SCC0602 - Algoritmos e Estruturas de Dados I

Recurrence

Professor: André C. P. L. F. de Carvalho, ICMC-USP
PAE: Rafael Martins D'Addio
Monitor: Joao Pedro Rodrigues Mattos

Today

- Defining bounds
- The Divide-and-conquer technique for algorithm design
- Example problems:
  - Tromino puzzle
  - Searching (*binary search*)
  - Sorting (*merge sort*)

Defining bounds

$\Theta(g(n)) = \{f(n) : \exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$

Find n_0, c_1 and c_2 such that

$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

Choose $\begin{cases} n_0 \geq 1 \\ c_1 > 0 \\ c_2 > 0 \end{cases}$

Which value of n_0 makes the inequality true?

Defining bounds

$\Theta(g(n)) = \{f(n) : \exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$

Find n_0, c_1 and c_2 such that

$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

Choose $\begin{cases} n_0 \geq 1 \\ c_1 > 0 \\ c_2 > 0 \end{cases}$

Which value of n_0 makes the inequality true? 1?

Defining bounds

$\Theta(g(n)) = \{f(n) : \exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$

Find n_0, c_1 and c_2 such that

$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

Choose $\begin{cases} n_0 \geq 1 \\ c_1 > 0 \\ c_2 > 0 \end{cases}$

Which value of n_0 makes the inequality true? 2?

Defining bounds

$\Theta(g(n)) = \{f(n) : \exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$

Find n_0, c_1 and c_2 such that

$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

Choose $\begin{cases} n_0 \geq 1 \\ c_1 > 0 \\ c_2 > 0 \end{cases}$

Which value of n_0 makes the inequality true? 6?

Defining bounds

$\Theta(g(n)) = \{f(n) : \exists$ **positive constants** $c_1, c_2,$ **and** $n_0,$ **such that** $\forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$

Find n_0, c_1 and c_2 such that

$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

Choose $\begin{cases} n_0 \geq 1 \\ c_1 > 0 \\ c_2 > 0 \end{cases}$

Which value of n_0 makes the inequality true? 7?

Defining bounds

$\Theta(g(n)) = \{f(n) : \exists$ **positive constants** $c_1, c_2,$ **and** $n_0,$ **such that** $\forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$

Find n_0, c_1 and c_2 such that

$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

Choose $\begin{cases} n_0 = 7 \\ c_1 > 0 \\ c_2 > 0 \end{cases}$

Which values of c_1 and c_1 make the inequality true?

Defining bounds

$\Theta(g(n)) = \{f(n) : \exists$ **positive constants** $c_1, c_2,$ **and** $n_0,$ **such that** $\forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$

Find n_0, c_1 and c_2 such that

$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

Choose $\begin{cases} n_0 = 7 \\ c_1 = \frac{1}{14} \\ c_2 > 0 \end{cases}$

Which values of c_1 and c_1 make the inequality true?

Defining bounds

$\Theta(g(n)) = \{f(n) : \exists$ **positive constants** $c_1, c_2,$ **and** $n_0,$ **such that** $\forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$

Find n_0, c_1 and c_2 such that

$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

Choose $\begin{cases} n_0 = 7 \\ c_1 = \frac{1}{14} \\ c_2 = 1 \end{cases}$

Which values of c_1 and c_1 make the inequality true?

Exercise

$\Theta(g(n)) = \{f(n) : \exists$ **positive constants** $c_1, c_2,$ **and** $n_0,$ **such that** $\forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = 3n^3 - 4n + 5 = \Theta(n^3)$

Find n_0, c_1 and c_2 such that

$c_1 n^3 \leq 3n^3 - 4n + 5 \leq c_2 n^3$

Choose $\begin{cases} n_0 \geq 1 \\ c_1 > 0 \\ c_2 > 0 \end{cases}$

Which value of n_0, c_1 and c_1 Make the inequality true?

Exercise

$\Theta(g(n)) = \{f(n) : \exists$ **positive constants** $c_1, c_2,$ **and** $n_0,$ **such that** $\forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = 3n^3 - 4n + 5 = \Theta(n^3)$

Find n_0, c_1 and c_2 t

$c_1 n^3 \leq 3n^3 - 4n + 5 \leq c_2 n^3$

$c_1 \leq 3 - \frac{4}{n^2} + \frac{5}{n^3} \leq c_2$

Choose $\begin{cases} n_0 \geq 1 \\ c_1 > 0 \\ c_2 > 0 \end{cases}$

Which value of n_0, c_1 and c_1 Make the inequality true?

