Algorithms: Divide and Conquer

Amotz Bar-Noy

CUNY
A Dictionary Search Problem

**Input**
- A key $K$.

**Output**
- Does $K$ appear in $A$? **YES** or **NO**.
- If **YES**: The first index $i$ such that $A[i] = K$.
- If **NO**: The largest index $i$ such that $A[i] < K$ or $i = 0$ if $K < A[1]$.

**Method**
- **Comparisons** between $K$ and the keys in the array.

**Complexity**
- Number of **comparisons**.
A Search Game

Game
- **Player 1**: Selects an integer \( x \) in the range \([1..n]\).
- **Player 2**: Searches for \( x \) only with comparisons of the type \( x \leq i \) for some \( 1 \leq i \leq n \).

Players Goal
- **Player 1** tries to maximize the number of comparisons until finding \( x \).
- **Player 2** tries to minimize the number of comparisons until finding \( x \).

Complexity
- In the worst case or in the average case.
- As a function of \( n \).
The Two Models are “Equivalent”

**Equivalence**
- $x \leq i$ is “equivalent” to $K \leq A[i]$
- Algorithms can be “converted” from one model to another while preserving the complexity.

**Convenience**
- It is “easier” to design algorithms in the search game model.
- It is “easier” to prove bounds and limitations on algorithms in the search game model.
Sequential Search

Algorithm outline

- Assume a search for $x$ in the range $[1..n]$.
- Throughout the algorithm, maintain a lower bound $\ell$ on $x$ such that $\ell \leq x \leq n$.
- Initially, $\ell = 1$.
- In each round, compare $x$ with the lower bound $\ell$.
  - If $x > \ell$ then increment $\ell$ by 1.
  - If $x \leq \ell$ then return $\ell$. 
Sequential Search

Example

- **Input:** \( n = 10 \) and \( x = 7 \) \( \Rightarrow \) \( (* x \in [1..10] *) \)

- **Search procedure:**
  - Q1: \( x \leq 1 \) \( \Rightarrow \) A1: NO \( (* x \in [2..10] *) \)
  - Q2: \( x \leq 2 \) \( \Rightarrow \) A2: NO \( (* x \in [3..10] *) \)
  - Q3: \( x \leq 3 \) \( \Rightarrow \) A3: NO \( (* x \in [4..10] *) \)
  - Q4: \( x \leq 4 \) \( \Rightarrow \) A4: NO \( (* x \in [5..10] *) \)
  - Q5: \( x \leq 5 \) \( \Rightarrow \) A5: NO \( (* x \in [6..10] *) \)
  - Q6: \( x \leq 6 \) \( \Rightarrow \) A6: NO \( (* x \in [7..10] *) \)
  - Q7: \( x \leq 7 \) \( \Rightarrow \) A7: YES \( (* x \in [7..7] *) \)

- **Output:** \( x = 7 \)

- **Complexity:** 7 comparisons.
Sequential Search

Algorithm pseudocode I

Sequential-Search \((n, x)\)

\[ \ell = 1 \]

repeat

if \( x \leq \ell \) (* comparison *)

then return \( \ell \)

else \( \ell = \ell + 1 \)

Algorithm pseudocode II

Sequential-Search \((n, x)\)

\[ \ell = 1 \]

while \( x > \ell \) do (* comparison *)

\[ \ell = \ell + 1 \]

return \( \ell \)
Sequential Search

Correctness
- By induction, $\ell \leq x \leq n$ after $\ell - 1$ comparisons with a NO answer.

Termination
- If $x \leq \ell$ then necessarily $x = \ell$ because by the induction hypothesis $x \geq \ell$.
- Eventually $x \leq n$. 

Amotz Bar-Noy (CUNY)
Divide and Conquer
Sequential Search

Worst case complexity

- \textit{n comparisons} in the worst case when \( x = n \).
- In fact, only \( n - 1 \) \textit{comparisons} since there is no need for the last comparison when \( x = n \).

Best case complexity

- Only 1 \textit{comparison} when \( x = 1 \).

Average case complexity

- \((n + 1)/2 \) \textit{comparisons} on average for a random \( x \) selected with a uniform distribution from the range \([1..n]\):

\[
\frac{1}{n} \left(1 + 2 + \cdots + n\right) = \frac{1}{n} \cdot \frac{n(n + 1)}{2} = \frac{n + 1}{2}
\]
Sequential Search

Searching in an array pseudocode

- if \(K < A[1]\) then return \((K < A[1])\)  (* comparison *)
- if \(K > A[n]\) then return \((K > A[n])\)  (* comparison *)

\(\ell = 1\)

while \(K > A[\ell]\) do  (* comparison *)
- \(\ell = \ell + 1\)
- if \(K < A[\ell]\)  (* comparison *)
  - then return \((A[\ell - 1] < K < A[\ell])\)
  - else return \((K = A[\ell])\)

Worst case number of comparisons

- \(n + 3\) comparisons when \(K = A[n]\)
Binary Search

Algorithm outline

- Assume a search for $x$ in the range $[1..n]$.
- Throughout the algorithm, maintain a range $[\ell..u]$ such that $\ell \leq x \leq u$.
- Initially, $\ell = 1$ and $u = n$.
- In each round, compare $x$ with the middle of the range $m = \left\lfloor \frac{u+\ell}{2} \right\rfloor$.
- If $x \leq m$ then update $u = m$.
- If $x > m$ then update $\ell = m + 1$.
- Terminate when $\ell = u$.
- Return $x = \ell = u$. 
Binary Search – Example

- **Input:** \( n = 128 \) and \( x = 50 \) \( \Rightarrow \) (\( x \in [1..128] \)).

