CS 341: Algorithms

Douglas R. Stinson

David R. Cheriton School of Computer Science
University of Waterloo

January 15, 2019
1. Course Information
2. Introduction
3. Divide-and-Conquer Algorithms
4. Greedy Algorithms
5. Dynamic Programming Algorithms
6. Graph Algorithms
7. Intractability and Undecidability
Table of Contents

3 Divide-and-Conquer Algorithms
- Recurrence Relations
- Master Theorem
- Divide-and-Conquer Design Strategy
- Mergesort
- Divide-and-Conquer Recurrence Relations
- Non-dominated Points
- Closest Pair
- Multiprecision Multiplication
- Matrix Multiplication
- Selection and Median
Recurrence Relations

Suppose $a_1, a_2, \ldots$, is an infinite sequence of real numbers.

A recurrence relation is a formula that expresses a general term $a_n$ in terms of one or more previous terms $a_1, \ldots, a_{n-1}$.

A recurrence relation will also specify one or more initial values starting at $a_1$.

Solving a recurrence relation means finding a formula for $a_n$ that does not involve any previous terms $a_1, \ldots, a_{n-1}$.

There are many methods of solving recurrence relations. Two important methods are guess-and-check and the recursion tree method.

We will make extensive use of the recursion tree method. However, we first take a quick look at the guess-and-check method.
Guess-and-check Method

step 1  Tabulate some values $a_1, a_2, \ldots$ using the recurrence relation.

step 2  Guess that the solution $a_n$ has a specific form, involving undetermined constants.

step 3  Use $a_1, a_2, \ldots$ to determine specific values for the unspecified constants.

step 4  Use induction to prove your guess for $a_n$ is correct.
Example of the Guess-and-check Method

Suppose we have the recurrence $T(n) = T(n - 1) + 6n - 5$, $T(0) = 4$. We compute a few values: $T(1) = 5$, $T(2) = 12$, $T(3) = 25$, $T(4) = 44$.

If we are sufficiently perspicacious, we might guess that $T(n)$ is a quadratic function, e.g., $T(n) = an^2 + bn + c$.

Next, we use $T(0) = 4$, $T(1) = 5$, $T(2) = 12$ to compute $a$, $b$ and $c$ by solving three equations in three unknowns.

We get $a = 3$, $b = -2$, $c = 4$.

Now we can use induction to prove that $T(n) = 3n^2 - 2n + 4$ for all $n \geq 0$. 

Another Example

Consider the recurrence

\[ T(n) = T(\lfloor n/2 \rfloor) + T(\lfloor n/3 \rfloor) + n \]

\[ T(1) = 1 \]
\[ T(2) = 2 \]

Suppose we tabulate some values of \( T(n) \) and then guess that \( T(n) \leq cn \) for all \( n \geq 1 \), for some constant \( c \).

We can use empirical data to guess an appropriate value for \( c \).

However, an alternative approach it to carry out the induction proof in order to determine a value of \( c \) that works.
Recursion Tree Method

The following recurrence relation arises in the analysis of Mergesort:

\[
T(n) = \begin{cases} 
2T\left(\frac{n}{2}\right) + cn & \text{if } n > 1 \text{ is a power of } 2 \\
d & \text{if } n = 1,
\end{cases}
\]

where \( c \) and \( d \) are constants.

We can solve this recurrence relation when \( n \) is a power of two, by constructing a recursion tree, as follows:

\begin{enumerate}
  \item \textbf{step 1} Start with a one-node tree, say \( N \), having the value \( T(n) \).
  \item \textbf{step 2} Grow two children of \( N \). These children, say \( N_1 \) and \( N_2 \), have the value \( T(n/2) \), and the value of \( N \) is replaced by \( cn \).
  \item \textbf{step 3} Repeat this process recursively, terminating when a node receives the value \( T(1) = d \).
  \item \textbf{step 4} Sum the values on each level of the tree, and then compute the sum of all these sums; the result is \( T(n) \).
\end{enumerate}
Solving the Mergesort Recurrence

Let $n = 2^j$.

<table>
<thead>
<tr>
<th>level</th>
<th># nodes</th>
<th>value at each node</th>
<th>value of the level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j$</td>
<td>1</td>
<td>$c2^j$</td>
<td>$c2^j$</td>
</tr>
<tr>
<td>$j-1$</td>
<td>2</td>
<td>$c2^{j-1}$</td>
<td>$c2^j$</td>
</tr>
<tr>
<td>$j-2$</td>
<td>$2^2$</td>
<td>$c2^{j-2}$</td>
<td>$c2^j$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>$2^{j-1}$</td>
<td>$c2^1$</td>
<td>$c2^j$</td>
</tr>
<tr>
<td>0</td>
<td>$2^j$</td>
<td>$d$</td>
<td>$d2^j$</td>
</tr>
</tbody>
</table>

Summing the values at all levels of the recursion tree, we have that

$$T(n) = d2^j + cj2^j.$$ 

Since $n = 2^j$, we have $j = \log_2 n$ and

$$T(n) = dn + cn \log_2 n \in \Theta(n \log n).$$
Another Example

Recall the recurrence $T(n) = T([n/2]) + T([n/3]) + n$. We showed by induction that $T(n) \in O(n)$.