Exercise

$\Theta(g(n)) = \{f(n) : \exists$ **positive constants** c_1, c_2, **and** n_0, **such that** $\forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = an^2 + bn + c = \Theta(n^2)$

Exercise

$\Theta(g(n)) = \{f(n) : \exists$ **positive constants** c_1, c_2, **and** n_0, **such that** $\forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

Show that the runtime $f(n) = an^2 + bn + c = \Theta(n^2)$

If any of a, b, and c are less than 0 replace the constant with its absolute value

$an^2 + bn + c \leq (a+b+c)n^2 + (a+b+c)n + (a+b+c)$

$\leq 3(a + b + c)n^2$ for $n \geq 1$

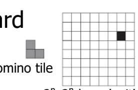
Let $c' = 3(a + b + c)$ and let $n_0 = 1$

Exercise

- Show that insertion sort is $O(n^2)$
- Is insertion sort $O(n^3)$?

Tromino puzzle

- Tromino: group of 3 squares with an L shape
- Board: *mxm* array of squares, where $m = 2^n$
  - There is one forbidden tile (hole) in the board
- Goal: tiling (filling) of the board
  - All squares are covered
    - Apart from the forbidden tile
  - Trominoes do not overlap
  - All trominoes are completely inside the board



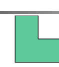
Tromino tile

$2^n x 2^n$ board with a hole

Tromino puzzle

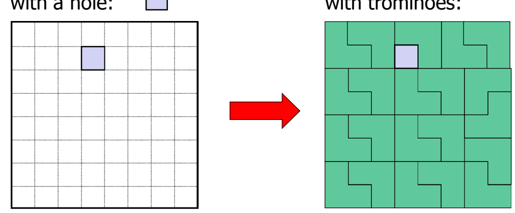
- How to play the 8-by-8 Tromino Puzzle
  - Place the square tile on one of the 64 square grid cells
  - Repeat
    - Place the L-shaped trominoes into a position one at a time
  - Until there is no empty cell

Tiling

Tromino tile: 

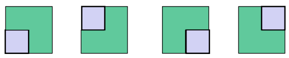
$2^n x 2^n$ board with a hole: 

A tiling of the board with trominoes:



Tiling: Trivial Case (n = 1)

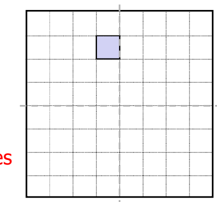
- Trivial case (n = 1): tiling a 2x2 board with a hole:



- Suggestion:
  - Try to reduce the size of the original problem until get to 2x2 boards
    - which are easy to solve…

Tiling: Dividing the Problem

- To get smaller square boards, the original board can be divided into four boards
  - One of the problems has size $2^{n-1}x2^{n-1}$
  - But the other three are different from the original problem
    - They do not have holes
    - What should we do?



Tiling: Algorithm

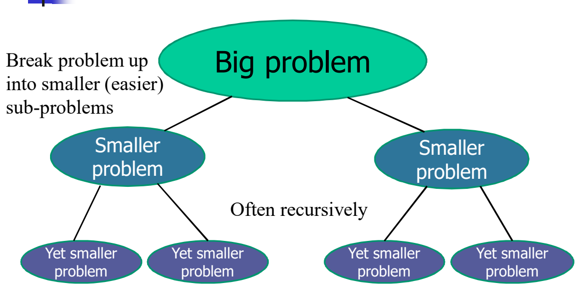
```
INPUT: n – the board size (2^n x2^n board), L – location of the hole
OUTPUT: tiling of the board

Tile(n, L)
  if n = 1 then
    Trivial case
    Tile with one tromino
    return
  Divide the board into four equal-sized boards
  Place one tromino at the center to simulate 3 additional holes
  Let L1, L2, L3, L4 denote the positions of the 4 holes
  Tile(n-1, L1)
  Tile(n-1, L2)
  Tile(n-1, L3)
  Tile(n-1, L4)
```


