\sum_{j=0}^{\log_b n} a^j \left(\frac{n}{b^j}\right)^d$ for a = 7, b = 2 and d = 1

(e) $\sum_{j=0}^{\log_b n} a^j \left(\frac{n}{b^j}\right)^d$ for a = 1, b = 2 and d = 1

©zyBooks 12/15/22 00:33 1361995
John Farrell
COLOSTATECS220SeaboltFall2022