- **Search procedure:**
  - Q1: \( x \leq 64 \) \( \Rightarrow \) A1: YES (\( x \in [1..64] \))
  - Q2: \( x \leq 32 \) \( \Rightarrow \) A2: NO (\( x \in [33..64] \))
  - Q3: \( x \leq 48 \) \( \Rightarrow \) A3: NO (\( x \in [49..64] \))
  - Q4: \( x \leq 56 \) \( \Rightarrow \) A4: YES (\( x \in [49..56] \))
  - Q5: \( x \leq 52 \) \( \Rightarrow \) A5: YES (\( x \in [49..52] \))
  - Q6: \( x \leq 50 \) \( \Rightarrow \) A6: YES (\( x \in [49..50] \))
  - Q7: \( x \leq 49 \) \( \Rightarrow \) A7: NO (\( x \in [50..50] \))

- **Output:** \( x = 50 \)

- **Complexity:** \( 7 = \log_2(128) \) comparisons.
Algorithm Pseudocode

**Binary-Search** \((n, x)\)

\[
\begin{align*}
\ell &= 1 \\
u &= n \\
\text{while } \ell &< u \\
m &= \left\lfloor \frac{u + \ell}{2} \right\rfloor \\
\text{if } x &\leq m \quad (*) \text{ comparison } (*) \\
\text{then } u &= m \\
\text{else } \ell &= m + 1 \\
\text{return } \ell
\end{align*}
\]
Binary Search

Notations
- Let $u_j$ and $\ell_j$ be the values of $u$ and $\ell$ after iteration $j$ of the algorithm.
- Let $\Delta_j = u_j - \ell_j + 1$ be the size of the range $[\ell_j..u_j]$.
- Initially $\ell_0 = 1$, $u_0 = n$, and $\Delta_0 = n$.

Observation
- $\Delta_{j+1} \leq \left\lceil \frac{\Delta_j}{2} \right\rceil$ for $j \geq 0$.

Corollary
- $\Delta_k = 1$ for $k = \lceil \log_2 n \rceil$. 
Correctness

- By induction, always $\ell \leq x \leq u$.
- At the end, $\Delta = 1$ and therefore $\ell = u$ which implies that $x = \ell = u$.

Complexity

- There are at most $\lceil \log_2 n \rceil$ iterations and one comparison per iteration.
- Therefore, the worst-case complexity is $\lceil \log_2 n \rceil$.
- If $n$ is not a power of 2, then for some $x$ there are only $\lfloor \log_2 n \rfloor$ iterations.
- Therefore, the average-case complexity is approximately $\log_2 n$. 
Binary Search

Searching in an array pseudocode


if \(K < A[1]\) then return \((K < A[1])\) (* comparison *)

if \(K > A[n]\) then return \((K > A[n])\) (* comparison *)

\(\ell = 1\) and \(u = n\)

while \(\ell < u\)

\[m = \left\lfloor \frac{u + \ell}{2} \right\rfloor\]

if \(K \leq A[m]\) (* comparison *)

then \(u = m\)

else \(\ell = m + 1\)

if \(K < A[\ell]\) (* comparison *)

then return \((A[\ell - 1] < K < A[\ell])\)

else return \((K = A[\ell])\)

Number of comparisons

- \(\lceil \log_2 n \rceil + 3\) comparisons
## Binary-Search vs. Sequential-Search

<table>
<thead>
<tr>
<th></th>
<th>Binary-Search</th>
<th>Sequential-Search</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Best-Case</strong></td>
<td>$\lceil \log_2 n \rceil$</td>
<td>1</td>
</tr>
<tr>
<td><strong>Worst-Case</strong></td>
<td>$\lceil \log_2 n \rceil$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td><strong>Average-Case</strong></td>
<td>$\approx \log_2 n$</td>
<td>$\approx n/2$</td>
</tr>
</tbody>
</table>
Adversary Player I

Goal
- Maximize the number of comparisons until Player 2 finds $x$.

Strategy
- **Player 1 does not** select $x$ at the beginning of the game. Instead, it maintains a set $S$ of candidates for $x$.
- Given a search question:
  - $S(Y)$ – the set of candidates if the answer is **YES**.
  - $S(N)$ – the set of candidates if the answer is **NO**.
- The adversary answer rule:
  - **YES** if $|S(Y)| \geq |S(N)|$.
  - **NO** if $|S(Y)| < |S(N)|$.
Adversary Player I

Example: A possible algorithm

- **Input:** \( n = 34 \) (* \( x \in [1..34] \) *)
- **Search:**
  - Q1: \( x \leq 13 \Rightarrow A1: \text{NO} \) (* \( x \in [14..34] \) *)
  - Q2: \( x \leq 26 \Rightarrow A2: \text{YES} \) (* \( x \in [14..26] \) *)
  - Q3: \( x \leq 18 \Rightarrow A3: \text{NO} \) (* \( x \in [19..26] \) *)
  - Q4: \( x \leq 23 \Rightarrow A4: \text{YES} \) (* \( x \in [19..23] \) *)
  - Q5: \( x \leq 20 \Rightarrow A5: \text{NO} \) (* \( x \in [21..23] \) *)
  - Q6: \( x \leq 22 \Rightarrow A6: \text{YES} \) (* \( x \in [21..22] \) *)
  - Q7: \( x \leq 21 \Rightarrow A7: \text{YES} \) (* \( x \in [21..21] \) *)
- **Output:** \( x = 21 \).
Adversary Player I

