Algorithm Efficiency deals with:

- Methods of solution rather than programs
- Efficient use of time and memory
- Mathematical methods independent of coding, computers or data
Don't be misled by

• How are the algorithms coded?
• What computer should you use?
• What data should programs use?
Execution Time

• Depends on number of operations

• Consider two List implementations:
  • Array-based: Each item requires one step to access
  • Pointer-based: Last item requires $n$ steps to access

• Conclusion:
  • Array-based: Size of array does not affect time
  • Pointer-based: The larger the list the more time to access (time is proportional to $n$)
Traversing a linked list
// List consists of N nodes
Node *cur = head;

while (cur != NULL) {
    cout << cur->item << endl;
    cur = cur->next;
}

1 assignment
n+1 comparison
n writes
n assignments

total: \((n+1)(a + c) + n \times (w) \propto N\)
Towers of Hanoi

- N disks require $2^n - 1$ moves
- Each move requires $m$ time units
- $2^n - 1$ moves require $(2^n - 1) \times m$ time units
- Exponential times grow very quickly. 😞
nesting is the bottleneck
for (i = 1 through n) // n-times
  for (j = 1 through i) // i-times
    for (k = 1 through 5) // 5-times
      task T

T: (5) + (5 + 5) + (5 + 5 + 5) + ... + 1 + 2 + 3 + ... + N
: N * (N + 1) / 2 ∝ N^2
BigO formalities
Time v N

- Establish a relationship between time and n
- Time proportional to some function of n, f(n)
- This function is the growth rate, indicated as $O(f(n))$
From this point on, the number of operations is less than $k \times f(n)$.
Beyond this point, the operations for the algorithm are bounded above by $n^2$ and by $n^3$. The actual number of operations for each $n \propto O(n^3) \propto O(n^2)$.
<table>
<thead>
<tr>
<th>Function</th>
<th>10</th>
<th>100</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>log&lt;sub&gt;2&lt;/sub&gt;n</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>16</td>
<td>19</td>
</tr>
<tr>
<td>n</td>
<td>10</td>
<td>10&lt;sup&gt;2&lt;/sup&gt;</td>
<td>10&lt;sup&gt;3&lt;/sup&gt;</td>
<td>10&lt;sup&gt;4&lt;/sup&gt;</td>
<td>10&lt;sup&gt;5&lt;/sup&gt;</td>
<td>10&lt;sup&gt;6&lt;/sup&gt;</td>
</tr>
<tr>
<td>n * log&lt;sub&gt;2&lt;/sub&gt;n</td>
<td>30</td>
<td>664</td>
<td>9,965</td>
<td>10&lt;sup&gt;5&lt;/sup&gt;</td>
<td>10&lt;sup&gt;6&lt;/sup&gt;</td>
<td>10&lt;sup&gt;7&lt;/sup&gt;</td>
</tr>
<tr>
<td>n&lt;sup&gt;2&lt;/sup&gt;</td>
<td>10&lt;sup&gt;2&lt;/sup&gt;</td>
<td>10&lt;sup&gt;4&lt;/sup&gt;</td>
<td>10&lt;sup&gt;6&lt;/sup&gt;</td>
<td>10&lt;sup&gt;8&lt;/sup&gt;</td>
<td>10&lt;sup&gt;10&lt;/sup&gt;</td>
<td>10&lt;sup&gt;12&lt;/sup&gt;</td>
</tr>
<tr>
<td>n&lt;sup&gt;3&lt;/sup&gt;</td>
<td>10&lt;sup&gt;3&lt;/sup&gt;</td>
<td>10&lt;sup&gt;6&lt;/sup&gt;</td>
<td>10&lt;sup&gt;9&lt;/sup&gt;</td>
<td>10&lt;sup&gt;12&lt;/sup&gt;</td>
<td>10&lt;sup&gt;15&lt;/sup&gt;</td>
<td>10&lt;sup&gt;18&lt;/sup&gt;</td>
</tr>
<tr>
<td>2&lt;sup&gt;n&lt;/sup&gt;</td>
<td>10&lt;sup&gt;3&lt;/sup&gt;</td>
<td>10&lt;sup&gt;30&lt;/sup&gt;</td>
<td>10&lt;sup&gt;301&lt;/sup&gt;</td>
<td>10&lt;sup&gt;3,010&lt;/sup&gt;</td>
<td>10&lt;sup&gt;30,103&lt;/sup&gt;</td>
<td>10&lt;sup&gt;301,030&lt;/sup&gt;</td>
</tr>
</tbody>
</table>
don't be quick to judge
Algorithm A requires $n^2/5$ seconds

Algorithm B requires $5\times n$ seconds
$1 < \log n < n < n \log n < n^2 < n^3 < 2^n$
What you are going for

• Largest power of $n$ in polynomial
• Ignore constants
• Worst order wins (eg $n$ over $\log n$)
What is measured
Worst Case v Avg Case
total = 0;

for (int row = 0; row < n; row++) {
    RowTotal[row] = 0;
    for (int col = 0; col < n; col++) {
        RowTotal[row] += mat[row][col];
        total += mat[row][col];
    }
}

n * n = O(n^2)

