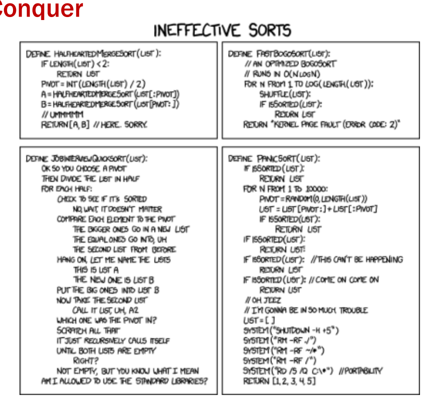
CSE 421: Algorithms

Winter 2014

Lecture 11: Divide & Conquer

Reading:
Sections 5.1-5.4



fast exponentiation

- Power(**a**,**n**)
  - **Input:** integer **n** and number **a**
  - **Output:** a^n
- Obvious algorithm
  - **n-1** multiplications
- Observation:
  - if **n** is even, **n=2m**, then $a^n = a^m \bullet a^m$

divide & conquer qlgorithm

```
Power(a,n):
  if n=0 then
    return(1)
  else if n=1 then
    return(a)
  else
    x ← Power(a, ⌊n/2⌋)
    if n is even then
      return(x•x)
    else
      return(a•x•x)
```

analysis

- Worst-case recurrence
  - $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2$ for $n \geq 1$
  - $T(1) = 0$
- Time:
- More precise analysis:

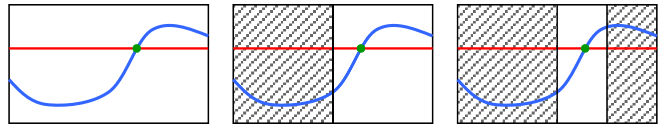
  $T(n) = \lceil \log_2 n \rceil +$ # of **1**'s in **n**'s binary representation

practical application: RSA

- Instead of a^n want a^n mod N
  - a^{i+j} mod N = ((a^i mod N)•(a^j mod N)) mod N
  - same algorithm applies with each x•y replaced by
    ((x mod N)•(y mod N)) mod N

- In RSA cryptosystem (widely used for security)
  - need a^n mod N where a, n, N each typically have 1024 bits
  - Power: at most 2048 multiplies of 1024 bit numbers
    relatively easy for modern machines
  - Naive algorithm: 2^{1024} multiplies

binary search for roots (bisection method)



- Given:
  - continuous function f and two points a<b with f(a) ≤ 0 and f(b) > 0

- Find:
  - approximation to c s.t. f(c)=0 and a<c<b

bisection method

```
Bisection(a,b,ε):
  if (a-b) < ε then
    return(a)
  else
    c ←(a+b)/2
    if f(c) ≤ 0 then
        return(Bisection(c,b,ε))
    else
        return(Bisection(a,c,ε))
```

analysis

- At each step we halved the size of the interval
- It started at size b-a
- It ended at size ε

- # of calls to f is $\log_2\left(\frac{b-a}{\epsilon}\right)$

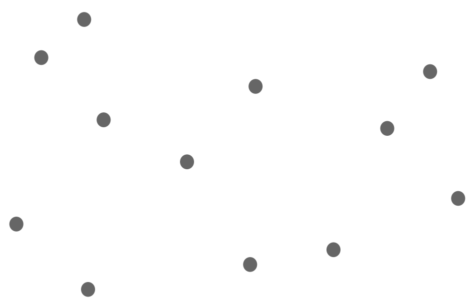
old favorites

- Binary search
  - One subproblem of half size plus one comparison
  - Recurrence $T(n) = T(\lceil n/2 \rceil)+1$ for $n \geq 2$
    $T(1) = 0$
  So $T(n)$ is $\lceil \log_2 n \rceil + 1$
- Mergesort
  - Two subproblems of half size plus merge cost of $n-1$ comparisons
  - Recurrence $T(n) \leq 2T(\lceil n/2 \rceil)+n-1$ for $n \geq 2$
    $T(1) = 0$
    Roughly n comparisons at each of $\log_2 n$ levels of recursion
    So $T(n)$ is roughly $2n \log_2 n$

euclidean closest pair

- Given a set P of n points $p_1,\ldots,p_n$ with real-valued coordinates
- Find the pair of points $p_i,p_j \in P$ such that the Euclidean distance $d(p_i,p_j)$ is minimized
- $\Theta(n^2)$ possible pairs
- In one dimension?
- What about points in the plane?

closest pair in the plane

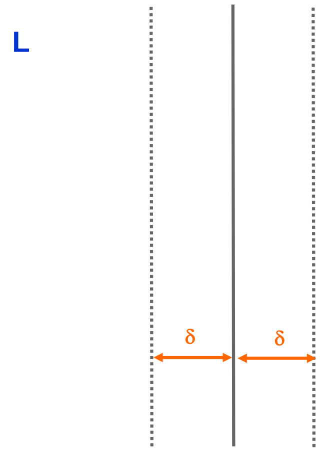


No single direction along which one can sort points to guarantee success!

divide and conquer?