Example: Binary-Search

- **Input:** \( n = 34 \)  \((* x \in [1..34] *)\)
- **Search:**
  - Q1: \( x \leq 17 \) \( \Rightarrow \) A1: YES  \((* x \in [1..17] *)\)
  - Q2: \( x \leq 9 \) \( \Rightarrow \) A2: YES  \((* x \in [1..9] *)\)
  - Q3: \( x \leq 5 \) \( \Rightarrow \) A3: YES  \((* x \in [1..5] *)\)
  - Q4: \( x \leq 3 \) \( \Rightarrow \) A4: YES  \((* x \in [1..3] *)\)
  - Q5: \( x \leq 2 \) \( \Rightarrow \) A5: YES  \((* x \in [1..2] *)\)
  - Q6: \( x \leq 1 \) \( \Rightarrow \) A6: YES  \((* x \in [1..1] *)\)
- **Output:** \( x = 1 \).

Observation

- With Binary-Search the search always ends up with \( x = 1 \).
Impossible to Search Faster than Binary Search

**Theorem**

There exists $1 \leq x \leq n$ for which the adversary forces the second player to ask at least $\lceil \log_2 n \rceil$ comparisons.

**Proof**

- Assume that **Player 2** asks $k$ comparisons to find $x$.
- Let $S_i$ be the set of candidates after $i$ comparisons.
- In particular, $|S_0| = n$ and $|S_k| = 1$.
- $S = S(Y) \cup S(N)$ implies that $|S_{i+1}| / |S_i| \geq (1/2)$ for $1 \leq i \leq k - 1$.
- $\lceil \log_2 n \rceil$ rounds are required to decrease $n$ to 1 by halving.
- Therefore, $k \geq \lceil \log_2 n \rceil$. 
Remarks

**Worst case**
- The $\lceil \log_2 n \rceil$ lower bound is a worst case bound.
- No algorithm can guarantee less comparisons for all values of $x$.

**Average case**
- It is possible to prove an $\Omega(\log n)$ average case lower bound.

**Other search models**
- The theorem holds for a “stronger” Player 2. One that may ask any YES/NO questions. For example,
  - Is $x$ even?
  - Is $x$ a prime number?
  - Does $x \in \{1, 2, 3, 5, 8, 13, 21, 34\}$?
Searching with “Clues”

Clue

- **Player 1** selects only even numbers 2, 4, 6, 8, \ldots between 1 and an even \( n \).

A modified Binary Search

- The search domain is 1, 2, \ldots, \( n/2 \).
- Instead of asking “if \( x \leq i \)”, **Player 2** asks “if \( x \leq 2i \)” and then considers the answer as if it was the answer to “if \( x \leq i \)”.
- When the search outputs \( x = i \) the modified search outputs \( 2i \).

Complexity

- \( \lceil \log_2(n/2) \rceil \approx \log_2(n/2) = \log_2(n) - 1 \) comparisons.
- The saving is only 1 comparison although the clue “eliminated” about half of the candidates!
**Searching with “Clues”**

**Clue**
- **Player 1** selects only even numbers 2, 4, 6, 8, ... between 1 and an even \( n \).

**Example**
- \( n = 32 = 2 \cdot 16 \) and \( x = 20 = 2 \cdot 10 \).
- The possible 16 values for \( x \) are 2, 4, 6, ... , 32 and the search domain is 1, 2, ... , 16.

**Running the algorithm**
- **Question 1:** \( x \leq (2 \cdot 8 = 16) \)? because \( 8 = \lfloor (1 + 16)/2 \rfloor \).
- **Question 2:** \( x \leq (2 \cdot 12 = 24) \)? because \( 12 = \lfloor (9 + 16)/2 \rfloor \).
- **Question 3:** \( x \leq (2 \cdot 10 = 20) \)? because \( 10 = \lfloor (9 + 12)/2 \rfloor \).
- **Question 4:** \( x \leq (2 \cdot 9 = 18) \)? because \( 9 = \lfloor (9 + 10)/2 \rfloor \).
- \( x = 20 \) found with \( 4 = \log_2 16 = \log_2 32 - 1 \) comparisons.
Searching with “Clues”

Clue

- **Player 1** selects only square numbers 1, 4, 9, 16, ... between 1 and a square number $n$.

A modified Binary Search

- The search domain is $1, 2, \ldots, \sqrt{n}$.
- Instead of asking “if $x \leq i$”, **Player 2** asks “if $x \leq i^2$” and then considers the answer as if it was the answer to “if $x \leq i$”.
- When the search outputs $x = i$ the modified search outputs $i^2$.

Complexity

- $\lceil \log_2(\sqrt{n}) \rceil \approx \log_2(\sqrt{n}) = \frac{1}{2} \log_2(n)$ comparisons.
- The saving is only half of the comparisons although the clue “eliminated” almost all the candidates!
Searching with “Clues”

Clue

- **Player 1** selects only square numbers 1, 4, 9, 16, \ldots \text{ between 1 and a square number } n.

Example

- \( n = 256 = 16^2 \) and \( x = 100 = 10^2 \).
- The possible 16 values for \( x \) are 1, 4, 9, \ldots, 256 and the search domain is 1, 2, \ldots, 16.

