CSE 101, Winter 2016

Design and Analysis of Algorithms

Lecture 4: Divide and Conquer (I)

Class URL: http://vlsicad.ucsd.edu/courses/cse101-w16/
Divide and Conquer ("DQ")

- **First “paradigm” or “framework”**
  
  \[
  \text{DQ}(S) \\
  \text{if } S \text{ is small return } \text{ADHOC}(S) \\
  \text{else} \\
  \quad \text{decompose } S \text{ into subproblems } S_1, \ldots, S_k \\
  \quad \text{for } i = 1 \text{ to } k \text{ do } y_i = \text{DQ}(S_i) \\
  \quad \text{recombine } y_i \text{ into solution } y \\
  \text{return } y
  \]

- **“Universal method”:** Mergesort, Quicksort, FFT, Matrix/Integer arithmetic are classic examples
DQ for Sorting

• Divide (into two equal parts)
• Conquer (solve for each part separately)
• Combine separate solutions
• Mergesort
  – Divide into two equal parts
  – Sort each part using Mergesort (recursion!!!)
  – Merge two sorted subsequences

(Note: Quicksort is another DQ algorithm for sorting)
Merging Two Subsequences

Sequence #1: \( x[1], x[2], \ldots, x[m] \)
Sequence #2: \( y[1], y[2], \ldots, y[n] \)

If \( y[i] < x[j] \) then compare \( y[i+1] \) and \( x[j] \),
else compare \( y[i] \) and \( x[j+1] \)

After comparing \( x[1] \) and \( y[1] \), we advance in \( x[.] \) and \( y[.] \)
a total of \((m-1) + (n-1)\) times \( \Rightarrow m + n - 1 \) comparisons
(edges between red and blue) = linear time

\( O(m+n) \)

i.e., linear time
Mergesort Tree of Recursive Subproblems

DIVIDE

log \( n \) levels

CONQUER, COMBINE

\((n \text{ comparisons per level}) \times (\log n \text{ levels}) = (n \log n \text{ runtime})\)
Integer Multiplication

- Multiplying Large Integers (Karatsuba-Ofman)
  \[ A = r^{n-1}a_{n-1} + \ldots + r^1a_1 + a_0, \quad r = \text{radix} \]
- “Classic” approach = what you learned 12 years ago

\[
\begin{array}{c}
\text{a}_{n-1}\text{a}_{n-2}\ldots\text{a}_1\text{a}_0 \\
\times \quad \text{b}_{n-1}\text{b}_{n-2}\ldots\text{b}_1\text{b}_0 \\
\hline
\end{array}
\]

- $\Theta(n^2)$ work

\[ a_{i}, b_{i}, a_{i}b_{i}, a_{i}b_{i}, a_{j}b_{j}, a_{j}b_{j}, 0 \]

4 “subproblems” each half the size of “problem” (each 1x1)
(Unhelpful) DQ for Integer Multiplication

- Can we apply D/Q?
  - Let $n = 2s$, $r = 10 \equiv \text{radix}$
  - $A = wx$, $B = yz$
  - $AB = xz + 10^s(wz + xy) + 10^{2s}wy$

- Complexity analysis
  - 4 $n/2$-digit multiplications: $xz$, $wz$, $xy$, $wy$
  - Digit-shifting: multiplication by $10^s$, $10^{2s}$
  - 3 additions

$$T(n) = 4T(n/2) + \Theta(n)$$

$$T(n) \leq 4T(n/2) + cn$$

$$\leq 4 \left[ 4T(n/4) + \frac{cn}{2} \right] + cn$$

$$= 16T(n/4) + (1 + 2)cn$$

$$\leq 16 \left[ 4T(n/8) + \frac{cn}{4} \right] + (1 + 2)cn$$

$$= 64T(n/8) + (1 + 2 + 4)cn$$

$$\ldots$$

$$\leq 4^k T(n/2^k) + (1 + 2 + 4 + \ldots + 2^{k-1})cn$$

For $k = \log_2 n$: $T(n) \leq n^2 T(1) + cn^2 = O(n^2)$
(Helpful) DQ for Integer Multiplication

- Observation: \( r' = (w+x)(y+z) = wy + (wz+xy) + xz \)
  - \( r' \leftarrow (w+x)(y+z) \)
  - \( p \leftarrow wy \)
  - \( q \leftarrow xz \)
  - return \( 10^{2sp} + 10^s(r'-p-q) + q \)
  - \( T(n) = 3T(n/2) + \theta(n) \)

