COMP 355
Advanced Algorithms

Divide-and-Conquer Algorithms:
Median and Selection
Matrix Multiplication
Hints for HW

• 2 of the problems related to the inversion counting problem
• 1 related to Max Dominance
Max Dominance

Input and initial partition.

Solutions to subproblems.

Merged solution.
Max Dominance

MaxDom3(P, n) {
    Sort P in ascending order by x-coordinate;
    output P[n]; // last point is always maximal
    j = n;
    for i = n-1 downto 1 {
        if (P[i].y >= P[j].y) { // is P[i] maximal?
            output P[i]; // yes...output it
            j = i; // P[i] has the largest y so far
        }
    }
}

MaxDom4(P, n) {
    if (n == 1) return {P[1]}; // one point is trivially maximal
    m = n/2; // midpoint of list
    M1 = MaxDom4(P[1..m], m); // solve for first half
    M2 = MaxDom4(P[m+1..n], n-m); // solve for second half
    return MaxMerge(M1, M2); // merge the results
}
Suppose you are given a set of n numbers.

Def. The rank of an element is $1 + \#$ of elements that are smaller than that element.

Simplifying assumption: No duplicates in the set.

If set of numbers is sorted, the rank of an element is its final position. (Minimum rank: 1, Maximum rank: n)
Of particular interest in statistics is the median. Once the list is sorted, it is trivial to find the median.

If $n$ is odd:
- Median = element of rank $(n + 1)/2$

If $n$ is even:
- Median = element of rank $n/2$ or $n/2 + 1$.
- Common to take average of these in statistics; we will consider the median to be either of these elements.
Median

• Central tendency of a set
  – Better metric than average when data values are highly skewed
  – Example: Bill Gates moves into your neighborhood, his gigantic income may significantly bias the average, whereas it will have very little influence on the median

• Divide-and-conquer algorithms often want to partition on the median value to split a set into two sets of roughly equal size

• Generalization of the median problem is called the selection problem.
Selection

• Related problem to sorting is selection
• Given an array A of n numbers (not sorted) and an integer k, where $1 \leq k \leq n$, return the kth smallest value of A.

• Easy algorithm:
  – Sort the array ($\Theta (n \log n)$)
  – Return kth element

• Harder algorithm
  – $O(n)$
  – Variant of QuickSort
The Sieve Technique

• This algorithm illustrates a very important case of divide-and-conquer, which we’ll call the sieve technique.
• Instead of dividing a problem into a small number of sub-problems, the sieve technique only uses 1 sub-problem.
• Applies to problems where we are interested in finding a single item from a larger set of \( n \) items.
• Works in phases:
  – After doing some amount of analysis of the data, taking say \( O(n^k) \) time, for some constant \( k \), we find that we do not know what the desired item is, but we can identify a large enough number of elements that cannot be the desired value.
  – Solve the problem recursively on whatever items remain.
Applying the Sieve to Selection

Choose a *pivot* element from A  
Partition A into 3 subarrays:  
- $A[q] = \text{pivot element (x)}$  
- $A[p...q-1] = \text{all elements of } A < x$  
- $A[q+1..r] = \text{all elements of } A > x$  
- Note: *Subarrays may appear in any order*  

Rank of the pivot $x$ is $q-p + 1$  
If $k < x\text{Rank}$, recursively search for rank $k$ in $A[p..q-1]$  
Else If $k > x\text{Rank}$, recursively search for rank $k$ in $A[q+1...r]$  
Else: $x\text{Rank} = k$ and we’re done.
Selection Algorithm

```
Select(array A, int p, int r, int k) { // return kth smallest of A[p..r]
    if (p == r) return A[p] // only 1 item left, return it
    else {
        x = ChoosePivot(A, p, r) // choose the pivot element
        q = Partition(A, p, r, x) // <A[p..q-1], x, A[q+1..r]>
        xRank = q - p + 1 // rank of the pivot
        if (k == xRank) return x // the pivot is the kth smallest
        else if (k < xRank)
            return Select(A, p, q-1, k) // select from left
        else
            return Select(A, q+1, r, k-xRank) // select from right
    }
}
```
Selection Example

Fig. 86: Selection Algorithm.
Quickselect demo

3–way partition array so that:

• Pivot element \( p \) is in place.

• Equal elements in middle subarray \( M \).

Recur in one subarray—the one containing the \( k \)th smallest element.

select the \( k = 8 \)th smallest

\[
\begin{array}{cccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
65 & 28 & 59 & 33 & 21 & 56 & 22 & 95 & 50 & 12 & 90 & 53 & 28 & 77 & 39 \\
\end{array}
\]

\( k = 8 \)th smallest
Quickselect demo

3–way partition array so that:

- Pivot element $p$ is in place.
- Equal elements in middle subarray $M$.