Running the algorithm

- **Question 1:** \( x \leq (8^2 = 64) \) because \( 8 = \lfloor (1 + 16)/2 \rfloor \).
- **Question 2:** \( x \leq (12^2 = 144) \) because \( 12 = \lfloor (9 + 16)/2 \rfloor \).
- **Question 3:** \( x \leq (10^2 = 100?) \) because \( 10 = \lfloor (9 + 12)/10 \rfloor \).
- **Question 4:** \( x \leq (9^2 = 81?) \) because \( 9 = \lfloor (9 + 10)/10 \rfloor \).
- \( x = 100 \) found with \( 4 = \log_2 16 = (1/2) \log_2 256 \) comparisons.
Searching with “Clues”

Clue
- **Player 1** selects only powers of 2 numbers 2, 4, 8, 16, ... between 2 and a power of 2 number $n$.

A modified Binary Search
- The search domain is $1, 2, \ldots, \log_2 n$.
- Instead of asking “if $x \leq i$”, **Player 2** asks “if $x \leq 2^i$” and then considers the answer as if it was the answer to “if $x \leq i$”.
- When the search outputs $x = i$ the modified search outputs $2^i$.

Complexity
- $\lceil \log_2 (\log_2 (n)) \rceil \approx \log_2 (\log_2 (n))$ comparisons.
- For $n = 2^{32} = 4294967296$ the saving is from 32 to 5 comparisons although there are only 32 candidates!
Searching with “Clues”

**Clue**
- **Player 1** selects only powers of 2 numbers 2, 4, 8, 16, ... between 2 and a power of 2 number \( n \).

**Example**
- \( n = 65536 = 2^{16} \) and \( x = 1024 = 2^{10} \).
- The possible 16 values for \( x \) are 2, 4, 8, ..., 65536 and the search domain is 1, 2, ..., 16.

**Running the algorithm**
- **Question 1:** \( x \leq 2^8 = 256 \)? \( 8 = \lfloor (1 + 16)/2 \rfloor \).
- **Question 2:** \( x \leq 2^{12} = 4096 \)? \( 12 = \lfloor (9 + 16)/2 \rfloor \).
- **Question 3:** \( x \leq 2^{10} = 1024 \)? \( 10 = \lfloor (9 + 12)/10 \rfloor \).
- **Question 4:** \( x \leq 2^9 = 512 \)? \( 9 = \lfloor (9 + 10)/10 \rfloor \).
- \( x = 1024 \) found with \( 4 = \log_2 16 = \log_2 \log_2 65536 \) comparisons.
Searching with “Clues”

Clue
- **Player 1** selects only primes 2, 3, 5, 7, … not larger than \( n \).

A modified Binary Search
- The search domain is 1, 2, … , \( \pi(n) \) where \( \pi(n) \) is the number of primes between 2 and \( n \).
- Instead of asking “if \( x \leq i \)”, **Player 2** asks “if \( x \leq p_i \)” where \( p_i \) is the \( i \)th prime and then considers the answer as if it was the answer to “if \( x \leq i \)”.  
- When the search outputs \( x = i \) the modified search outputs \( p_i \).

Complexity
- \( \log_2(n/\ln(n)) \approx \log_2(n) - \log_2 \log_2(n) \) comparisons because there are approximately \( n/\ln(n) \) primes between 2 and \( n \).
- There are 78498 primes between 2 and 1000000. The clue saves only 3 comparisons, because \( \lceil \log_2(1000000) \rceil = 20 \) and \( \lceil \log_2(78498) \rceil = 17 \).
Searching with “Clues”

Clue
- Player 1 selects only primes 2, 3, 5, 7, … not larger than \( n \).

Example
- \( n = 53 \) and \( x = 29 \).
- The possible 16 values for \( x \) are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53
- and the search domain is 1, 2, \ldots, 16.

Running the algorithm
- Question 1: \( x \leq (p_8 = 19) \) ? \( 8 = \lfloor (1 + 16)/2 \rfloor \).
- Question 2: \( x \leq (p_{12} = 37) \) ? \( 12 = \lfloor (9 + 16)/2 \rfloor \).
- Question 3: \( x \leq (p_{10} = 29) \) ? \( 10 = \lfloor (9 + 12)/10 \rfloor \).
- Question 4: \( x \leq (p_9 = 23) \) ? \( 9 = \lfloor (9 + 10)/10 \rfloor \).
- \( x = 29 \) found with \( 4 \approx \log_2(53) - \log_2 \log_2(53) \) comparisons.
Searching an Unbounded Domain

**Game**
- **Player 1:** Selects any positive integer \( x \).
- **Player 2:** Searches for \( x \) with comparisons \( x \leq i \) for some integer \( i \).

**Adversary Player 1**
- Always answers **NO**.
- **Player 2** will never find \( x \)!

**Player 2 Goal**
- Find \( x \) with as minimum possible comparisons as a function of \( x \).
- Ask “**less**” comparisons when \( x \) is small and ask “**more**” comparisons when \( x \) is large.
Searching an Unbounded Domain

Sequential search
- Sequential search finds $x$ with exactly $x$ comparisons.

The doubling technique
- A strategy that finds $x$ with approximately $2 \log_2(x)$ comparisons.

A more sophisticated doubling technique
- A strategy that finds $x$ with approximately $\log_2(x) + 2 \log_2 \log_2(x)$ comparisons.