Here, we give an informal justification of this result (not a proof) using the recurrence tree method.

We ignore all “floors”, and compute the sum of the all the levels of the tree.
Master Theorem

The Master Theorem provides a formula for the solution of many recurrence relations typically encountered in the analysis of algorithms. The following is a simplified version of the Master Theorem:

**Theorem 3.1**

Suppose that $a \geq 1$ and $b > 1$. Consider the recurrence

$$T(n) = aT \left( \frac{n}{b} \right) + \Theta(n^y),$$

(1)

where $n$ is a power of $b$. Denote $x = \log_b a$. Then

$$T(n) \in \begin{cases} 
    \Theta(n^x) & \text{if } y < x \\
    \Theta(n^x \log n) & \text{if } y = x \\
    \Theta(n^y) & \text{if } y > x.
\end{cases}$$
Proof of the Master Theorem (simplified version)

Suppose that $a \geq 1$ and $b \geq 2$ are integers and

$$T(n) = aT\left(\frac{n}{b}\right) + cn^y, \quad T(1) = d.$$  

Let $n = b^j$.

<table>
<thead>
<tr>
<th>level</th>
<th># nodes</th>
<th>value at each node</th>
<th>value of the level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j$</td>
<td>1</td>
<td>$cn^y$</td>
<td>$cn^y$</td>
</tr>
<tr>
<td>$j-1$</td>
<td>$a$</td>
<td>$c(n/b)^y$</td>
<td>$ca(n/b)^y$</td>
</tr>
<tr>
<td>$j-2$</td>
<td>$a^2$</td>
<td>$c(n/b^2)^y$</td>
<td>$ca^2(n/b^2)^y$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>1</td>
<td>$a^{j-1}$</td>
<td>$c(n/b^{j-1})^y$</td>
<td>$ca^{j-1}(n/b^{j-1})^y$</td>
</tr>
<tr>
<td>0</td>
<td>$a^j$</td>
<td>$d$</td>
<td>$da^j$</td>
</tr>
</tbody>
</table>
Computing $T(n)$

Summing the values at all levels of the recursion tree, we have that

$$T(n) = da^j + cn^y \sum_{i=0}^{j-1} \left( \frac{a}{by} \right)^i.$$

Recall that $b^x = a$ and $n = b^j$. Hence $a^j = (b^x)^j = (b^j)^x = n^x$.

The formula for $T(n)$ is a geometric sequence with ratio $r = \frac{a}{by} = b^{x-y}$:

$$T(n) = dn^x + cn^y \sum_{i=0}^{j-1} r^i.$$

There are three cases, depending on whether $r > 1$, $r = 1$ or $r < 1$. 
Complexity of $T(n)$

<table>
<thead>
<tr>
<th>case</th>
<th>$r$</th>
<th>$y, x$</th>
<th>complexity of $T(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>heavy leaves</td>
<td>$r &gt; 1$</td>
<td>$y &lt; x$</td>
<td>$T(n) \in \Theta(n^x)$</td>
</tr>
<tr>
<td>balanced</td>
<td>$r = 1$</td>
<td>$y = x$</td>
<td>$T(n) \in \Theta(n^x \log n)$</td>
</tr>
<tr>
<td>heavy top</td>
<td>$r &lt; 1$</td>
<td>$y &gt; x$</td>
<td>$T(n) \in \Theta(n^y)$</td>
</tr>
</tbody>
</table>

**heavy leaves** means that the value of the recursion tree is dominated by the values of the leaf nodes.

**balanced** means that the values of the levels of the recursion tree are constant (except for the last level).

**heavy top** means that the value of the recursion tree is dominated by the value of the root node.
Complexity of $T(n)$ (cont.)

Let

$$S = \sum_{i=0}^{j-1} r^i.$$ 

In case 1, we have $x > y$ so $r > 1$. $S \in \Theta(r^j)$, so $T(n) \in \Theta(n^x + n^y r^j)$. However,

$$r^j = (b^{x-y})^j = (b^j)^{x-y} = n^{x-y}.$$

Therefore

$$T(n) \in \Theta(n^x + n^y n^{x-y}) = \Theta(n^x).$$

In case 2, we have $x = y$ so $r = 1$. $S \in \Theta(j) = \Theta(\log n)$, so

$$T(n) \in \Theta(n^x + n^y \log n) = \Theta(n^x + n^x \log n) = \Theta(n^x \log n).$$

In case 3, we have $x < y$ so $r < 1$. $S \in \Theta(1)$, so

$$T(n) \in \Theta(n^x + n^y) = \Theta(n^y).$$

The complexity does not depend on the initial value $d$. 