Divide and Conquer

- *Divide-and-conquer* method for algorithm design:
  - If the problem size is small enough to be directly solved, solve it
  - Else:
    - **Divide**: Divide the problem into two or more disjoint *subproblems*
    - **Conquer**: Apply divide-and-conquer recursively to solve the subproblems
    - **Combine**: Combine the solutions to the subproblems into a solution to the original problem

Divide and conquer



Tiling: Divide-and-Conquer

- Tiling is a divide-and-conquer algorithm:
  - If the board is 2x2 , do it trivially
  - Else:
    - **Divide** the board into four smaller boards (introduce holes at the corners of three smaller boards to make them look like original problems)
    - **Conquer** using the same algorithm recursively
    - **Combine** by placing a single tromino in the center of the board to cover the three introduced holes

Binary Search

- Find a number in a sorted array:
  - If the array is of one element, just do it trivially
  - Else
    - **Divide** into two equal halves and **solve** each half
    - **Combine** the results

```
INPUT: A[1..n]: a sorted (non-decreasing) array of integers, s: an integer
OUTPUT: an index j such that A[j] = s.   NIL, if ∀j (1≤j≤n): A[j] ≠ s
Binary-search(A, l, r, s):
  if l = r then
    if A[l] = s then return l
    else return NIL
     q ←⌊(l+r)/2⌋
     ret  ←  Binary-search(A, l, q, s)
     if ret = NIL then
    return Binary-search(A, q+1, r, s)
  else return ret
```


Recurrences

- divide-and-conquer paradigm three steps:
  - Divide
  - Conquer
  - Combine
- Naturally solved by recurrence
  - A base case
  - A recursive case

Recurrences

- Base case
- Recursive case

Recurrences

- The three steps of the divide-and-conquer paradigm makes it recursive
- Large problems are solved with
  - A base case
  - A recursive case
- Fits very well with the divide-and-conquer paradigm
  - Provides a easy way to define th running time of divide-and-conquer algorithms

Recurrences

- **Recursive calls** in algorithms can be described using recurrences
- It is an equation or inequality that describes a function in terms of its value on smaller inputs

$$T(n) = \begin{cases} \text{solving\_trivial\_problem} & \text{if } n = 1 \\ \text{num\_pieces } T(n/\text{subproblem\_size\_factor}) + \text{dividing} + \text{combining} & \text{if } n > 1 \end{cases}$$

- Example: Binary search

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Binary Search (improved)

- T(n) = Θ(n) – not better than brute force!
- Clever way to **conquer**:
  - Solve only one half!

```
INPUT: A[1..n]: a sorted (non-decreasing) array of integers, s: an integer
OUTPUT: an index j such that A[j] = s. NIL, if ∀j (1≤j≤n): A[j] ≠ s
Binary-search(A, l, r, s):
  if l = r then
    if A[l] = s then return l
    else return NIL
     q ←⌊ (l+r)/2 ⌋
     if A[q] ≤ s then return Binary-search(A, l, q, s)
  else return Binary-search(A, q+1, r, s)
```

Running Time of Binary Search

- Can be expressed as a recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(n/2) + \Theta(1) & \text{if } n > 1 \end{cases}$$

$$T(n) = \Theta(\lg n)$$

Merge Sort: Algorithm

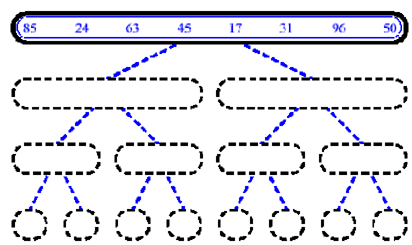
- **Divide**: If *S* has at least 2 elements (nothing needs to be done if *S* has 1 element)
  - Remove all the elements from *S* and put them into 2 sequences, S_1 and S_2
    - Each with half of the elements of S
      - S_1 contains the first $\lceil n/2 \rceil$ elements and S_2 contains the remaining $\lfloor n/2 \rfloor$ elements
- **Conquer**: Sort sequences S_1 and S_2 using Merge Sort
- **Combine**: Put the elements back into *S* by merging the sorted sequences S_1 and S_2 into one sorted sequence