Optimal solution
- A strategy that finds $x$ with approximately $\log_2(x) + \log_2 \log_2(x) + \log_2 \log_2 \log_2(x) + \cdots$ comparisons.
The Doubling Technique

Strategy

- **Phase 1:** Ask the following comparisons until the answer is YES:
  \[ x \leq 1? \quad x \leq 2? \quad x \leq 4? \quad x \leq 8? \quad \cdots \quad x \leq 2^j? \quad \cdots \]
- Assume \( 2^{k-1} < x \leq 2^k \)
- **Phase 2:** Apply binary search on the domain \([2^{k-1} + 1..2^k]\)

Complexity

- \( k + 1 \) comparisons are asked in **Phase 1**.
- The number of comparisons asked in **Phase 2** is
  \[ \lceil \log_2(2^k - (2^{k-1} + 1) + 1) \rceil = \lceil \log_2(2^{k-1}) \rceil = k - 1 \]

Total number of comparisons:
  \[ (k + 1) + (k - 1) = 2k = 2 \lceil \log_2(x) \rceil \]
The Doubling Technique – Example

**Input:** \( x = 50 \)

**Search procedure:**

- Q1: \( x \leq 1 \) ⇒ A1: NO (* \( x \in [2..\infty] \) *).
- Q2: \( x \leq 2 \) ⇒ A2: NO (* \( x \in [3..\infty] \) *).
- Q3: \( x \leq 4 \) ⇒ A3: NO (* \( x \in [5..\infty] \) *).
- Q4: \( x \leq 8 \) ⇒ A4: NO (* \( x \in [9..\infty] \) *).
- Q5: \( x \leq 16 \) ⇒ A5: NO (* \( x \in [17..\infty] \) *).
- Q6: \( x \leq 32 \) ⇒ A6: NO (* \( x \in [33..\infty] \) *).
- Q7: \( x \leq 64 \) ⇒ A7: YES (* \( x \in [33..64] \) *).
- Q8: \( x \leq 48 \) ⇒ A8: NO (* \( x \in [49..64] \) *).
- Q9: \( x \leq 56 \) ⇒ A9: YES (* \( x \in [49..56] \) *).
- Q10: \( x \leq 52 \) ⇒ A10: YES (* \( x \in [49..52] \) *).
- Q11: \( x \leq 50 \) ⇒ A11: YES (* \( x \in [49..50] \) *).
- Q12: \( x \leq 49 \) ⇒ A12: NO (* \( x \in [50..50] \) *).

**Output:** \( x = 50 \)

**Complexity:** \( 12 = \lceil 2 \log_2 50 \rceil \) comparisons.
**Input**

- Two integers \( I \) and \( J \) each represented by \( n \geq 1 \) (binary) bits.

**Output**

- The product \( I \times J \).

**Observation**

- The product has at most \( 2n \) bits in its binary representation.

**Complexity Objective**

- Minimize the number of bit operations.
  - Multiplications.
  - Additions and Subtractions.
  - Shifts.
Multiplying Two Base 10 Integers

\[
\begin{array}{cccccc}
5 & 3 & 6 & 8 \\
2 & 9 & 1 & 7 \\
\hline
3 & 7 & 5 & 7 & 6 \\
5 & 3 & 6 & 8 \\
4 & 8 & 3 & 1 & 2 \\
1 & 0 & 7 & 3 & 6 \\
\hline
1 & 5 & 6 & 5 & 8 & 4 & 5 & 6
\end{array}
\]
Multiply two base 2 (binary) integers:

Example

- \( I = 1001 \) in base 2 which is 9 in base 10.
- \( J = 1101 \) in base 2 which is 13 in base 10.
- \( I \times J = 9 \times 13 = 117 \) in base 10.
- \( 117 = 64 + 32 + 16 + 4 + 1 \) is 1110101 in base 2

```
  1 0 0 1
+ 1 1 0 1
  --------
  1 0 0 1
  0 0 0 0
+ 1 0 0 1
  1 0 0 1
  1 1 1 0 1
```
Algorithms

**Direct**
- \( I \times J \) can be computed with \( \Theta(n^2) \) bit operations using the traditional algorithm.

**Divide and Conquer**
- \( I \times J \) can be computed with \( \Theta(n^{\log_2 3}) \approx \Theta(n^{1.585}) \) bit operations.

**Fast-Transform-Fourier**
- Using the Fast-Transform-Fourier, \( I \times J \) can be computed with \( \Theta(n \log n) \) bit operations.

**Lower bound**
- \( \Omega(n) \) bit operations.
Other Operations

Additions and Subtractions
- $I + J$ and $I - J$ can be computed with $\Theta(n)$ bit operations.
- $I + J$ and $I - J$ need at least $\Omega(n)$ bit operations.

Multiplying by a power of 2
- Multiplying an $m$-bit integer by $2^k$ can be done with $\Theta(k + m)$ bit operations (shifting).
The Divide and Conquer Setting

Assumption
- The length $n$ of $I$ and $J$ is a power of 2.

Input representation
\[
I = I_h \cdot 2^{n/2} + I_l \\
J = J_h \cdot 2^{n/2} + J_l
\]

Example
- $1973 = 19 \cdot 10^2 + 73$: nineteen hundreds seventy three.