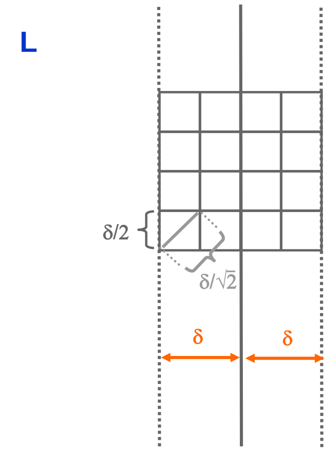
- Sort the points by their x coordinates
- Split the points into two sets of $n/2$ points L and R by x coordinate
- Recursively compute
  - closest pair of points in L, (p_L,q_L)
  - closest pair of points in R, (p_R,q_R)
- Let $\delta=\min\{d(p_L,q_L),d(p_R,q_R)\}$ and let (p,q) be the pair of points that has distance δ

clever girl



Any pair of points p∈L and q∈R with d(p,q)<δ must lie in band

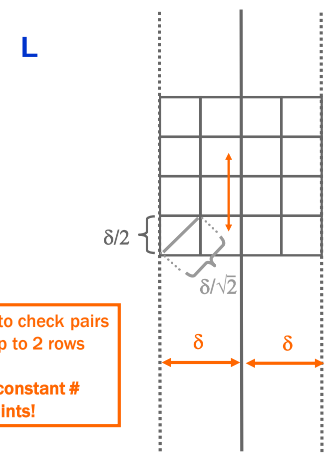
clever girl



Any pair of points p∈L and q∈R with d(p,q)<δ must lie in band

No two points can be in the same grey box

clever girl



Any pair of points p∈L and q∈R with d(p,q)<δ must lie in band

No two points can be in the same grey box

Only need to check pairs of points up to 2 rows apart.
At most a constant # of other points!

closest pair recombining

- Sort points by y coordinate ahead of time
- On recombination only compare each point in δ-band of L∪R to the 11 points in δ-band of L∪R above it in the y sorted order
  - If any of those distances is better than δ replace (p,q) by the best of those pairs
- O(n log n) for x and y sorting at start
- Two recursive calls on problems on half size
- O(n) recombination
- Total:

sometimes two sub-problems aren't enough

- More general divide and conquer
  - You've broken the problem into a different sub-problems
  - Each has size at most n/b
  - The cost of the break-up and recombining the sub-problem solutions is $O(n^k)$
- Recurrence: $T(n) \le a \cdot T(n/b) + c \cdot n^k$

master divide and conquer recurrence

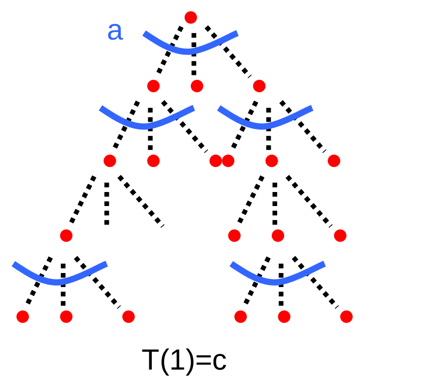
- If $T(n) \le a \cdot T(n/b) + c \cdot n^k$ for $n > b$ then
  - if $a > b^k$ then $T(n)$ is $\Theta\left(n^{\log_b a}\right)$
  - if $a < b^k$ then $T(n)$ is $\Theta\left(n^k\right)$
  - if $a = b^k$ then $T(n)$ is $\Theta(n^k \log n)$

master divide and conquer recurrence

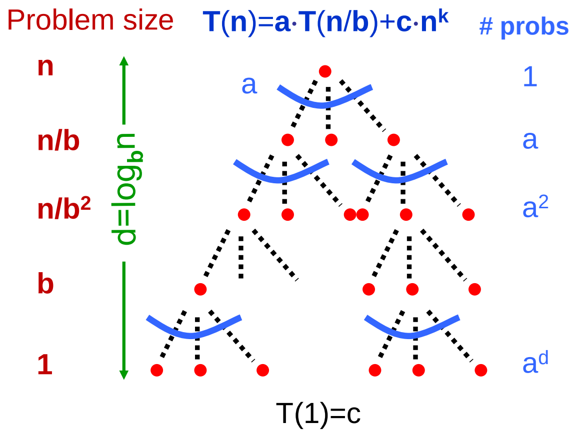
- If $T(n) \le a \cdot T(n/b) + c \cdot n^k$ for $n > b$ then
  - if $a > b^k$ then $T(n)$ is $\Theta\left(n^{\log_b a}\right)$
  - if $a < b^k$ then $T(n)$ is $\Theta\left(n^k\right)$
  - if $a = b^k$ then $T(n)$ is $\Theta(n^k \log n)$
- Works even if it is $\left\lceil \frac{n}{b} \right\rceil$ instead of $\frac{n}{b}$.

proving the master recurrence

Problem size $T(n)=a \cdot T(n/b)+c \cdot n^k$ # probs



proving the master recurrence



geometric series

- $S \;= t + tr + tr^2 + \dots + tr^{n-1}$
- $r{\cdot}S \;= \quad tr + tr^2 + \dots + tr^{n-1} + tr^n$
- $(r-1)S = tr^n - t$
- so $S = t\,(r^n - 1)/(r-1)$ if $r \neq 1$.

- **Simple rule**
  - If $r \neq 1$ then S is a constant times largest term in series

total cost

- Geometric series
  - ratio a/b^k
  - $d+1 = \log_b n + 1$ terms
  - first term cn^k, last term ca^d
- If $a/b^k = 1$
  - all terms are equal T(n) is $\Theta(n^k \log n)$
- If $a/b^k < 1$
  - first term is largest T(n) is $\Theta(n^k)$
- If $a/b^k > 1$
  - last term is largest T(n) is

$$\Theta(a^d) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$$

proving the master recurrence