Some Examples of Applying the Formulas

1. \( T(n) = 2T(n/2) + cn. \)

2. \( T(n) = 3T(n/2) + cn. \)

3. \( T(n) = 4T(n/2) + cn. \)

4. \( T(n) = 2T(n/2) + cn^{3/2}. \)
Master Theorem (modified general version)

Theorem 3.2

Suppose that \( a \geq 1 \) and \( b > 1 \). Consider the recurrence

\[
T(n) = aT\left(\frac{n}{b}\right) + f(n),
\]

where \( n \) is a power of \( b \). Denote \( x = \log_b a \). Then

\[
T(n) \in \begin{cases} 
\Theta(n^x) & \text{if } f(n) \in O(n^{x-\epsilon}) \text{ for some } \epsilon > 0 \\
\Theta(n^x \log n) & \text{if } f(n) \in \Theta(n^x) \\
\Theta(f(n)) & \text{if } f(n)/n^{x+\epsilon} \text{ is an increasing function of } n \\
\end{cases}
\]

for some \( \epsilon > 0 \).
Some Examples

1. \[ T(n) = 3T(n/4) + n \log n. \]

2. \[ T(n) = 2T(n/2) + n \log n. \]
Solving the Second Recurrence

We can solve the above \( T(n) = 2T(n/2) + n \log n \) using the recursion tree method. Assume \( T(1) = 1 \). Let \( n = 2^j \).

<table>
<thead>
<tr>
<th>level ( j )</th>
<th># nodes</th>
<th>value at each node</th>
<th>value of the level</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j )</td>
<td>1</td>
<td>( j2^j )</td>
<td>( j2^j )</td>
</tr>
<tr>
<td>( j - 1 )</td>
<td>2</td>
<td>( (j - 1)2^{j-1} )</td>
<td>( (j - 1)2^j )</td>
</tr>
<tr>
<td>( j - 2 )</td>
<td>( 2^2 )</td>
<td>( (j - 2)2^{j-2} )</td>
<td>( (j - 2)2^j )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>( 2^{j-1} )</td>
<td>( 2^1 )</td>
<td>( 2^j )</td>
</tr>
<tr>
<td>0</td>
<td>( 2^j )</td>
<td>1</td>
<td>( 2^j )</td>
</tr>
</tbody>
</table>

Summing the values at all levels of the recursion tree, we have that

\[
T(n) = 2^j \left( 1 + \sum_{i=1}^{j} i \right) = 2^j \left( 1 + \frac{j(j + 1)}{2} \right).
\]

Since \( n = 2^j \), we have \( j = \log_2 n \) and \( T(n) \in \Theta(n(\log n)^2) \).
The Divide-and-Conquer Design Strategy

**divide:** Given a problem instance $I$, construct one or more smaller problem instances, denoted $I_1, \ldots, I_a$ (these are called subproblems). Usually, we want the size of these subproblems to be small compared to the size of $I$, e.g., half the size.

**conquer:** For $1 \leq j \leq a$, solve instance $I_j$ recursively, obtaining solutions $S_1, \ldots, S_a$.

**combine:** Given $S_1, \ldots, S_a$, use an appropriate combining function to find the solution $S$ to the problem instance $I$, i.e., $S \leftarrow \text{Combine}(S_1, \ldots, S_a)$. 
Example: Design of Mergesort

Here, a problem instance consists of an array $A$ of $n$ integers, which we want to sort in increasing order. The size of the problem instance is $n$.

**divide:** Split $A$ into two subarrays: $A_L$ consists of the first $\lceil \frac{n}{2} \rceil$ elements in $A$ and $A_R$ consists of the last $\lfloor \frac{n}{2} \rfloor$ elements in $A$.

**conquer:** Run *Mergesort* on $A_L$ and $A_R$.

**combine:** After $A_L$ and $A_R$ have been sorted, use a function *Merge* to merge $A_L$ and $A_R$ into a single sorted array. Recall that this can be done in time $\Theta(n)$ with a single pass through $A_L$ and $A_R$. We simply keep track of the “current” element of $A_L$ and $A_R$, always copying the smaller one into the sorted array.
Mergesort

Algorithm: \textit{Mergesort}(A : array; n : integer)

\begin{align*}
\text{if } n = 1 & \quad \text{then } S \leftarrow A \\
\phantom{\text{if } n = 1} & \quad \text{else} \\
\phantom{\text{if } n = 1} & \quad \begin{cases} \\
\phantom{\text{else } n} & \quad n_L \leftarrow \left\lfloor \frac{n}{2} \right\rfloor \\
\phantom{\text{else } n} & \quad n_R \leftarrow \left\lceil \frac{n}{2} \right\rceil \\
\phantom{\text{else } n} & \quad A_L \leftarrow [A[1], \ldots, A[n_L]] \\
\phantom{\text{else } n} & \quad A_R \leftarrow [A[n_L + 1], \ldots, A[n]] \\
\end{cases} \\
\phantom{\text{else } n} & \quad S_L \leftarrow \text{Mergesort}(A_L, n_L) \\
\phantom{\text{else } n} & \quad S_R \leftarrow \text{Mergesort}(A_R, n_R) \\
\phantom{\text{else } n} & \quad S \leftarrow \text{Merge}(S_L, n_L, S_R, n_R) \\
\end{align*}

\text{return } (S, n)
Analysis of Mergesort

Let $T(n)$ denote the time to run Mergesort on an array of length $n$.