Objective
- Compute the product $I \times J$ of two length-$n$ integers recursively using multiplications among the four length-$n/2$ integers:
  \[
  I_h \quad I_l \quad J_h \quad J_l
  \]
Integer Multiplication

Divide and Conquer I

Computation

\[ I \times J = (I_h 2^{n/2} + I_\ell) \times (J_h 2^{n/2} + J_\ell) \]
\[ = (I_h \times J_h) 2^n + (I_h \times J_\ell + I_\ell \times J_h) 2^{n/2} + I_\ell \times J_\ell \]

Example

- \( I = 5368, J = 2917 \implies I \times J = 15658456 \)
- \( I_h = 53, I_\ell = 68, J_h = 29, J_\ell = 17 \)
- \( I_h \times J_h = 53 \times 29 = 1537 \)
- \( I_h \times J_\ell + I_\ell \times J_h = 53 \times 17 + 68 \times 29 = 901 + 1972 = 2873 \)
- \( I_\ell \times J_\ell = 68 \times 17 = 1156 \)

\[ I \times J = (1537)10^4 + (2873)10^2 + 1156 \]
\[ = 15370000 + 287300 + 1156 = 15658456 \]
Integer Multiplication

Divide and Conquer I

Computation

\[ I \times J = \left(I_h 2^{n/2} + I_\ell\right) \times \left(J_h 2^{n/2} + J_\ell\right) \]
\[ = \left(I_h \times J_h\right)2^n + \left(I_h \times J_\ell + I_\ell \times J_h\right)2^{n/2} + I_\ell \times J_\ell \]

Number of bit operations

\[ T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n) \]
\[ = \Theta(n^2) \]

Result

- Not an improvement!
Solving the Recursion

\[
T(1) = 1 \\
T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n)
\]

- \(a = 4\).
- \(b = 2\).
- \(\log_b(a) = 2\).
- \(d = 1\).
- \(d < \log_b(a)\).

**Master Theorem Case 1:** \(T(n) = \Theta(n^2)\).
Key idea

- Compute the product $I \times J$ using **only three** multiplications among $I_h, I_\ell, J_h, J_\ell$ with $\Theta(n)$ extra work.

Computation

\[
(I_h - I_\ell)(J_\ell - J_h) = I_h J_\ell - I_h J_h - I_\ell J_\ell + I_\ell J_h
\]

\[
I_h J_\ell + I_\ell J_h = (I_h - I_\ell)(J_\ell - J_h) + I_h J_h + I_\ell J_\ell
\]

$A = I_h \times J_h$

$B = I_\ell \times J_\ell$

$C = (I_h - I_\ell) \times (J_\ell - J_h)$

$I \times J = I_h J_h 2^n + (I_h J_\ell + I_\ell J_h)2^{n/2} + I_\ell J_\ell$

$I \times J = A2^n + (C + A + B)2^{n/2} + B$
Example

- \( I = 5368, J = 2917 \implies I \times J = 15658456. \)
- \( I_h = 53, I_\ell = 68, J_h = 29, J_\ell = 17. \)
- \( A = I_h \times J_h = 53 \times 29 = 1537. \)
- \( B = I_\ell \times J_\ell = 68 \times 17 = 1156. \)
- \( C = (I_h - I_\ell) \times (J_\ell - J_h) = (53 - 68)(17 - 29) = (-15)(-12) = 180. \)

\[
I \times J \quad = \quad (1537)10^4 + (180 + 1537 + 1156)10^2 + 1156 \\
\quad = \quad (1537)10^4 + (2873)10^2 + 1156 \\
\quad = \quad 15370000 + 287300 + 1156 = 15658456
\]
**Divide and Conquer II**

**Computation**

\[
A = I_h \times J_h \\
B = I_\ell \times J_\ell \\
C = (I_h - I_\ell) \times (J_\ell - J_h) \\
I \times J = A2^n + (C + A + B)2^{n/2} + B
\]

**Number of bit operations**

\[
T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) \\
= \Theta(n^{\log_2 3}) \approx \Theta(n^{1.585})
\]

**Result**

- Better than \(\Theta(n^2)\)!
Solving the Recursion

\[
T(1) = 1 \\
T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)
\]

- \(a = 3\).
- \(b = 2\).
- \(\log_b(a) \approx 1.585\).
- \(d = 1\).
- \(d < \log_b(a)\).

**Master Theorem Case 1:** \(T(n) = \Theta(n^{\log_2 3})\).
Arbitrary $n \geq 1$

Input

- $2^{k-1} < n \leq 2^k \implies 2^k < 2n$.

Algorithm

- **Add** $2^k - n$ zeros in front of both integers.
- **Run** the computation with the new integers of length $2^k$.
- **Omit** zeros from the beginning of the product.

Observation

- $n \log_2 3 \leq (2^k) \log_2 3 < (2n) \log_2 3 = 2 \log_2 3 n \log_2 3 = 3n \log_2 3$

Complexity

- The algorithm complexity $\Theta((2^k) \log_2 3)$ is $\Theta(n \log_2 3)$. 
Online Resources

13-minute Video lecture: Algorithm without analysis justification
- https://www.youtube.com/watch?v=JCbZayFr9RE

10 slides: Summary with a detailed example

Article: Motivation, details, but no analysis
Matrix Multiplication

Input
- For $n \geq 1$, two $n \times n$ matrices $A$ and $B$ of scalars (usually numbers).

Output
- An $n \times n$ matrix $C = A \times B$.

