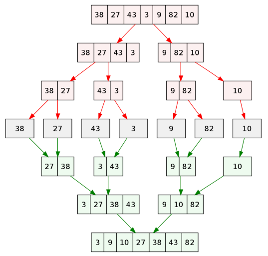


Divide and Conquer
Lecture 6

COSC 242 – Algorithms and Data Structures

Today’s outline

1. Divide and conquer
2. Binary search
3. Binary search analysis
4. Merge
5. Mergesort

Types of algorithms

In COSC242, we will be looking at 3 general types of algorithms:

1. Divide-and-conquer algorithms
2. Greedy algorithms
3. Dynamic programming algorithms

Each type of algorithm can be used to naturally, or more easily, solve a particular type of problem.

It is useful to keep a list of the typical problems that a given type of algorithm is good for.

Incremental approach

In L01 we looked at Insertion sort. This algorithm applied an incremental approach:

Having already sort A[1..j-1], we insert the single element A[j] into its proper place, yielding the sorted subarray A[1..j].

Today we will look at a different approach, known as “divide and conquer”.

Divide and conquer

Many useful algorithms are **recursive** in structure. To solve a given problem, they call themselves recursively one or more times to deal with closely related smaller problems.

These algorithms typically follow a **divide-and-conquer** approach.

Divide-and-conquer algorithms usually work on sequential data structures of known size. Thus, they are commonly used when working with *arrays*.

Divide, then conquer. But first lets divide…

The divide-and-conquer paradigm involves three steps at each level of the recursion:

1. **Divide** – the problem into a number of subproblems that are smaller instances of the same problem.
2. **Conquer** – the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
3. **Combine** - the solutions to the subproblems into the solution for the original problem.

Divide, then conquer. But first lets divide…

Divide-and-conquer processes the data structure X as:

```
if X is an atom then
    process X directly
else
    divide X into two or more smaller pieces
    apply the algorithm to each piece “recursively”
    combine the processed pieces (if necessary)
```

Looking up a name

Lets say you’re looking for company in the phone book. We’ll call them the “Max’s mini donuts”.

How would you go about finding them, assuming for a moment there’s no search function?

Would you start at “AAA aardvarks” and work your way forward, one entry at a time? Or would you start in the middle, as “M” is not too far from the middle.

Discussion: Guessing game

Your friend asked you to guess a number between 1 and 100. You have 7 tries, and they will only respond with: correct, higher, or lower.

Questions

What number would you start with?

Can you win in 7 tries?

Would this work for pulling numbers out of a jar? Why, why not?

Binary search

The strategy we've just seen is an example of **binary search**. It's an efficient algorithm for finding an item in a *sorted list*.

Consider an array A[0..n-1] of sorted keys, and suppose we want to locate a target value x. The simplest way to search is sequentially, but sequential search is linear: $\mathrm{O(n)}$.

In binary search, we find the index $m = \lfloor (n-1)/2 \rfloor$ of the middle element, then compare x with $\mathrm{A[m]}$.

Binary search

There are 4 possibilities:

If $x = A[m]$, we have found x.

If $x < A[m]$, we have the smaller array $A[0..m\text{-}1]$

If $x > A[m]$, we search the smaller array $A[m+1..n\text{-}1]$

If the breaking down process ever gives an empty array, we've gone too far and can stop.

A	0				mid					n-1

Binary search pseudocode

```
Binary_search(A, x, low, high):
if low > high then
    report failure and stop
else
    mid ← (low + high) / 2
    if x = A[mid] then
        report success and return mid
    else if x < A[mid] then
        return Binary_search(A, x, low, mid – 1)
    else if x > A[mid] then
        return Binary_search(A, x, mid+1, high)
```

Class challenge 1

Activity Time.

Questions

1. Write the function call execute a binary search on A, below, searching for number 31.

2. Trace the operations of the binary search.

A

8	13	20	21	22	31	33	39	42	55

Binary search analysis

To look for x in an array A of length n, you would call $\mathrm{Binary\_search}(A, x, 0, n - 1)$

Analysis: How many times (in the worst case) can we divide the length n in half? Try it for 8, 16, 32, 64.

2^n = 2x2x2x…2. How many times can I divide that by 2?

Complexity function is f =

Merge two arrays

We have two arrays X and Y each in sorted order. We want to build array Z containing all the keys of X and Y in sorted order. Length(X) = l, length(Y) = m:

```
initialise i, j, k to 0 (i, j, k index the arrays X, Y, Z)
 while i < l and j < m do
     if X[i] < Y[j] then
         Z[k] ← X[i]
         i ← i + 1
     else if X[i] ≥ Y[j] then
         Z[k] ← Y[j]
         j ← j + 1
     k ← k + 1
 if i ≥ l then copy the end of Y to the end of Z
 else copy the end of X to the end of Z
```

Merge analysis

How many times through the merge while loop?

One thing we notice is that i, j, k all start at 0. Each time through the loop k is incremented and either i or j is incremented. Neither i, j or k ever gets smaller.

At the end of the loop:

either ($i = l$ and $j < m$) or ($j = m$ and $i < l$),

k is just the sum of i and j, so $k < l + m$.

Merge analysis

Since k is incremented every time through the loop, the number of times through the loop is less than $l+m$. Then, whichever array has not been fully scanned is copied onto the end of Z.