- \( T(n) \leq 3 \cdot T(n/2) + cn \)
  - \( \leq 3 \left[ 3T(n/4) + cn/2 \right] + cn \)
  - \( = 9 \cdot T(n/4) + (1 + 3/2)cn \)
  - \( \leq 9 \left[ 3T(n/8) + cn/4 \right] + (1 + 3/2)cn \)
  - \( = 27 \cdot T(n/8) + (1 + 3/2 + 9/4)cn \)
  - \( \ldots \)
  - \( \leq 3^k \cdot T(n/2^k) + (1 + 3/2 + 9/4 + \ldots + (3/2)^{k-1})cn \)

For \( k = \log_2 n \):

- \( T(n) \leq 3^{\log_2 n} \cdot T(1) + cn \cdot 2 \cdot (3/2)^{\log_2 n} \)
  - \( = O(3^{\log_2 n}) \approx O(n^{\log_2 3}) = O(n^{1.5849\ldots}) \)

What is \( (x^n - 1) / (x - 1) \)?

\[ \frac{a^{\log_b c}}{c^{\log_b a}} = \frac{\log_b a + \log_b c}{\log_b a \cdot \log_b c} \]
(Take log base b)
Matrix Multiplication

- $A = [...], B = [...]$ are $n \times n$ matrices
- $a_{11}, a_{12},$ etc are $n/2 \times n/2$ submatrices
- $M = AB = [...]$
  - where $m_{11} = a_{11}b_{11} + a_{12}b_{21}$ etc.
  - Evaluation requires 8 multiplies, 4 adds
- $T(n) = 8T(n/2) + O(n^2)$
  $\in \Theta(n^3)$ (will see why later)
Strassen’s Matrix Multiplication

\[ p_1 = (a_{21} + a_{22} - a_{11})(b_{22} - b_{12} + b_{11}) \]

\[ p_2 = a_{11}b_{11} \]

\[ p_3 = a_{12}b_{21} \]

\[ p_4 = (a_{11} - a_{21})(b_{22} - b_{12}) \]

\[ p_5 = (a_{21} + a_{22})(b_{12} - b_{11}) \]

\[ p_6 = (a_{12} - a_{21} + a_{11} - a_{22})b_{22} \]

\[ p_7 = a_{22}(b_{11} + b_{22} - b_{12} - b_{21}) \]

\[ AB_{11} = p_2 + p_3 \]

\[ AB_{12} = p_1 + p_2 + p_5 + p_6 \]

\[ AB_{21} = p_1 + p_2 + p_4 + p_7 \]

\[ AB_{22} = p_1 + p_2 + p_4 + p_5 \]

- \( T(n) = 7T(n/2) + O(n) \) // 7 multiplies, 24 adds
  - Can get to 15 adds

\[ \in \Theta(n^{2.81}) \text{ (will see why later)} \]
General Formula (aka “Master Theorem”)

Recurrence \[ T(n) \leq a \cdot T(n/b) + O(n^d) \]

1) \( T(n) = O(n^d) \) if \( a < b^d \)

2) \( T(n) = O(n^d \log n) \) if \( a = b^d \)

3) \( T(n) = O(n^{\log_b a}) \) if \( a > b^d \)

Type (3): integer multiplication, Strassen

Type (2): mergesort
Master Theorem Examples

• Mergesort \( T(n) = 2T(n/2) + \Theta(n) \)

\[
T(n) = \Theta(n \log n)
\]

• Strassen Matrix Multiply \( T(n) = 7T(n/2) + \Theta(n^2) \)

\[
T(n) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.807})
\]
Proof of “Master Theorem” for Recurrences

Recurrence \( T(n) \leq a \cdot T(n/b) + O(n^d) \)

1) \( T(n) = O(n^d) \) if \( a < b^d \)

2) \( T(n) = O(n^d \log n) \) if \( a = b^d \)

3) \( T(n) = O(n^{\log_b a}) \) if \( a > b^d \)

\( \bullet \) Assume \( n \) is a power of \( b \) \( \rightarrow \) can ignore rounding in \( \lceil n/b \rceil \)

\( \bullet \) Subproblem size decreases by factor of \( b \) with each level of recursion \( \rightarrow \) reaches base case after \( \log_b n \) levels

\( \bullet \) Branching factor \( a \) \( \rightarrow \) \( k^{th} \) level of tree has \( a^k \) subproblems each of size \( n/b^k \)