Definition
- For all $1 \leq i, j \leq n$:
  \[ c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \]

Complexity objective
- Minimize the number of additions, subtractions, and multiplications between two scalars.
Matrix Multiplication

Multiplying two $2 \times 2$ Matrices

$$C = A \times B$$

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

The 4 scalars in $C$

- $r = ae + bf$
- $s = ag + bh$
- $t = ce + df$
- $u = cg + dh$

Complexity

- Total of 8 multiplications and 4 additions.
# Matrix Multiplication

## Multiplying Two $3 \times 3$ Matrices

The equation for multiplying two $3 \times 3$ matrices is given by

$$\mathbf{C} = \mathbf{A} \times \mathbf{B}$$

where

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

The 9 scalars in $\mathbf{C}$ are:

- $c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$
- $c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32}$
- $c_{13} = a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33}$
- $c_{21} = a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31}$
- $c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32}$
- $c_{23} = a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33}$
- $c_{31} = a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31}$
- $c_{32} = a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32}$
- $c_{33} = a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33}$

## Complexity

- Total of 27 multiplications and 18 additions.
Matrix Multiplication

The Direct Algorithm

Pseudocode

\[ C = A \times B \quad (\text{* A, B, C are } n \times n \text{ matrices of numbers *}) \]

for \( i = 1 \) to \( n \) do
  for \( j = 1 \) to \( n \) do
    \[ c_{ij} = a_{i1} b_{1j} \]
    for \( k = 2 \) to \( n \) do
      \[ c_{ij} = c_{ij} + a_{ik} b_{kj} \]

Complexity

- Total of \( n^3 \) multiplications and \((n - 1)n^2\) additions.
Matrix Multiplication – Algorithms

Direct algorithm
- $\Theta(n^3)$ operations: $n^3$ multiplications and $n^2(n - 1)$ additions.

Strassen algorithm
- $\Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$ operations.

Best known algorithm (2022)
- $O(n^{2.37188})$ operations.

Lower bound
- $\Omega(n^2)$ operations.
**Matrix Multiplication**

**Input**
- For $n \geq 1$, two $n \times n$ matrices $A$ and $B$ of scalars.

**Output**
- An $n \times n$ matrix $C = A + B$.

**Definition**
- For all $1 \leq i, j \leq n$:
  \[ c_{ij} = a_{ij} + b_{ij} \]

**Complexity**
- Exactly $n^2$ additions.
- **Optimal** since any computation must be based on all the $2n^2$ scalars from $A$ and $B$. 
Adding $2 \times 2$ Matrices

\[ \mathbf{C} = \mathbf{A} + \mathbf{B} \]

\[
\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} + \begin{pmatrix} e & g \\ f & h \end{pmatrix}
\]

The 4 scalars in \( \mathbf{C} \)
- \( r = a + e \)
- \( s = b + g \)
- \( t = c + f \)
- \( u = d + h \)

Complexity
- Total of \( 4 = 2^2 \) additions.
Adding $3 \times 3$ Matrices

\[ \textbf{C} = \textbf{A} \times \textbf{B} \]

\[
\begin{pmatrix}
  c_{11} & c_{12} & c_{13} \\
  c_{21} & c_{22} & c_{23} \\
  c_{31} & c_{32} & c_{33}
\end{pmatrix}
= 
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{pmatrix}
+ 
\begin{pmatrix}
  b_{11} & b_{12} & b_{13} \\
  b_{21} & b_{22} & b_{23} \\
  b_{31} & b_{32} & b_{33}
\end{pmatrix}
\]

The 9 scalars in $\textbf{C}$

- $c_{11} = a_{11} + b_{11}$
- $c_{12} = a_{12} + b_{12}$
- $c_{13} = a_{13} + b_{13}$
- $c_{21} = a_{21} + b_{21}$
- $c_{22} = a_{22} + b_{22}$
- $c_{23} = a_{23} + b_{23}$
- $c_{31} = a_{31} + b_{31}$
- $c_{32} = a_{32} + b_{32}$
- $c_{33} = a_{33} + b_{33}$

Complexity

- Total of $9 = 3^2$ additions.
The Divide and Conquer Setting

Assumption
- The size $n$ of both $A$ and $B$ is a power of 2.

Input representation

$$\begin{pmatrix} (C_{11}) & (C_{12}) \\ (C_{21}) & (C_{22}) \end{pmatrix} = \begin{pmatrix} (A_{11}) & (A_{12}) \\ (A_{21}) & (A_{22}) \end{pmatrix} \times \begin{pmatrix} (B_{11}) & (B_{12}) \\ (B_{21}) & (B_{22}) \end{pmatrix}$$

Objective
- Compute the product $A \times B$ of two $n \times n$ matrices using multiplications among the eight $(n/2) \times (n/2)$ matrices

$$A_{11}, A_{12}, A_{21}, A_{22}, B_{11}, B_{12}, B_{21}, B_{22}$$

- In other words, show how to represent each one of the four $(n/2) \times (n/2)$ matrices $C_{11}, C_{12}, C_{21}, C_{22}$ as a function of the above eight $(n/2) \times (n/2)$ matrices.
Matrix Multiplication

Divide-and-Conquer I

\[ C = A \times B \]

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

Lemma

- The multiplication procedure works for sub-matrices as well.

The 4 submatrices in \( C \)

- \((C_{11}) = (A_{11}) \times (B_{11}) + (A_{12}) \times (B_{21})\)
- \((C_{12}) = (A_{11}) \times (B_{12}) + (A_{12}) \times (B_{22})\)
- \((C_{21}) = (A_{21}) \times (B_{11}) + (A_{22}) \times (B_{21})\)
- \((C_{22}) = (A_{21}) \times (B_{12}) + (A_{22}) \times (B_{22})\)
Matrix Multiplication

Divide-and-Conquer I

**Algorithm**
- **Partition** $A$ and $B$ into 4 sub-matrices each of size $\frac{n}{2} \times \frac{n}{2}$.
- **Recursively compute** the 8 sub-matrices multiplications.
- **Do** the 4 matrices additions each with $(n/2)^2$ addition operations.

**Number of scalar operations**

\[
T(n) = 8T\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2
\]

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

\[
= \Theta(n^3)
\]

**Result**
- Not an improvement!
Solving the Recursion

\[ T(1) = 1 \]
\[ T(n) = 8T \left( \frac{n}{2} \right) + \Theta(n^2) \]

- \( a = 8 \).
- \( b = 2 \).
- \( \log_b(a) = 3 \).
- \( d = 2 \).
- \( d < \log_b(a) \).