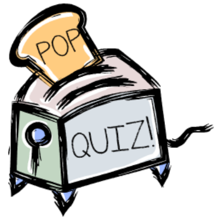
So the number of operations is some constant times $l+m$. Let $n = l+m$, so Merge is $O(n)$. Or more specifically, it's $\boldsymbol{\Theta}(n)$.

Mergesort

To mergesort an array A[0 .. n - 1] of keys, we repeatedly split A, and after getting to the bottom we rebuild by merging the pieces. To identify the pieces that must be split or patched together, we use indices *left* and *right*.

```
Mergesort(A, left, right) // sorts the keys in A[left .. right]
if left ≥ right then
   stop since A[left .. right] is sorted
else
    mid ← (left + right) / 2
    Mergesort(A, left, mid)
    Mergesort(A, mid + 1, right)
    Merge subarrays A[left .. mid] and A[mid + 1 .. right]
```

Pop quiz 1



What is the primary way in which binary search and mergesort algorithms are related?

Mergesort analysis

Let's call the time complexity function T.

If $\mathrm{n} = 1$, then mergesort takes constant time, so $\mathrm{T}(1) = 1$.

Otherwise the number of operations needed to mergesort *n* keys is equal to the number of operations needed to do two mergesorts of size $\mathrm{n}/2$ (the recursive calls), plus the merge needed to patch the two sorted arrays of length $\mathrm{n}/2$ together (which is linear, shown previously).

So $\mathrm{T}(\mathrm{n}) = 2\mathrm{T}(\mathrm{n}/2) + \mathrm{n}$.

We'll see how to analyse these sorts of complexity functions in the next lecture.

Suggested reading

Divide and conquer algorithms are discussed in section 2.3 of the textbook, including a look at mergesort and its analysis*.

*Section numbers refer to those in the 3rd edition.

Solutions

Discussion: Guessing game

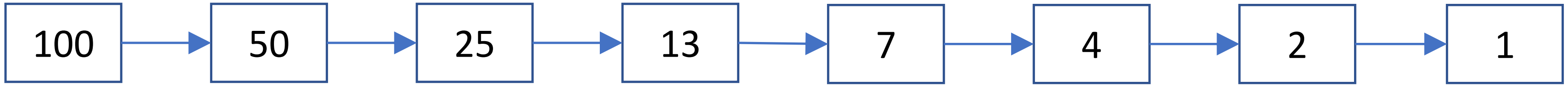
Your friend asked you to guess a number between 1 and 100. You have 7 tries, and they will only respond with: correct, higher, or lower.

Questions

What number would you start with?

Can you win in 7 tries?

Would this work for pulling numbers out of a jar? Why, why not?



7 guesses

Class challenge 1

Binary_search(A, 31, 0, 9)

1. Mid = $\lfloor 0 + (9) \rfloor / 2 = 4$. x != 22, and x > 22

A	8	13	20	21	22	31	33	39	42	55
	0	1	2	3	4	5	6	7	8	9

Class challenge 1

Binary_search(A, 31, 0, 9)

1. Mid = $\lfloor 0 + (9) \rfloor / 2 = 4$. x != 22, and x > 22
2. Low = mid + 1 = 5. Mid = $\lfloor 5 + (9) \rfloor / 2 = 7$. x != 39, and x < 39

A

0	1	2	3	4	5	6	7	8	9
8	13	20	21	22	31	33	39	42	55

Class challenge 1

Activity Time.

Binary_search(A, 31, 0, 9)

1. Mid = $\lfloor 0 + (9) \rfloor / 2 = 4$. x != 22, and x > 22
2. Low = mid + 1 = 5. Mid = $\lfloor 5 + (9) \rfloor / 2 = 7$. x != 39, and x < 39
3. High = mid – 1 = 6. Mid = $\lfloor 5 + (7) \rfloor / 2 = 6$. x != 6, and x < 33.

A	8	13	20	21	22	31	33	39	42	55
	0	1	2	3	4	5	6	7	8	9

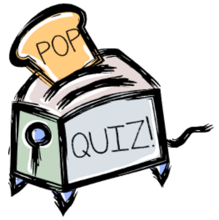
Class challenge 1

Binary_search(A, 31, 0, 9)

1. Mid = $\lfloor 0 + (9) \rfloor / 2 = 4$. x != 22, and x > 22
2. Low = mid + 1 = 5. Mid = $\lfloor 5 + (9) \rfloor / 2 = 7$. x != 39, and x < 39
3. High = mid – 1 = 6. Mid = $\lfloor 5 + (7) \rfloor / 2 = 6$. x != 6, and x < 33.
4. High = mid – 1 = 5. x == 5, report success and return 5.

A	8	13	20	21	22	31	33	39	42	55
	0	1	2	3	4	5	6	7	8	9

Pop quiz 1



What is the primary way in which binary search and mergesort algorithms are related?

Answers

Use of a divide and conquer, which entails recursion

Image attributions

This Photo by Unknown Author is licensed under CC0

This Photo by Unknown Author is licensed under CC BY

Disclaimer: Images and attribution text provided by PowerPoint search. The author has no connection with, nor endorses, the attributed parties and/or websites listed above.