**divide** takes time $\Theta(n)$

**conquer** takes time $T\left(\lceil \frac{n}{2} \rceil \right) + T\left(\lfloor \frac{n}{2} \rfloor \right)$

**combine** takes time $\Theta(n)$

Recurrence relation:

$$T(n) = \begin{cases} 
T\left(\lceil \frac{n}{2} \rceil \right) + T\left(\lfloor \frac{n}{2} \rfloor \right) + \Theta(n) & \text{if } n > 1 \\
\Theta(1) & \text{if } n = 1.
\end{cases}$$
Sloppy and Exact Recurrence Relations

It is simpler to replace the $\Theta(n)$ term by $cn$, where $c$ is an unspecified constant. The resulting recurrence relation is called the exact recurrence.

$$T(n) = \begin{cases} T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lfloor \frac{n}{2} \rfloor\right) + cn & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

If we then remove the floors and ceilings, we obtain the so-called sloppy recurrence:

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + cn & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

The exact and sloppy recurrences are identical when $n$ is a power of two. Further, the sloppy recurrence makes sense only when $n$ is a power of two.
Solution to the Recurrence

The **Master Theorem** provides the **exact** solution of the recurrence when \( n = 2^j \) (it is in fact a **proof** for these values of \( n \)).

We can express this solution (for powers of 2) as a function of \( n \), using \( \Theta \)-notation.

It can be shown that the resulting function of \( n \) will in fact yield the **complexity** of the solution of the exact recurrence for **all values** of \( n \).

This derivation of the complexity of \( T(n) \) is **not a proof**, however. If a rigorous mathematical proof is required, then it is necessary to use **induction** along with the **exact recurrence**.
Non-dominated Points

Given two points \((x_1, y_1), (x_2, y_2)\) in the Euclidean plane, we say that \((x_1, y_1)\) dominates \((x_2, y_2)\) if \(x_1 > x_2\) and \(y_1 > y_2\).

Problem 3.3

Non-dominated Points

**Instance:** A set \(S\) of \(n\) points in the Euclidean plane, say \(S = \{S[1], \ldots, S[n]\}\). For simplicity, we will assume that the \(x\)-co-ordinates of all these points are distinct, and the the \(y\)-co-ordinates of all these points are also distinct.

**Question:** Find all the non-dominated points in \(S\), i.e., all the points that are not dominated by any other point in \(S\).

Non-dominated Points has a trivial \(\Theta(n^2)\) algorithm to solve it, based on comparing all pairs of points in \(S\). Can we do better?
Staircases

Observe that the non-dominated points form a **staircase** and all the other points are “under” this staircase.

The **treads** of the staircase are determined by the $y$-co-ordinates of the non-dominated points. The **risers** of the staircase are determined by the $x$-co-ordinates of the non-dominated points. The staircase descends from left to right.
**Problem Decomposition**

Suppose we **pre-sort** the points in $S$ with respect to their $x$-co-ordinates. This takes time $\Theta(n \log n)$.

**Divide:** Let the first $n/2$ points be denoted $S_1$ and let the last $n/2$ points be denoted $S_2$.

**Conquer:** Recursively solve the subproblems defined by the two instances $S_1$ and $S_2$.

**Combine:** Given the non-dominated points in $S_1$ and the non-dominated points in $S_2$, how do we find the non-dominated points in $S$?

Observe that **no point in $S_1$ dominates a point in $S_2$**.

Therefore we only need to eliminate the points in $S_1$ that are dominated by a point in $S_2$. It turns out that this can be done in time $O(n)$. 
The Combine Step

We compute $k$ to be the maximum $i$ such that

$$\text{the } y\text{-co-ordinate of } Q_i \text{ is } > \text{ the } y\text{-co-ordinate of } R_1.$$  

This is just a linear search. (We could actually do a binary search, but the overall complexity will not be affected.)

Then, $\text{Combine}(\text{ND}(S_1), \text{ND}(S_2)) = (Q_1, \ldots, Q_k, R_1, \ldots, R_m)$.