Merge Sort Algorithm

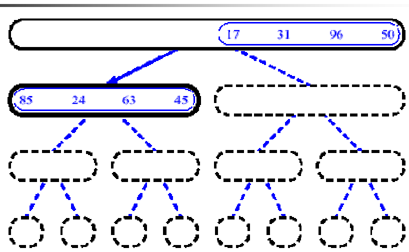
```
Merge-Sort(A, l, r)
   if l < r then
      q←(l+r)/2
      Merge-Sort(A, l, q)
      Merge-Sort(A, q+1, r)
      Merge(A, l, q, r)
```

```
Merge(A, l, q, r)
   Take the smallest of the two topmost elements of
sequences A[l..q] and A[q+1..r] and put into the
resulting sequence. Repeat this, until both sequences
are empty. Copy the resulting sequence into A[l..r].
```

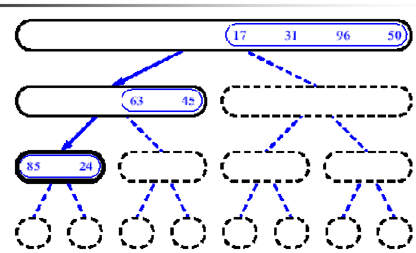

Merge Sort (Example) - 1



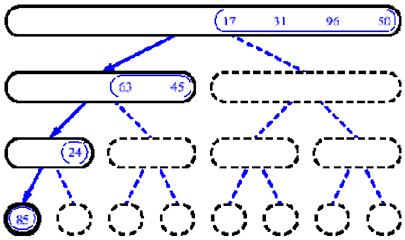
Merge Sort (Example) - 2



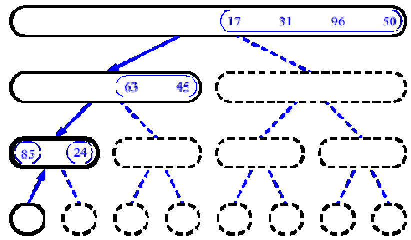
Merge Sort (Example) - 3



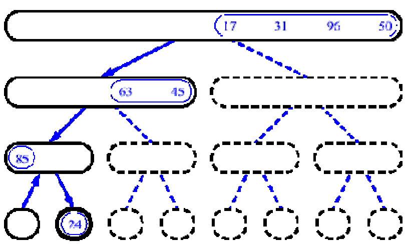
Merge Sort (Example) - 4



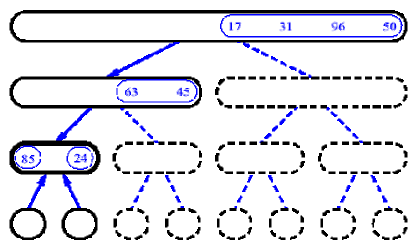
Merge Sort (Example) - 5



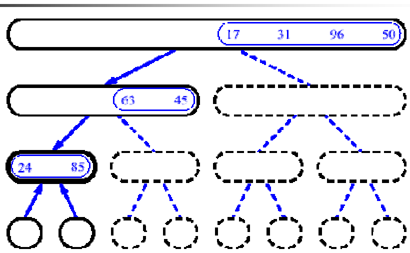
Merge Sort (Example) - 6



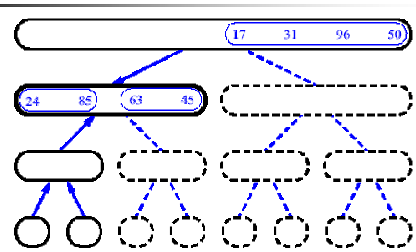
Merge Sort (Example) - 7



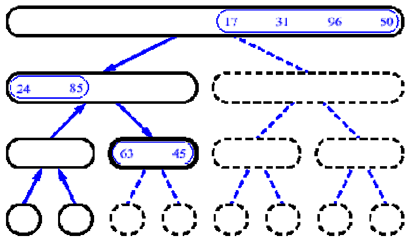
Merge Sort (Example) - 8



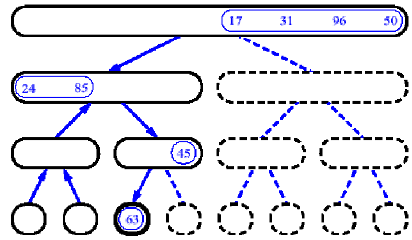
Merge Sort (Example) - 9



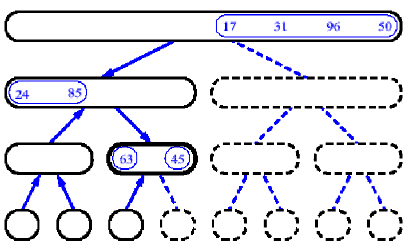
Merge Sort (Example) - 10