**Master Theorem Case 1:** \( T(n) = \Theta(n^3) \).
The "out of the blue" computation

With 7 multiplications and 10 additions, compute 7 help variables:

\[ p_1 = a(g - h) \]
\[ p_2 = (a + b)h \]
\[ p_3 = (c + d)e \]
\[ p_4 = d(f - e) \]
\[ p_5 = (a + d)(e + h) \]
\[ p_6 = (b - d)(f + h) \]
\[ p_7 = (a - c)(e + g) \]

With 8 more additions, compute \( r, s, t, u \):

\[ r = p_5 + p_4 - p_2 + p_6 \]
\[ s = p_1 + p_2 \]
\[ t = p_3 + p_4 \]
\[ u = p_5 + p_1 - p_3 - p_7 \]

Complexity

Total of 7 multiplications and 18 additions.
Matrix Multiplication

Verifying the Computation

\[
\begin{pmatrix}
  r & s \\
  t & u
\end{pmatrix} = \begin{pmatrix}
  a & b \\
  c & d
\end{pmatrix} \times \begin{pmatrix}
  e & g \\
  f & h
\end{pmatrix}
\]

Verifying the value of \( r \)

\[
r = p_5 + p_4 - p_2 + p_6 \\
= (a + d)(e + h) + d(f - e) - (a + b)h + (b - d)(f + h) \\
= ae + ah + de + dh + df - de - ah - bh + bf + bh - df - dh \\
= ae + ah + de + dh + df - de - ah - bh + bf + bh - df - dh \\
= ae + bf
\]
Matrix Multiplication

Verifying the Computation

\[
\begin{pmatrix}
  r & s \\
  t & u
\end{pmatrix} = \begin{pmatrix}
  a & b \\
  c & d
\end{pmatrix} \times \begin{pmatrix}
  e & g \\
  f & h
\end{pmatrix}
\]

Verifying the value of \( s \)

\[
s = p_1 + p_2 \\
= a(g - h) + (a + b)h \\
= ag - ah + ah + bh \\
= ag - ah + ah + bh \\
= ag + bh
\]
Matrix Multiplication

Verifying the Computation

\[
\begin{pmatrix}
  r & s \\
t & u
\end{pmatrix} =
\begin{pmatrix}
  a & b \\
c & d
\end{pmatrix} \times
\begin{pmatrix}
  e & g \\
f & h
\end{pmatrix}
\]

Verifying the value of \( t \)

\[
\begin{aligned}
t &= p_3 + p_4 \\
  &= (c + d)e + d(f - e) \\
  &= ce + de + df - de \\
  &= ce + df
\end{aligned}
\]
Verifying the Computation

\[
\begin{pmatrix}
  r & s \\
  t & u
\end{pmatrix}
= \begin{pmatrix}
  a & b \\
  c & d
\end{pmatrix}
\times \begin{pmatrix}
  e & g \\
  f & h
\end{pmatrix}
\]

Verifying the value of \( u \)

\[
u = p_5 + p_1 - p_3 - p_7
\]
\[
= (a + d)(e + h) + a(g - h) - (c + d)e - (a - c)(e + g)
\]
\[
= ae + ah + de + dh + ag - ah - ce - de - ae - ag + ce + cg
\]
\[
= ae + ah + de + dh + ag - ah - ce - de - ae - ag + cg + dh
\]
\[
= cg + dh
\]
Matrix Multiplication

Divide-and-Conquer II

Algorithm

- **Partition** $A$ and $B$ into 4 sub-matrices of size $\frac{n}{2} \times \frac{n}{2}$.
- **Recursively compute** the 7 sub-matrices multiplications.
- **Do** the 18 matrices additions each with $(n/2)^2$ addition operations.

Number of scalar operations

$$T(n) = 7T\left(\frac{n}{2}\right) + 18 \left(\frac{n}{2}\right)^2$$

$$= 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$= \Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$$

Result

- Better than $\Theta(n^3)$!
Solving the Recursion

\[ T(1) = 1 \]
\[ T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2) \]

- \( a = 7. \)
- \( b = 2. \)
- \( \log_b(a) \approx 2.81. \)
- \( d = 2. \)
- \( d < \log_b(a). \)

\textbf{Master Theorem Case 1: } \( T(n) = \Theta\left(n^{\log_2 7}\right). \)
**Arbitrary \( n \geq 1 \)**

**Input**
- \( 2^{k-1} < n \leq 2^k \implies 2^k < 2n \).

**Algorithm**
- **Add** \((2^k - n)\) zero-columns and rows to both \(A\) and \(B\).
- **Run** the algorithm for the new matrices of size \(2^k \times 2^k\).
- **Omit** the zero columns and rows from \(C\).

**Observation**
- \( n^{\log_2 7} \leq (2^k)^{\log_2 7} < (2n)^{\log_2 7} = 2^{\log_2 7}n^{\log_2 7} = 7n^{\log_2 7} \)

**Complexity**
- The algorithm complexity \( \Theta((2^k)^{\log_2 7}) \) is \( \Theta(n^{\log_2 7}) \).
Online Resources

23-minute video lecture: Algorithm without analysis justification
https://www.youtube.com/watch?v=ORrM-aSNZUs

Text lecture: Includes most of the details with implementations
https://www.geeksforgeeks.org/strassens-matrix-multiplication/