\( \bullet \) Total work done at \( k^{th} \) level is \( a^k \times O(n/b^k)^d = O(n^d) \times (a / b^d)^k \)
Proof of “Master Theorem” for Recurrences

Recurrence: \( T(n) \leq a \cdot T(n/b) + O(n^d) \)

1) \( T(n) = O(n^d) \) if \( a < b^d \)
2) \( T(n) = O(n^d \log n) \) if \( a = b^d \)
3) \( T(n) = O(n^{\log_b a}) \) if \( a > b^d \)

- Total work done at \( k \)th level is \( a^k \times O(n/b^k)^d = O(n^d) \times (a / b^d)^k \)
- As \( k \) goes from 0 (root) to \( \log_b n \) (leaves), have geometric series with ratio \( a / b^d \)
  - \( a / b^d < 1 \rightarrow \) sum is \( O(n^d) \)
  - \( a / b^d > 1 \rightarrow \) sum is given by last term, \( O(n^{\log_b a}) \)
  - \( a / b^d = 1 \rightarrow \) sum is given by \( O(\log n) \) terms equal to \( O(n^d) \)
Discussion of Lower Bounds

• Lower bounds are useful
  – **Know** that you can’t do any better than a certain algorithmic (time, space, etc.) complexity

• Lower bounds can be “migrated” from one problem to another, via a type of argument called “**reduction**”

• Roughly:
  – “We know that problem A has complexity lower bound X.”
  – “But, I can solve any instance of A using a solver for problem B.”
  – “Therefore, problem B must **also** have complexity lower bound X.”

• We will see a lot of reductions if/when we discuss NP-completeness later this quarter …
Convex Hull (Lecture 1, Slide 31)

- Given a planar pointset $S$, its **convex hull** consists of all extreme points, i.e., points of $S$ that cannot be expressed as convex combinations of any other points of $S$

→ If you put a rubber band around the pointset, where does it ‘snap tight’?

- $n^2$ pairs
- does pair $(p_i, p_j)$ define an edge of the convex hull? $\Theta(n)$ to $\Theta(n^3)$ algorithm!
Convex Hull

- Task: Return points in convex hull of \( S \) in order
- What is a lower bound on complexity?
  - Q: If sorting requires \( \Omega(n \log n) \) time, then why MUST convex hull also require \( \Omega(n \log n) \) time? // will return to this

- How would you go about finding the convex hull?
Convex Hull (1)

- Task: Return points in convex hull of S in order

- Many known approaches:
  - Stretch $\theta(n^2)$
  - Giftwrap $\theta(n^2)$
  - Graham Scan $\theta(n \log n)$
  - D/Q $\theta(n \log n)$
    - Algorithm by Preparata and Hong
    - left-right split
    - find convex hulls
    - merge convex hulls
Convex Hull (2)

- Task: Return points in convex hull of S in order

- Many known approaches:
  - Stretch $\theta(n^2)$
  - Giftwrap $\theta(n^2)$
  - Graham Scan $\theta(n \log n)$
  - D/Q $\theta(n \log n)$
    - Algorithm by Preparata and Hong
    - left-right split
    - find convex hulls
    - merge convex hulls
Convex Hull (3)

• Task: Return points in convex hull of S in order

• Many known approaches:
  – Stretch $\theta(n^2)$
  – Giftwrap $\theta(n^2)$
  – Graham Scan $\theta(n \log n)$
  – D/Q $\theta(n \log n)$
    • Algorithm by Preparata and Hong
    • left-right split
    • find convex hulls
    • merge convex hulls
A Lower Bound on Convex Hull

- Task: \text{sort} the set of numbers \{3,1,4,5,2\}

- Task: \text{find the convex hull of} the set of points \{(3,9), (1,1), (4,16), (5,25), (2,4)\}
A Lower Bound on Convex Hull

• Given: Sorting LB is $\Omega(n \log n)$
• Reduction from Sorting to Convex Hull
  – Input: an arbitrary instance $I_{\text{SORT}}$ of SORT
  – Transform $I_{\text{SORT}} = \{x_1, x_2, \ldots, x_n\}$ into an instance $I_{\text{C-HULL}} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$ of C-HULL
  – Solve the C-HULL problem for instance $I_{\text{C-HULL}}$
  – Transform solution of $I_{\text{C-HULL}}$ to solution of $I_{\text{SORT}}$
A Lower Bound on Convex Hull