The $x$-co-ordinates of the points in $\text{Combine}(\text{ND}(S_1), \text{ND}(S_2))$ are in increasing order, so this can be regarded as a post-condition of the algorithm.
Non-dominated Points

Algorithm: \textit{Non-dominated}(S_1, \ldots, S_n)

\textbf{comment:} these \(n\) points are in increasing order WRT their \(x\)-co-ordinates

\textbf{if} \(n = 1\) \textbf{then return} \(S[1]\)

\textbf{else}

\begin{align*}
(Q[1], \ldots, Q[\ell]) & \leftarrow \text{Non-dominated}(S[1], \ldots, S[\lfloor n/2 \rfloor]) \\
(R[1], \ldots, R[m]) & \leftarrow \text{Non-dominated}(S[\lceil n/2 \rceil + 1], \ldots, S[n])
\end{align*}

\textbf{else}

\begin{align*}
i & \leftarrow 1 \\
\text{while } i \leq \ell \text{ and } Q[i].y > R[1].y & \text{ do } i \leftarrow i + 1
\end{align*}

\textbf{return} \((Q[1], \ldots, Q[i-1], R[1], \ldots, R[m])\)

\textbf{comment:} these points are in increasing order WRT their \(x\)-co-ordinates
Closest Pair

Problem 3.4

Closest Pair

Instance: a set $Q$ of $n$ distinct points in the Euclidean plane,

$$Q = \{Q[1], \ldots, Q[n]\}.$$

Find: Two distinct points $Q[i] = (x, y), Q[j] = (x', y')$ such that the Euclidean distance

$$\sqrt{(x' - x)^2 + (y' - y)^2}$$

is minimized.
Closest Pair: Problem Decomposition

Suppose we pre-sort the points in $Q$ with respect to their $x$-coordinates. Then we can easily find the vertical line that partitions the set of points $Q$ into two sets of size $n/2$: this line has equation $x = Q[m].x$, where $m = n/2$.

**Divide:** We have two subproblems, consisting of the first $n/2$ points and the last $n/2$ points.

**Conquer:** Recursively solve the two subproblems.

**Combine:** Given that we have determined the shortest distance among the first $n/2$ points and the shortest distance among the last $n/2$ points, what additional work is required to determine the overall shortest distance?
Problem Decomposition (cont.)

We will construct the critical strip $R$ of width $2\delta$ consisting of all points whose $x$-coordinates are within $\delta$ of the vertical splitting line, which has equation $x = x_{mid}$, where $x_{mid} = Q[m].x$.

Suppose $\delta_L$ is the minimum distance in the left half, $\delta_R$ is the minimum distance in the right half. Let $\delta = \min\{\delta_L, \delta_R\}$.

If there is a pair of points having distance $< \delta$, they must be in the critical strip.

Perhaps all the points are in the critical strip, so it will not be efficient to check all pairs of points in the critical strip ($n/2 \times n/2 = n^2/4 \in \Theta(n^2)$).

Key idea: Sort the points in the critical strip WRT $y$-co-ordinates. This takes time $\Theta(n \log n)$.

It turns out that we only need to compute distances from each point to the next seven points This means that there are at most $7n$ pairs of points to check, which can be done in time $\Theta(n)$. 
The Critical Strip

Lemma 3.5

Suppose the points in the critical strip are sorted WRT their $y$-co-ordinates. Suppose that $R[j]$ and $R[k]$ are two points in the critical strip, where $j < k$, and suppose the distance between $R[j]$ and $R[k]$ is less than $\delta$. Then $k \leq j + 7$.

Proof.

Construct a rectangle $R$ having width $2\delta$ and height $\delta$, in which the base is the line $y = R[j].y$. Consider $R$ to be partitioned into eight squares of side $\delta/2$. There is at most one point inside each of these eight squares, one of which is $R[j]$. If $k \geq j + 8$, then $R[k]$ is above $R$ and the distance between $R[j]$ and $R[k]$ is greater than $\delta$. 

☐
Closest Pair: Solution 1

Algorithm: \( \text{ClosestPair1}(\ell, r) \)

\[
\begin{align*}
\text{if } \ell &= r \quad \text{then} \quad &\delta &\leftarrow \infty \\
& m \leftarrow \lfloor (\ell + r)/2 \rfloor \\
& \delta_L \leftarrow \text{ClosestPair1}(\ell, m) \\
& \delta_R \leftarrow \text{ClosestPair1}(m + 1, r) \\
\text{else} & \begin{cases} 
\delta &\leftarrow \min\{\delta_L, \delta_R\} \\
R &\leftarrow \text{SelectCandidates}(\ell, r, \delta, Q[m].x) \\
R &\leftarrow \text{SortY}(R) \\
\delta &\leftarrow \text{CheckStrip}(R, \delta) 
\end{cases}
\end{align*}
\]

