office hours weeks 8+9: Thursday noon-2 room TBA

CSC236 fall 2016

divide and conquer

recursive correctness

does not mean: correct because assumed correct...

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)

n^2

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

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

$\underline{\theta(n^t \lg n)}$

Proof sketch

Prove Master Theorem

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

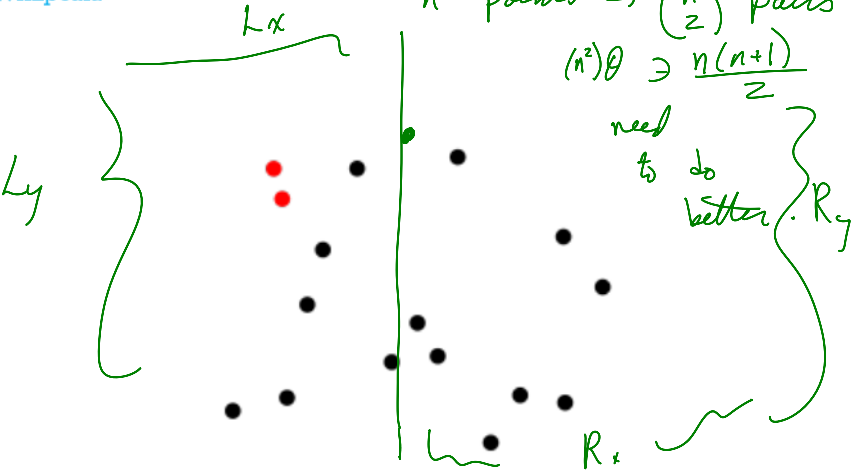
see Wikipedia

P = list of points

n points $\rightarrow \binom{n}{2}$ pairs.

$(n^2)\theta \ni \frac{n(n+1)}{2}$

need to do better.



divide-and-conquer v0.1

⓪ $P \rightarrow P$ sorted by x $\quad \Theta(n \lg n)$

① divide P in half – $\Theta(1)$

② find min_L, min_R $\sim T(n/2) \times 2$

③ $\delta = \min(min_L, min_R)$

④ take $\min(\delta, \underline{\text{across border}})$

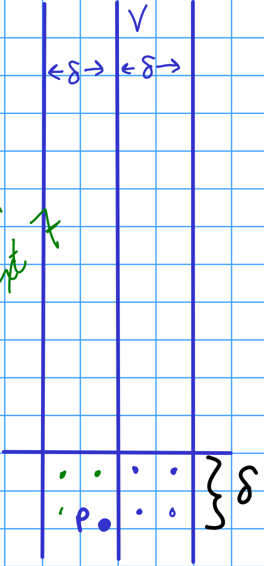
$a = 2$
$b = 2$
$d = 2$

$\left(\frac{n}{2}\right) \times \left(\frac{n}{2}\right) = \frac{n^2}{4} \in \Theta(n^2)$ complexity

$$T(n) = \begin{cases} k & n = 2 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \underline{f(n)} & n > 2 \end{cases}$$

$f(n) \in \Theta(n^2)$

$2 < 2^2$



traverse D_y bottom to top, for each point check next 7

look-ahead 7, then stop looking

quadrant

$$\sqrt{\left(\frac{\delta}{2}\right)^2+\left(\frac{\delta}{2}\right)^2}$$

$$\sqrt{\frac{2\delta^2}{4}} = \sqrt{\frac{\delta^2}{2}} = \frac{\delta}{\sqrt{2}} < \delta$$

an $n \lg n$ algorithm

$a \cdots b^d$
$2 \cdots 2$

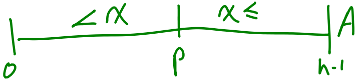
before D+C. outside
$\Theta(n \lg n)$ D+C.

P is a set of points

1. Construct (sort) P_x and P_y
2. For each recursive call, construct ordered $\overset{\Theta(1)}{L_x}, \overset{\Theta(n)}{L_y}, \overset{\Theta(1)}{R_x}, \overset{\Theta(n)}{R_y}$
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 $\Theta(n)$ $\Theta(n)$
5. Traverse D_y looking for mininum pairs ~~15~~ 7 places apart
6. Choose the minimum pair from D_y versus (l_0, l_1) and (r_0, r_1). $\Theta(1)$

Master Theorem $a = 2$ $b = 2$ $d = 1$ 7

recursive binary search



Array — non-decreasing
— comparable.

$0 \leq b \leq e$

$n = e - b + 1$

```
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)
```

7

finds p (position),
position useful for place to insert x

1. $0 \leq p \leq e+1$
2. $b < p \Rightarrow A[p-1] < x$
3. $p \leq e \Rightarrow A[p] \geq x$

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 ✓

Postconditions

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 $A[p-1] < x \leq A[p]$)

must show:

precondition $\Rightarrow$ 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: $x \le A[m]$!

$$0 \le 1 \le m-b+1 < \overbrace{e-b+1}^{n}$$

$m \ge b$

$m-b+1 \ge b-b+1$

$m<e$

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

terminates + returns p

1. $b \le p \le m+1$
2. $b<p \Rightarrow A[p-1]<x$
3. $p \le m \Rightarrow A[p] \ge x$

IH

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

1. $b \le p$ from IH. $p \le m+1 < e+1$ # by IH # $m<e$
2. $b<p \Rightarrow A[p-1]<x$ → # from IH.
3. $p \le m+1 \le e+1$

$\hookrightarrow p \le m$ or $p = m+1$

$\Longrightarrow A[p] \ge x$

NEVER HAPPENS if $p = m+1$, by (2) $A[p-1] = A[m] < x$

Case 2: $x > A[m]$

$e > m$

$e-(m+1)+1 < n = e-b+1$

$1 \le e - m \le e-b < e-b+1$

$\cancel{e-b}$ # $m \geq$

$b \le m$

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

b

- $m+1 \le p \le e+1$
- $m+1 < p \Rightarrow A[p-1] < x$
- $p \le e \Rightarrow A[p] \ge x$

IH

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

$p \le e+1$ — by IH

$p \ge m+1 \ge b$ # $m \ge b$

$p \le e \Rightarrow A[p] \ge x$ # by ///

$p > b \Rightarrow p = m+1$ or $p > m+1$ # IH

$p > m+1 \Rightarrow A[p-1] < x$ # IA

$p = m+1 \Rightarrow A[p-1] = A[m] < x$ # by

what could possibly go wrong?

- $m = \lceil \frac{e+b}{2.0} \rceil$

prove or disprove that this change keeps RBS correct

- $x < A[m]$

prove or disprove that this change keeps RBS correct

- ...

- 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

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

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