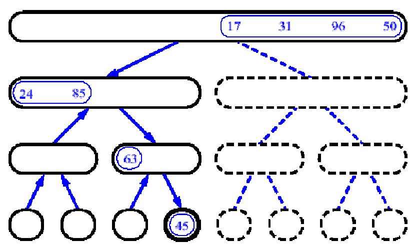
Merge Sort (Example) - 11



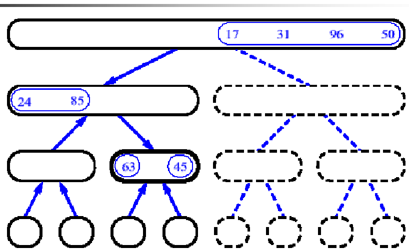
Merge Sort (Example) - 12



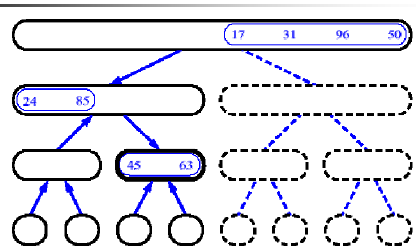
Merge Sort (Example) - 13



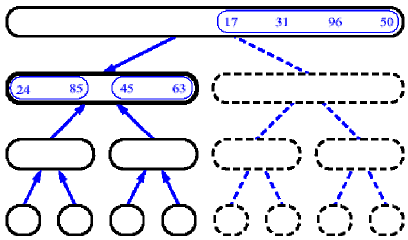
Merge Sort (Example) - 14



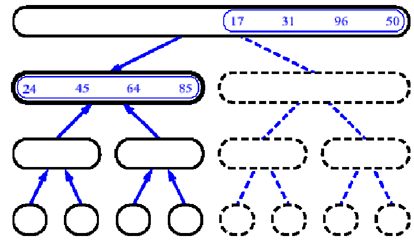
Merge Sort (Example) - 15



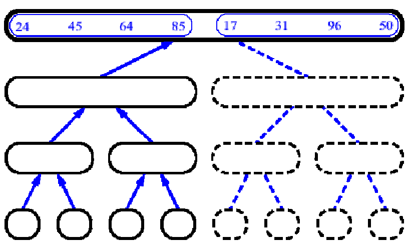
Merge Sort (Example) - 16



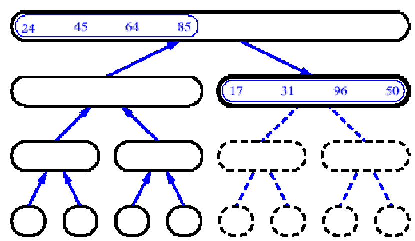
Merge Sort (Example) - 17



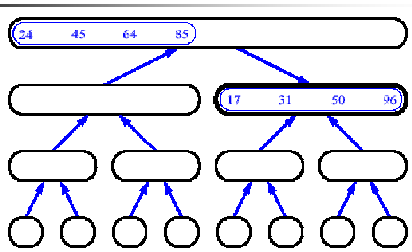
Merge Sort (Example) - 18



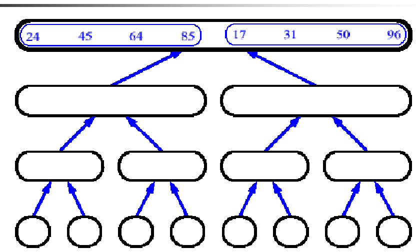
Merge Sort (Example) - 19



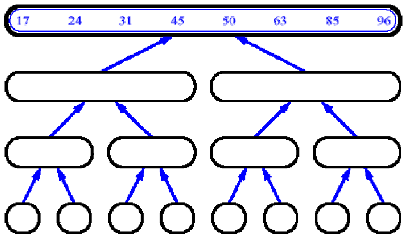
Merge Sort (Example) - 20



Merge Sort (Example) - 21

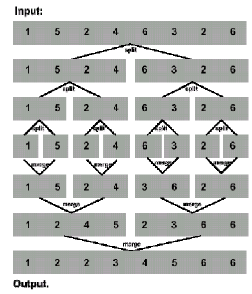


Merge Sort (Example) - 22



Merge Sort summarized

- To sort n numbers
  - If n=1 done!
  - Else
    - Recursively sort 2 lists of numbers $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ elements
    - Merge 2 sorted lists in $\Theta(n)$ time
- Strategy
  - Break problem into similar (smaller) subproblems
  - Recursively solve subproblems
  - Combine solutions to answer