\text{return (}\delta\text{)}
Selecting Candidates from the Vertical Strip

Algorithm: \textit{SelectCandidates}(\ell, r, \delta, x_{\text{mid}})

\begin{align*}
j &\leftarrow 0 \\
\text{for } i &\leftarrow \ell \text{ to } r \\
\text{do } & \begin{cases} 
\text{if } |Q[i].x - x_{\text{mid}}| \leq \delta \\
\text{then} & \\
& \begin{cases} 
 j &\leftarrow j + 1 \\
 R[j] &\leftarrow Q[i]
\end{cases}
\end{cases} \\
\text{return } (R)
\end{align*}
Checking the Vertical Strip

Algorithm: \textit{CheckStrip}(R, \delta)

\begin{align*}
t & \leftarrow \text{size}(R) \\
\delta' & \leftarrow \delta \\
\text{for } j & \leftarrow 1 \text{ to } t - 1 \\
\quad & \text{for } k \leftarrow j + 1 \text{ to } \min\{t, j + 7\} \\
\quad & \quad \text{do} \\
\quad & \quad \quad \text{do} \\
\quad & \quad \quad \quad x \leftarrow R[j].x \\
\quad & \quad \quad \quad x' \leftarrow R[k].x \\
\quad & \quad \quad \quad y \leftarrow R[j].y \\
\quad & \quad \quad \quad y' \leftarrow R[k].y \\
\quad & \quad \quad \quad \delta' \leftarrow \min \left\{ \delta', \sqrt{(x' - x)^2 + (y' - y)^2} \right\} \\
\quad & \quad \text{do} \\
\quad & \text{return } (\delta')
\end{align*}
An Improvement

To improve the complexity, we eliminate the sorting of the points in critical strip WRT their $y$-co-ordinates.

The precondition for $ClosestPair2$ is that the relevant points in $Q$, namely $Q[\ell],\ldots,Q[r]$, are sorted WRT their $x$-co-ordinates.

The postcondition for $ClosestPair2$ is that $Q[\ell],\ldots,Q[r]$ are sorted WRT their $y$-co-ordinates.

This can be accomplished by merging two sublists $Q[\ell],\ldots,Q[m]$ and $Q[m+1],\ldots,Q[r]$ which are recursively sorted WRT their $y$-co-ordinates (this is identical to the merging step in $MergeSort$).
Closest Pair: Solution 2

Algorithm: \textit{ClosestPair2}(\ell, r)

\textbf{if } \ell = r \textbf{ then } \delta \leftarrow \infty

\textbf{else}

\begin{align*}
  & m \leftarrow \lfloor (\ell + r) / 2 \rfloor \\
  & X_{\text{mid}} \leftarrow Q[m].x \\
  & \delta_L \leftarrow \text{ClosestPair2}(\ell, m) \\
  & \text{comment: } Q[\ell], \ldots, Q[m] \text{ is sorted WRT } y\text{-coordinates} \\
  & \delta_R \leftarrow \text{ClosestPair2}(m + 1, r) \\
  & \text{comment: } Q[m + 1], \ldots, Q[r] \text{ is sorted WRT } y\text{-coordinates} \\
  & \delta \leftarrow \min\{\delta_L, \delta_R\} \\
  & \text{Merge}(\ell, m, r) \\
  & R \leftarrow \text{SelectCandidates}(\ell, r, \delta, X_{\text{mid}}) \\
  & \delta \leftarrow \text{CheckStrip}(R, \delta)
\end{align*}

\textbf{return } (\delta)
Multiprecision Multiplication

Problem 3.6

Multiprecision Multiplication

Instance: Two $k$-bit positive integers, $X$ and $Y$, having binary representations

\[ X = [X[k - 1], ..., X[0]] \]

and

\[ Y = [Y[k - 1], ..., Y[0]]. \]

Question: Compute the $2k$-bit positive integer $Z = XY$, where

\[ Z = (Z[2k - 1], ..., Z[0]). \]

Here, we are interested in the bit complexity of algorithms that solve Multiprecision Multiplication, which means that the complexity is expressed as a function of $k$ (the size of the problem instance is $2k$ bits).
A Divide-and-Conquer Approach

Assume \( k \) is even.

Let \( X_L \) be the integer formed by the \( k/2 \) high-order bits of \( X \) and let \( X_R \) be the integer formed by the \( k/2 \) low-order bits of \( X \).

Similarly for \( Y \).

Thus

\[
X = 2^{k/2}X_L + X_R \quad \text{and} \quad Y = 2^{k/2}Y_L + Y_R.
\]

Therefore, we have

\[
XY = 2^k X_L Y_L + 2^{k/2}(X_L Y_R + X_R Y_L) + X_R Y_R.
\]

Multiplication by a power of 2 is just a left shift.
Not-So-Fast D&C Multiprecision Multiplication

Algorithm: NotSoFastMultiply($X, Y, k$)

if $k = 1$
then $Z \leftarrow X[0] \times Y[0]$
    $Z_1 \leftarrow$ NotSoFastMultiply($X_L, Y_L, k/2$)
    $Z_2 \leftarrow$ NotSoFastMultiply($X_R, Y_R, k/2$)
else
    $Z_3 \leftarrow$ NotSoFastMultiply($X_L, Y_R, k/2$)
    $Z_4 \leftarrow$ NotSoFastMultiply($X_R, Y_L, k/2$)
    $Z \leftarrow$ LeftShift($Z_1, k$) + $Z_2$ + LeftShift($Z_3 + Z_4, k/2$)
return $Z$

What is the complexity of this algorithm?
An Improvement

Recall

\[ XY = 2^k X_L Y_L + 2^{k/2}(X_L Y_R + X_R Y_L) + X_R Y_R. \]

*Karatsuba’s algorithm* reduces the number of subproblems from 4 to 3.

