CSC236 fall 2016

divide and conquer
recursive correctness

Danny Heap
heap@cs.toronto.edu / BA4270 (behind elevators)
http://www.cdf.toronto.edu/~csc236h/fall/
416-978-5899

Using Introduction to the Theory of Computation, Chapter 3

Outline

divide and conquer (recombine)

using the Master Theorem

binary search

more D&C: fast multiplication

Notes

General case

revisit. . .

Class of algorithms: partition problem into b *roughly* equal subproblems, solve, and recombine:

$$T(n) = \begin{cases} k & \text{if } n \leq B \\ a_1 T(\lceil n/b \rceil) + a_2 T(\lfloor n/b \rfloor) + f(n) & \text{if } n > B \end{cases}$$

where $B, k > 0$, $a_1, a_2 \geq 0$, and $a_1 + a_2 > 0$. $f(n)$ is the cost of splitting and recombining.

Master Theorem

(for divide-and-conquer recurrences)

If f from the previous slide has $f \in \theta(n^d)$, then

$$T(n) \in \begin{cases} \theta(n^d) & \text{if } a < b^d \\ \theta(n^d \log n) & \text{if } a = b^d \\ \theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Proof sketch

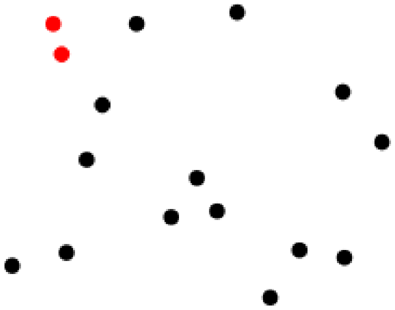
1. Unwind the recurrence, and prove a result for $n = b^k$

2. Prove that T is non-decreasing

3. Extend to all n, similar to MergeSort

closest point pairs

see Wikipedia



divide-and-conquer v0.1

an $n \lg n$ algorithm

P is a set of points

1. Construct (sort) P_x and P_y
2. For each recursive call, construct ordered L_x, L_y, R_x, R_y
3. Recursively find closest pairs (l_0, l_1) and (r_0, r_1), with minimum distance δ
4. V is the vertical line splitting L and R, D is the δ-neighbourhood of V, and D_y is D ordered by y-ordinate
5. Traverse D_y looking for mininum pairs 15 places apart
6. Choose the minimum pair from D_y versus (l_0, l_1) and (r_0, r_1).

recursive binary search

```
def recBinSearch(x, A, b, e) :
  if b == e :
    if x <= A[b] :
      return b
    else :
      return e + 1
  else :
    m = (b + e) // 2 # midpoint
    if x <= A[m] :
      return recBinSearch(x, A, b, m)
    else :
      return recBinSearch(x, A, m+1, e)
```

conditions, pre- and post-

- x and elements of A are comparable
- e and b are valid indices, $b \leq e$
- $A[b..e]$ is sorted non-decreasing

RecBinSearch(x, A, b, e) terminates and returns index p

- $b \leq p \leq e + 1$
- $b < p \Rightarrow A[p-1] < x$
- $p \leq e \Rightarrow x \leq A[p]$

(except for boundaries, returns p so that $A[p-1] < x \leq A[p]$)

precondition ⇒ termination and postcondition

Proof: induction on $n = e - b + 1$

Base case, $n = 1$: Terminates because there are no loops or further calls, returns $p = b = e \Leftrightarrow x \leq A[b = p]$ **or** $p = b + 1 = e + 1 \Leftrightarrow x > A[b = p - 1]$, so postcondition satisfied. Notice that the choice forces if-and-only-if.

Induction step: Assume $n > 1$ and that the postcondition is satisfied for inputs of size $1 \leq k < n$ that satisfy the precondition. Call RecBinSearch(A,x,b,e) when $n = e - b + 1 > 1$. Since $b < e$ in this case, the test on line 1 fails, and line 7 executes. **Exercise:** $b \leq m < e$ in this case. There are two cases, according to whether $x \leq A[m]$ or $x > A[m]$.

Case 1: $x \leq A[m]$

- Show that IH applies to RBS(x,A,b,m)
- Translate the postcondition to RBS(x,A,b,m)

- Show that RBS(x,A,b,e) satisfies postcondition

Case 2: $x > A[m]$

- Show that IH applies to RBS(x,A,m+1,e)
- Translate postcondition to RBS(x,A,m+1,e)

- Show that RBS(x,A,b,e)

what could possibly go wrong?

- $m = \lceil \frac{e+b}{2.0} \rceil$
- $x < A[m]$
- $\ldots$
- Either prove correct, or find a counter-example

multiply lots of bits

what if they don't fit into a machine instruction?

$$\begin{array}{r} 1101 \\ \times 1011 \\ \hline \end{array}$$

divide and recombine

recursively...

$$\begin{array}{r|l} 11 & 01 \\ \hline \times 10 & 11 \\ \hline \end{array}$$

$$xy = 2^n x_1 y_1 + 2^{n/2}(x_1 y_0 + y_1 x_0) + x_0 y_0$$

compare costs

n n-bit additions versus:

1. divide each factor (roughly) in half
2. multiply the halves (recursively, if they're too big)
3. combine the products with shifts and adds

Gauss's trick

$$xy = 2^n x_1 y_1 + x_0 y_0 + 2^{n/2}\left((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0\right)$$

Gauss's payoff

lose one multiplication

1. divide each factor (roughly) in half
2. sum the halves
3. multiply the sum and the halves Gauss-wise
4. combine the products with shifts and adds

Notes