Running Time of Merge Sort

- Can also be expressed as a recurrence

$$T(n) = \begin{cases} \text{solving\_trivial\_problem} & \text{if } n = 1 \\ \text{num\_pieces } T(n / \text{subproblem\_size\_factor}) + \text{dividing} + \text{combining} & \text{if } n > 1 \end{cases}$$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$$T(n) = \Theta(n \lg n)$$

Recurrence

- There are three methods to solve recurrence (obtain O and Θ bounds)
  - Substitution method
    - Based on mathematical induction
  - Recursion-tree method
    - Bases on trees
  - Master method
    - Requires memorization of three cases

Substitution method

- Two steps
  - Guess the form of the solution
    - Applied to cases when it is easy to guess the answer
  - Use mathematical induction to find the constants and show that the solution works
- Can be used to estimate upper (lower) bound of a recurrence
  - Substitutes the guesses answer by the function defined using induction

Substitution method

- Find the running time (upper bound) of merge sort
  - Assume that $n=2^b$, for some b

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$T(n) = 2T(n/2) + \Theta(n)$ Assume Θ(n)= n

$T(n) = 2T(n/2) + n$

Guess that $T(n) = \Theta(n \log n)$

Prove that $T(n) \leq cn \lg n$ *for a proper choice of c*

Substitution method

- Find the running time (upper bound) of merge sort
  - Assume that $n=2^b$, for some b

$$T(n)=\begin{cases}1 & \text{if } n=1\\ 2T(n/2)+n & \text{if } n>1\end{cases}$$

T(n) = 2T(n/2) + n

Guess that $T(n)=\Theta(n\log n)$

Prove that $T(n) \le cn \lg n$ *for a proper choice of c*

Substitution method

- Find the running time (upper bound) of merge sort
  - Assume that $n=2^b$, for some b

$$T(n)=\begin{cases}1 & \text{if } n=1\\ 2T(n/2)+n & \text{if } n>1\end{cases}$$

$n_0 = 1 \rightarrow T(1) = 1$
$1 \le c\lg 1 \rightarrow$ No

T(n) = 2T(n/2) + n

Guess that $T(n)=\Theta(n\log n)$

Prove that $T(n) \le cn \lg n$ *for a proper choice of c*

Substitution method

- Find the running time (upper bound) of merge sort
  - Assume that $n=2^b$, for some b

$$T(n)=\begin{cases}1 & \text{if } n=1\\ 2T(n/2)+n & \text{if } n>1\end{cases}$$

$n_0 = \cancel{1 \rightarrow T(1) = 1}$
$1 \le \cancel{c\lg 1 \rightarrow}$ No
Replace T(1) by T(2) and T(3):
T(2) = 4
T(3) = 5
$n_0 = 2$
For n> 3, recurrence does not depend on T(1)

T(n) = 2T(n/2) + n

Guess that $T(n)=\Theta(n\lg n)$

Prove that $T(n) \le cn \lg n$ *for a proper choice of c*

Substitution method

T(n) = 2T(n/2) + n

Prove that $T(n) \le cn \lg n$

Assuming that the bound holds for n/2, $T(n/2) \le cn/2 \lg (n/2)$

$$\begin{aligned} T(n) &\le 2[cn/2 \lg (n/2)] + n\\ &\le cn \lg (n/2) + n\\ &\le cn \lg n - cn\lg 2 + n\\ &\le cn \lg n - cn + n\\ &\le cn \lg n\end{aligned}$$

Choose positive value of c that holds for T(2) and T(3)