Define

\[
\begin{align*}
Z_1 &= X_L Y_L \\
Z_2 &= X_R Y_R \\
Z_3 &= (X_L + X_R)(Y_L + Y_R).
\end{align*}
\]

Then

\[ X_L Y_R + X_R Y_L = Z_3 - Z_1 - Z_2. \]
Karatsuba Multiplication

Algorithm: \texttt{Karatsuba}(X, Y, k)

\begin{align*}
\text{if } k & = 1 \\
& \text{then } Z \leftarrow X[0] \times Y[0] \\
& \quad \begin{cases} 
X_T \leftarrow X_L + X_R \\
Y_T \leftarrow Y_L + Y_R 
\end{cases} \\
\text{else} & \\
& \quad \begin{cases} 
Z_1 \leftarrow \texttt{Karatsuba}(X_L, Y_L, k/2) \\
Z_2 \leftarrow \texttt{Karatsuba}(X_R, Y_R, k/2) \\
Z_3 \leftarrow \texttt{Karatsuba}(X_T, Y_T, k/2), \\
Z \leftarrow \texttt{Karatsuba}(Z_1, k) + Z_2 + \text{LeftShift}(Z_3 - Z_1 - Z_2, k/2)
\end{cases}
\end{align*}

\text{return } (Z)

What is the complexity of this algorithm?
Matrix Multiplication

Problem 3.7

Matrix Multiplication

Instance: Two $n$ by $n$ matrices, $A$ and $B$.
Question: Compute the $n$ by $n$ matrix product $C = AB$.

The naive algorithm for Matrix Multiplication has complexity $\Theta(n^3)$. 
D&C Matrix Multiplication: Problem Decomposition

Let

\[ A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}, \quad C = AB = \begin{pmatrix} r & s \\ t & u \end{pmatrix} \]

If \( A, B \) are \( n \) by \( n \) matrices, then \( a, b, ..., h, r, s, t, u \) are \( \frac{n}{2} \) by \( \frac{n}{2} \) matrices, where

\[ r = ae + bg \quad s = af + bh \\
\quad t = ce + dg \quad u = cf + dh \]

We require 8 multiplications of \( \frac{n}{2} \) by \( \frac{n}{2} \) matrices in order to compute \( C = AB \).

What is the complexity of the resulting divide-and-conquer algorithm?
Strassen Matrix Multiplication

Define

\[ P_1 = a(f - h) \]
\[ P_3 = (c + d)e \]
\[ P_5 = (a + d)(e + h) \]
\[ P_7 = (a - c)(e + f) \]
\[ P_2 = (a + b)h \]
\[ P_4 = d(g - e) \]
\[ P_6 = (b - d)(g + h) \]

Then, compute

\[ r = P_5 + P_4 - P_2 + P_6 \]
\[ s = P_1 + P_2 \]
\[ t = P_3 + P_4 \]
\[ u = P_5 + P_1 - P_3 - P_7 \]

We now require only 7 multiplications of \( \frac{n}{2} \) by \( \frac{n}{2} \) matrices in order to compute \( C = AB \).

What is the complexity of the resulting divide-and-conquer algorithm?
Selection

Problem 3.8

Selection

Instance: An array $A[1], \ldots, A[n]$ of distinct integer values, and an integer $k$, where $1 \leq k \leq n$.

Find: The $k$th smallest integer in the array $A$.

The problem Median is the special case of Selection where $k = \lceil \frac{n}{2} \rceil$. 
QuickSelect

Suppose we choose a pivot element $y$ in the array $A$, and we restructure $A$ so that all elements less than $y$ precede $y$ in $A$, and all elements greater than $y$ occur after $y$ in $A$. (This is exactly what is done in Quicksort, and it takes linear time.)


Then the $k$th smallest element of $A$ is

$$
\begin{align*}
& \begin{cases} 
y & \text{if } k = posn \\
& \text{the } k\text{th smallest element of } A_L & \text{if } k < posn \\
& \text{the } (k - posn)\text{th smallest element of } A_R & \text{if } k > posn.
\end{cases}
\end{align*}
$$

We make (at most) one recursive call at each level of the recursion.
Average-case Analysis of QuickSelect

We say that a pivot is **good** if \( posn \) is in the middle half of \( A \), i.e., \( n/4 \leq posn \leq 3n/4 \).