Recur in one subarray—the one containing the $k^{\text{th}}$ smallest element.

choose a pivot element at random and partition

\[
\begin{array}{ccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
65 & 28 & 59 & 33 & 21 & 56 & 22 & 95 & 50 & 12 & 90 & 53 & 28 & 77 & 39 \\
\end{array}
\]

$k = 8^{\text{th}}$ smallest
Quickselect demo

3–way partition array so that:

- Pivot element \( p \) is in place.
- Equal elements in middle subarray \( M \).

Recur in one subarray—the one containing the \( k^{th} \) smallest element.

**partitioned array**

\[
\begin{array}{cccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
28 & 33 & 21 & 56 & 22 & 50 & 12 & 53 & 28 & 39 & 59 & 65 & 95 & 90 & 77 \\
k = 8^{th} \text{ smallest}
\end{array}
\]
Quickselect demo

3–way partition array so that:

- Pivot element $p$ is in place.
- Equal elements in middle subarray $M$.

Recur in one subarray—the one containing the $k^{th}$ smallest element.

Recursively select 8th smallest element in left subarray

\[
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
28 & 33 & 21 & 56 & 22 & 50 & 12 & 53 & 28 & 39 & 59 & 65 & 95 & 90 & 77 \\
\end{array}
\]

$k = 8^{th}$ smallest
Quickselect demo

3-way partition array so that:

- Pivot element $p$ is in place.
- Equal elements in middle subarray $M$.

Recur in one subarray—the one containing the $k^{th}$ smallest element.

choose a pivot element at random and partition

choose a pivot element at random and partition

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>33</td>
<td>21</td>
<td>56</td>
<td>22</td>
<td>50</td>
<td>12</td>
<td>53</td>
<td>28</td>
<td>39</td>
<td>59</td>
<td>65</td>
<td>95</td>
<td>90</td>
<td>77</td>
</tr>
</tbody>
</table>

$k = 8^{th}$ smallest
Quickselect demo

3–way partition array so that:

• Pivot element $p$ is in place.

• Equal elements in middle subarray $M$.

Recur in one subarray—the one containing the $k^{th}$ smallest element.

partitioned array

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>22</td>
<td>12</td>
<td>28</td>
<td>28</td>
<td>33</td>
<td>56</td>
<td>50</td>
<td>53</td>
<td>39</td>
<td>59</td>
<td>65</td>
<td>95</td>
<td>90</td>
<td>77</td>
</tr>
</tbody>
</table>

$k = 8^{th}$ smallest
Quickselect demo

3–way partition array so that:

- Pivot element $p$ is in place.
- Equal elements in middle subarray $M$.

Recur in one subarray—the one containing the $k^{th}$ smallest element.

```plaintext
recursively select the 3rd smallest element in right subarray

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
21 22 12 28 28 33 56 50 53 39 59 65 95 90 77

k = 3rd smallest
```
Quickselect demo

3–way partition array so that:

- Pivot element $p$ is in place.
- Equal elements in middle subarray $M$.

Recur in one subarray—the one containing the $k^{th}$ smallest element.

Choose a pivot element at random and partition

\[ \begin{array}{ccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
21 & 22 & 12 & 28 & 28 & 33 & 56 & 50 & 53 & 39 & 59 & 65 & 95 & 90 & 77 \\
\end{array} \]

$k = 3^{rd}$ smallest
Quickselect demo

3-way partition array so that:

- Pivot element $p$ is in place.
- Equal elements in middle subarray $M$.

Recur in one subarray—the one containing the $k^{th}$ smallest element.

partitioned array

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>22</td>
<td>12</td>
<td>28</td>
<td>28</td>
<td>33</td>
<td>39</td>
<td>50</td>
<td>53</td>
<td>56</td>
<td>59</td>
<td>65</td>
<td>95</td>
<td>90</td>
<td>77</td>
</tr>
</tbody>
</table>

$k = 3^{rd}$ smallest
Quickselect demo

3–way partition array so that:

- Pivot element $p$ is in place.
- Equal elements in middle subarray $M$.

Recur in one subarray—the one containing the $k^{th}$ smallest element.

stop: desired element is in middle subarray
Selection Algorithm Analysis

If we (optimistically) choose our pivot element to approximately divide our n numbers into 2 sets of n/2, we get:

\[ T(n) = \begin{cases} 
1 & \text{if } n = 1, \\
T(n/2) + n & \text{otherwise.}
\end{cases} \]

Solve by the Master Theorem:

\( a = 1, \ b = 2, \ d = 1 \), so case 1, and we get \( T(n) = \Theta(n) \)

How do we choose a pivot to ensure dividing the set evenly though?
Choosing a Pivot

Groups of 5: Partition $A$ into groups of 5 elements, e.g. $A[1..5]$, $A[6..10]$, $A[11..15]$, etc. There will be exactly $m = \lceil n/5 \rceil$ such groups (the last one might have fewer than 5 elements). This can easily be done in $\Theta(n)$ time.