Substitution method

T(n) = 2T(n/2) + n

Prove that $T(n) \le cn \lg n$

Assuming that the bound holds for n/2, $T(n/2) \le cn/2 \lg (n/2)$

$$\begin{aligned} T(n) &\le 2[cn/2 \lg (n/2)] + n\\ &\le cn \lg (n/2) + n\\ &\le cn \lg n - cn\lg 2 + n\\ &\le cn \lg n - cn + n\\ &\le cn \lg n\end{aligned}$$

Choose positive value of c that holds for T(2) and T(3)
If c = 1
$T(2) \le 1x2lg2$
$4 \le 2 \rightarrow$ does not hold
If c = 2
$T(2) \le 2x2lg2$
$4 \le 4 \rightarrow$ holds

Substitution method

T(n) = 2T(n/2) + n

Prove that $T(n) \le cn \lg n$

Assuming that the bound holds for n/2, $T(n/2) \le cn/2 \lg (n/2)$

$$\begin{aligned} T(n) &\le 2[cn/2 \lg (n/2)] + n\\ &\le cn \lg (n/2) + n\\ &\le cn \lg n - cn\lg 2 + n\\ &\le cn \lg n - cn + n\\ &\le cn \lg n \text{ (holds if } c > 1)\end{aligned}$$

Choose positive value of c that holds for T(2) and T(3)
If c = 1
$T(3) \le 1x3lg3$
$5 \le 3x1,6 \rightarrow$ does not hold
If c = 2
$T(3) \le 2x3lg3$
$5 \le 6x1,6 \rightarrow$ holds

Substitution method

- Necessary to shows that the solution holds for the boundary conditions
  - Show that they are suitable as base cases
    - For the previous example, show that we can choose a positive c such that $T(n) \leq cn \lg n$ (for $n \geq n_0$) works for the boundary conditions
      - Be $n_0 = 2$
      - It can be easily shown that any choice of $c \geq 2$, the solution holds

Observations

- Important to distinguish between
  - The base case of of the recurrence
    - When n =1
  - The base case of the inductive proof
    - When n =2 and n = 3
- For most recurrences, boundary conditions can be extended
  - In order to make inductive assumption to work for small values of n

Substitution method

- Why the name substitution?
  - Because it substitutes the function

    $$T(n) = 2T(n/2) + n$$

  - By the guessed solution

    $$T(n)=\Theta(n \lg n)$$

  - And access if the guessed solution works for small values

Example: Finding Min and Max

- Given an unsorted array, find a minimum and a maximum element in

INPUT: A[l..r] – an unsorted array of integers, $l \leq r$.
OUTPUT: (*min, max*) such that $\forall j$ ($l \leq j \leq r$): A[j] ≥ *min* and A[j] ≤ *max*

```
MinMax(A, l, r):
   if l = r then return (A[l], A[r])
   q←⌊(l+r)/2⌋
   (minl, maxl)← MinMax(A, l, q)
   (minr, maxr)← MinMax(A, q+1, r)
   if minl < minr then min = minl else min = minr
   if maxl > maxr then max = maxl else max = maxr
   return (min, max)
```


Example: Finding Min and Max

- Given an unsorted array, find a minimum and a maximum element in

INPUT: A[l..r] – an unsorted array of integers, $l \leq r$.
OUTPUT: (*min, max*) such that $\forall j$ ($l \leq j \leq r$): A[j] ≥ *min* and A[j] ≤ *max*

```
MinMax(A, l, r):
   if l = r then return (A[l], A[r])            Trivial case
   q←⌊(l+r)/2⌋                                   Divide
   (minl, maxl)← MinMax(A, l, q)              }
   (minr, maxr)← MinMax(A, q+1, r)            }  Conquer
   if minl < minr then min = minl else min = minr }
   if maxl > maxr then max = maxl else max = maxr }  Combine
   return (min, max)
```


Next Week

- Continue analysis of the running time of recursive algorithms
  - Like divide-and-conquer
- Master algorithm

Acknowledgement

- A large part of this material was adapted from
  - Simonas Šaltenis, Algorithms and Data Structures, Aalborg University, Denmark
  - Mary Wootters, Design and Analysis of Algorithms, Stanford University, USA
  - George Bebis, Analysis of Algorithms CS 477/677, University of Nevada, Reno
  - David A. Plaisted, Information Comp 550-001, University of North Carolina at Chapel Hill

Questions