The probability that a pivot is good is \( 1/2 \).

On average, after **two iterations**, we will encounter a good pivot.

If a pivot is good, then \( |A_L| \leq 3n/4 \) and \( |A_R| \leq 3n/4 \).

With an **expected** linear amount of work, the size of the subproblem is reduced by at least 25%.

Let’s consider the average-case recurrence relation:
\[ T(n) = T(3n/4) + \Theta(n). \]

Apply the **Master Theorem** with \( a = 1, \ b = 4/3 \) and \( y = 1 \). Here \( x = \log_{4/3} 1 = 0 < 1 = y \) so we are in case 3.

This yields \( T(n) \in \Theta(n) \) on average.
Achieving $O(n)$ Worst-Case Complexity: A Strategy for Choosing the Pivot

We choose the pivot to be a certain median-of-medians:

**step 1** Given $n \geq 15$, write $n = 10r + 5 + \theta$, where $r \geq 1$ and $0 \leq \theta \leq 9$.

**step 2** Divide $A$ into $2r + 1$ disjoint subarrays of 5 elements. Denote these subarrays by $B_1, \ldots, B_{2r+1}$.

**step 3** For $1 \leq i \leq 2r + 1$, find the median of $B_i$ non-recursively, i.e., by brute force, and denote it by $m_i$.

**step 4** Define $M$ to be the array consisting of elements $m_1, \ldots, m_{2r+1}$.

**step 5** Find the median $y$ of the array $M$ recursively.

**step 6** Use the element $y$ as the pivot for $A$. 
Example

Suppose $|A| = 15$ and we divide $A$ into three groups of size 5:

\[
\begin{array}{cccccc}
1 & 10 & 5 & 8 & 21 & \rightarrow 8 \\
34 & 6 & 7 & 12 & 23 & \rightarrow 12 \\
2 & 4 & 30 & 11 & 25 & \rightarrow 11 \\
\end{array}
\]

The median of the three medians is 11. Then we have

\[
\begin{align*}
A_L &= 1, 10, 5, 8, 6, 7, 2, 4 \\
A_R &= 21, 34, 12, 23, 30, 25
\end{align*}
\]
Median-of-medians-QuickSelect

**Algorithm:** \( \text{MOM-QuickSelect}(k, n, A) \)

1. \textbf{if} \( n \leq 14 \) \textbf{then} sort \( A \) and \textbf{return} \( (A[k]) \)
2. write \( n = 10r + 5 + \theta \), where \( 0 \leq \theta \leq 9 \)
3. construct \( B_1, \ldots, B_{2r+1} \) (subarrays of \( A \), each of size 5)
4. find medians \( m_1, \ldots, m_{2r+1} \) \textbf{non-recursively}
5. \( M \leftarrow [m_1, \ldots, m_{2r+1}] \)
6. \( y \leftarrow \text{MOM-QuickSelect}(r + 1, 2r + 1, M) \)
7. \( (A_L, A_R, \text{posn}) \leftarrow \text{Restructure}(A, y) \)
8. \textbf{if} \( k = \text{posn} \) \textbf{then return} \( (y) \)
9. \textbf{else if} \( k < \text{posn} \) \textbf{then return} \( (\text{MOM-QuickSelect}(k, \text{posn} - 1, A_L)) \)
10. \textbf{else return} \( (\text{MOM-QuickSelect}(k - \text{posn}, n - \text{posn}, A_R)) \)
Recursive Calls in **Mom-QuickSelect**

We claim that the number of elements $> y$, or $< y$, is at most $7n/10$ (roughly).

Consider $n/5$ groups of five numbers, $B_1, \ldots, B_{n/5}$, and let $m_i$ be the median of $B_i$, for $1 \leq i \leq n/5$.

Let $y$ be the median of the $m_i$’s.

There are $n/10$ $j$’s such that $m_j < y$.

For each such $j$, there are three elements in $B_j$ that are less than $y$ (namely, $m_j$ and two other elements of $B_j$).

So there are at least $3n/10$ elements that are less than $y$ and hence there are at most $7n/10$ elements that are greater than $y$.

Similarly, there are at most $7n/10$ elements that are less than $y$. 
Worst-case Analysis of MOM-QuickSelect

Therefore, the recursive call is to a subarray of size at most $7n/10$ (roughly).

More precisely, the worst-case complexity $T(n)$ of this algorithm satisfies the following recurrence:

$$T(n) \leq \begin{cases} T\left(\left\lfloor \frac{n}{5} \right\rfloor \right) + T\left(\left\lfloor \frac{7n+12}{10} \right\rfloor \right) + \Theta(n) & \text{if } n \geq 15 \\ \Theta(1) & \text{if } n \leq 14. \end{cases}$$

How do we prove that $T(n)$ is $O(n)$?