Group medians: Compute the median of each group of 5. There will be $m$ group medians. We do not need an intelligent algorithm to do this, since each group has only a constant number of elements. For example, we could just BubbleSort each group and take the middle element. Each will take $\Theta(1)$ time, and repeating this $\lceil n/5 \rceil$ times will give a total running time of $\Theta(n)$. Copy the group medians to a new array $B$.

Median of medians: Compute the median of the group medians. For this, we will have to call the selection algorithm recursively on $B$, e.g. $\text{Select}(B, 1, m, k)$, where $m = \lceil n/5 \rceil$, and $k = \lfloor (m + 1)/2 \rfloor$. Let $x$ be this median of medians. Return $x$ as the desired pivot.
Choosing the Pivot: Example

Fig. 87: Choosing the Pivot. 30 is the final pivot.
Proof of Correctness

Lemma: The element \( x \) is of rank at least \( n/4 \) and at most \( 3n/4 \) in \( A \).

Proof: We will show that \( x \) is of rank at least \( n/4 \). (Use symmetry to proof other part).

Need to show that there are at least \( n/4 \) elements that are less than or equal to \( x \).

Simplifying Assumption: \( n \) is evenly divisible by 5.

• At least half the group medians \( \leq x \) (because \( x \) is their median)

• For each group median, there are \( \leq 3 \) elements that are \( \leq \) this median (because it is a median)

• Therefore, there are at least \( 3((n/5)/2 = 3n/10 \geq n/4 \) elements that are less than or equal to \( x \) in the entire array.
Analysis of Choose Pivot

- We achieved the main goal, namely that of eliminating a constant fraction (at least 1/4) of the remaining list at each stage of the algorithm.
- The recursive call in Select() will be made to list no larger than $3n/4$.
- In order to achieve this, within Select Pivot() we needed to make a recursive call to Select() on an array B consisting of $\lceil n/5 \rceil$ elements.
- Everything else took only (n) time.

$$T(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ T(n/5) + T(3n/4) + n & \text{otherwise.} \end{cases}$$
Google Interview Question

Given two sorted arrays with N elements each, find the median of their union in O(log n).
Matrix Multiplication

Matrix multiplication. Given two n-by-n matrices A and B, compute C = AB.

\[ c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \]

Brute force. \( \Theta(n^3) \) arithmetic operations.

Fundamental question. Can we improve upon brute force?
**Matrix Multiplication: Warmup**

Divide-and-conquer.

- **Divide**: partition $A$ and $B$ into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- **Conquer**: multiply $8$ $\frac{1}{2}n$-by-$\frac{1}{2}n$ recursively.
- **Combine**: add appropriate products using 4 matrix additions.

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} =
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \times
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

\[
C_{11} = (A_{11} \times B_{11}) + (A_{12} \times B_{21})
\]
\[
C_{12} = (A_{11} \times B_{12}) + (A_{12} \times B_{22})
\]
\[
C_{21} = (A_{21} \times B_{11}) + (A_{22} \times B_{21})
\]
\[
C_{22} = (A_{21} \times B_{12}) + (A_{22} \times B_{22})
\]

\[
T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)
\]

\[
\Rightarrow T(n) = \Theta(n^3)
\]
Matrix Multiplication: Key Idea

Key idea. Multiply 2-by-2 block matrices with only 7 multiplications.

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} =
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \times
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

\[
C_{11} = P_3 + P_4 - P_2 + P_6
\]
\[
C_{12} = P_1 + P_2
\]
\[
C_{21} = P_3 + P_4
\]
\[
C_{22} = P_5 + P_1 - P_3 - P_7
\]

\[
\begin{align*}
P_1 &= A_{11} \times (B_{12} - B_{22}) \\
P_2 &= (A_{11} + A_{12}) \times B_{22} \\
P_3 &= (A_{21} + A_{22}) \times B_{11} \\
P_4 &= A_{22} \times (B_{21} - B_{11}) \\
P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\
P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\
P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12})
\end{align*}
\]

- 7 multiplications.
- 18 = 10 + 8 additions (or subtractions).
Fast Matrix Multiplication

Fast matrix multiplication. (Strassen, 1969)

• Divide: partition A and B into \( \frac{1}{2}n \)-by-\( \frac{1}{2}n \) blocks.
• Compute: 14 \( \frac{1}{2}n \)-by-\( \frac{1}{2}n \) matrices via 10 matrix additions.
• Conquer: multiply 7 \( \frac{1}{2}n \)-by-\( \frac{1}{2}n \) matrices recursively.
• Combine: 7 products into 4 terms using 8 matrix additions.

Analysis.

• Assume \( n \) is a power of 2.
• \( T(n) = \# \) arithmetic operations.

\[
T(n) = 7T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})
\]
Next Time

• Dynamic Programming Algorithms!