- **Given:** Sorting LB is $\Omega(n \log n)$
- **Reduction from Sorting to Convex Hull**
  - Input: an arbitrary instance $I_{\text{SORT}}$ of SORT
  - **Transform** $I_{\text{SORT}} = \{x_1, x_2, ..., x_n\}$ into an instance $I_{\text{C-HULL}} = \{(x_1, x_1^2), (x_2, x_2^2), ..., (x_n, x_n^2)\}$ of C-HULL
  - Solve the C-HULL problem for instance $I_{\text{C-HULL}}$
  - **Transform** solution of $I_{\text{C-HULL}}$ to solution of $I_{\text{SORT}}$

- **Note:** Each **Transform** has $O(n)$ complexity
  - $\Rightarrow$ C-HULL solver cannot be faster than $\Omega(n \log n)$
The Closest Pair Problem

• **Closest Pair:** Given n points in the plane, return the closest pair

  • Naïve $\Theta(n^2)$
  • D/Q Approach: $\Theta(n \log n)$
    – (1) Split into two pointsets $S_1$, $S_2$
      by x-coord $\equiv$ natural order
    – (2) Find closest pair distances $d_1$, $d_2$ in $S_1$, $S_2$ respectively
      without loss of generality, can assume $d_1 < d_2$
    – (3) Merge the solutions
      Do we have to check all $s_1$-$s_2$ pairs, $s_1 \in S_1$, $s_2 \in S_2$?
      What if there are lots of points in middle strip?
    – **Key:** Step (3)
      Observation: There are at most $O(1)$ points in middle strip with $\Delta y \leq d_1$
DQ Closest Pair

- (1) sort by x- and y-coords tool: one-time sort $O(n \log n)$
- (2) solve subproblems of size $n/2$ $2T(n/2)$
- (3) eliminate points outside strips $O(n)$
- (4) “sort” by y-coord within strips $O(n)$
- (5) compare each point to $\leq 6$ neighbors $O(n)$

Complexity
- $O(n \log n) + T(n) = 2T(n/2) + O(n)$
- $T(2) = 1$
- $T(n) \in O(n \log n)$

Again, with one-time sort, can always output subproblems in y-sorted order
Note that we exploited geometry

Example: what is the maximum possible degree of a vertex in a minimum spanning tree?

– If the problem is geometric?  “planar pointset”
– If the problem is non-geometric?  “edge-weighted graph”
Median and Selection

• Fact: The best pivot in QSort is the median
  – It is too expensive to find the median → we use a random element as the pivot instead

• This leads to the problem of **SELECTION**:  
  – Given a list L and a number k  
  – Select (L, k) returns the k\(^{th}\) smallest element of L  
  – e.g., \(k = \frac{|L|}{2}\) → Select(L,k) returns the median

• What is an efficient algorithm?
• Sorting + finding \(k^{th}\) smallest \(\rightarrow O(n \log n)\)
• **CAN DO BETTER**
DQ Selection

- Pick a value $v$
- Split $S$ into three parts: $S_{<v}$, $S_{=v}$, $S_{>v}$

Select $(S,k)$ =

Select($S_{<v},k$) if $k \leq |S_{<v}|$ (look in left)

$v$ if $|S_{<v}| < k \leq |S_{<v}| + |S_{=v}|$

Select($S_{>v},k - |S_{<v}| - |S_{=v}|$) if $k > |S_{<v}| + |S_{=v}|$ (look in right)

- Single split operation effectively reduces size of search space
  - By how much? Depends on $v$ !!!
  - If we could magically pick $v$ so that $|S_{<v}|, |S_{>v}| \approx |S|/2$
    then we would have $T(n) \leq T(n/2) + O(n) \Rightarrow T(n) = O(n)$

“balanced”

Time on array of size $n$  
Time for “Split”
EXTRA SLIDES
Sorting (With Comparisons)

- Input: sequence of numbers
  Output: a sorted sequence
- Observe: Sorting == Identifying a Permutation

KEY OBSERVATIONS
1. We need at least as many leaves in this tree as there are possible outcomes (= n! permutations of n elements)
2. The number of comparisons needed to get to the leaf at greatest depth from the root is the worst-case complexity of the algorithm that is embodied by this comparison tree
Searching an Ordered List

• Input: ordered list $L$ of $n$ numbers, and a target number $x$

Output: Find $x$ if it exists in $L$

• Observe: Searching == Identifying the index of $x$ in $L$

KEY OBSERVATIONS

1. We generally use binary search, which takes $\log(n)$ time in the worst case. Why is binary search the best possible strategy? (*)

2. The number of comparisons needed to get to the leaf at greatest depth from the root is the worst-case complexity of the algorithm that is embodied by this comparison tree

3. Need at least $n$ leaves

4. Height of comparison tree is $\Omega(\log n)$

(*) Interpolation Search is better than binary search when we know something about the distribution of the elements we are searching. For example, when humans use Interpolation Search when looking up something in a dictionary or a phone book.
A Lower Bound on Sorting Complexity

• In the “comparison model of computation”, can we find a lower bound on the complexity of sorting?
• From last slide: Sorting ≡ Identifying Permutation
• Binary Tree: Root at level (height) 0
• **Theorem:**
  – There exists some $c > 0$ such that for all algorithms which use comparisons to sort, and for all input sizes $n$, at least one input requires $cn \log n$ comparisons
• Fact:
  – Binary tree of height $h$ has at most $2^h$ leaves
• Observation from last slide:
  – $n!$ leaves needed $\Rightarrow$ comparison tree must have $h \geq \log(n!)$
  – $h$ is maximum (= worst-case) #comparisons needed to sort input of size $n$ using the corresponding algorithm
Sorting Lower Bound (DETAIL)

• **Goal:** \( \log(n!) \in \Theta(n \log n) \)

• **Claim:** \( \log(n!) \in O(n \log n) \)
   \[
   n! \leq n^n \Rightarrow \log n! \leq n \log n
   \]

• **Claim:** \( \log(n!) \in \Omega(n \log n) \)
   \[
   n! \geq (n/2)^{n/2} \Rightarrow \log n! \geq n/2 \log(n/2)
   \Rightarrow 2 \log n! \geq n \log (n/2)
   \]

**Observe:** \( \log (n/2) + 1 = \log n \)

\[
2 \log (n/2) \geq \log n \quad \forall n > 2
\]

\[
\Rightarrow 4 \log n! \geq n 2 \log (n/2)
\]

\[
\Rightarrow 4 \log n! \geq n \log n
\]
What About LB For the Average Case?

Of possible interest:
http://www.academia.edu/693793/A_simplified_derivation_of_timing_complexity_lower_bounds_for_sorting_by_comparisons

• Worst-case analysis can be uninformative
  – QuickSort: worst case is $n^2$
  – Simplex Method for linear programming: worst case is exponential

• Can we lower-bound “average case” complexity?

• First question: Is “average case” well-defined?
  – Want $\sum p_i d_i \equiv$ expected depth of a leaf in the comparison tree
  – $d_i \equiv$ depth of leaf $i$; $i =$ input with probability $= p_i$
  – Assume all input permutations equally probable (“equiprobable”)
Average-Case Complexity of Sorting

• Q: Even though all sorting algs have some input which requires \( n \log n \) time, is there an algorithm with better than \( (n \log n) \) average-case performance?

• Theorem: If all \( n! \) input permutations equiprobable, then any decision tree that sorts has expected depth \( \Omega(n \log n) \).
  
  – Let \( D(m) \equiv \) smallest sum of leaf depths over all binary trees with \( m \) leaves
  
  – **Claim**: \( D(m) \geq m \log m \).
  
  – If Claim true, use \( m = n! \) and fact that \( \log n! \in \Theta(n \log n) \)
    \( \rightarrow D(n!) \geq n! \log n! \rightarrow \) average leaf depth is \( \Omega(n \log n) \)
Average-Case Complexity of Sorting (cont.)

• **Claim:** \( D(m) \geq m \log m \)
  
  Proof by induction on \( m \).

  \( (D(T) \equiv \text{sum of leaf depths of tree } T, \text{ where unambiguous.}) \)

  Claim trivial for \( m = 1 \); assume Claim \( \forall m < k \) (**strong I.H.**).

  Any tree \( T \) with \( k \) leaves can be viewed as a root and two subtrees \( T_i \) and \( T_{k-i} \) (with \( i \) and \( k-i \) leaves respectively)

  \[ D(T) = i + D(T_i) + (k-i) + D(T_{k-i}) \]

  \[ D(k) = \min_{1 \leq i \leq k} [k + D(i) + D(k-i)] \]

  \[ \geq k + \min_i[D(i) + D(k-i)] \]

  \[ \geq k + \min_i [i \log i + (k-i) \log (k-i)] \quad \text{(**by I.H.**)} \]

  which is minimized for \( i = k/2 \).

  \[ \Rightarrow D(k) \geq k + k \log (k/2) = k + k(\log k - 1) = k \log k \]