Nesting is usually the issue
for (k = 1; k <= n / 2; k++) {
  ...
}

for (j = 1; j <= n * n; j++) {
  ...
}

\[ n \times n^2 = O(n^3) \]

Nesting is usually the issue
for (k = 1; k <= n / 2; k++) {
    ...
}
for (j = 1; j <= n * n; j++) {
    ...
}

\[ n + n^2 = O(n^2) \]

Sequential is usually better
```plaintext
k = n;
while (k >= 1) {
    ...
    k = k / 2;
}
\[ \log N \]
\[ O(\log N) \]
\[ 0(\log N) \]
```
for (k = 1; k <= n; k++) {
    j = n;
    while (j > 0) {
        ...
        j = j / 2;
    }
}

\[ n \times \log n = O(n \log \log N) \]
\[ k = 1; \]
\[ \text{do } \{ \]
\[ \hspace{0.5em} j = 1; \]
\[ \hspace{1.5em} \text{do } \{ \]
\[ \hspace{2.5em} j = 2 \ast j; \]
\[ \} \text{ while (} j \leq n); \]
\[ k++; \]
\[ \} \text{ while (} k \leq n); \]
\[ n \ast \log n = O(n \log n) \]
Sequential search

• Look at each item in the data collection in turn

• Stop when the desired item is found, or the end of the data is reached
Worst Case: $O(n)$

Avg Case: $O(n)$

Best Case: $O(1)$
Binary search

- To search a sorted array for a particular item
- Repeatedly divide the array in half
- Determine which half the item must be in, if it is indeed present, and discard the other half
Worst Case: $O(\log N)$
Avg Case: $O(\log N)$
Best Case: $O(1)$
Sorting

• Organize values (ascending or descending) by key

• Key is unique for each record
Internal reside in RAM

External reside on HD
Selection sort

• Place the largest item in its correct place
• Place the next largest item in its correct place, and so on
• Left aligned (ordering is from left to right)
Worst Case \(O(n^2)\)

Avg Case \(O(n^2)\)

Best Case \(O(n^2)\)
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>10</td>
<td>14</td>
<td>13</td>
<td>37</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>29</td>
<td>14</td>
<td>13</td>
<td>37</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>29</td>
<td>14</td>
<td>13</td>
<td>37</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>13</td>
<td>14</td>
<td>29</td>
<td>37</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>13</td>
<td>14</td>
<td>29</td>
<td>37</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>13</td>
<td>14</td>
<td>29</td>
<td>37</td>
</tr>
</tbody>
</table>
Bubble sort

• Compare adjacent elements and exchange them if they are out of order

• Will move the largest (or smallest) elements to the end of the array

• Repeating this process will eventually sort the array into ascending (or descending) order
Worst Case \( O(n^2) \)

Avg Case \( O(n^2) \)

Best Case \( O(n) \)
Insertion sort

• Partition the array into two regions: sorted and unsorted

• Take each item from the unsorted region and insert it into its correct order in the sorted region
Worst Case: $O(n^2)$
Avg Case: $O(n^2)$
Best Case: $O(n)$
<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>10</td>
<td>14</td>
<td>13</td>
<td>37</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>29</td>
<td>14</td>
<td>13</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>14</td>
<td>29</td>
<td>13</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>14</td>
<td>29</td>
<td>13</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>29</td>
<td>13</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>13</td>
<td>14</td>
<td>29</td>
<td>37</td>
</tr>
</tbody>
</table>
Merge sort

• Divide an array into halves
• Sort each half
• Merge the sorted halves into one sorted array
• Requires additional array
• Use with external data
Worst Case: $O(n\log N)$

Avg Case: $O(n\log N)$

Best Case: $O(n\log N)$
MergeSort(inout array:ItemArray,
in first:integer, int last:integer)

// sort the array by
// 1. sorting the first half
// 2. sorting the second half
// 3. merging the two halves

if (first < last) {
    mid = (first + last) / 2
    MergeSort(array, first, mid)
    MergeSort(array, mid + 1, last)

    // merge sorted halves, array[first..mid]
    // and array[mid + 1..last]
    Merge(array, first, mid, last)
}
<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3</td>
<td>5</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>5</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>10</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>22</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>22</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quick sort

- Partition an array into items that are less than the pivot and those that are greater than or equal to the pivot

- Sort the left section

- Sort the right section
Worst Case: $O(n^2)$

Avg Case: $O(n \log N)$

Best Case: $O(n \log N)$
QuickSort(inout array:ItemArray,
in first:integer, in last:integer) {
    if (first < last) {
        Choose array[0] as pivot
        Partition array[first...last] about p

        // the partition is
        // array[first..pivotIndex...last]

        // sort S1
        QuickSort(array, first, pivotIndex - 1)

        // sort S2
        QuickSort(array, pivotIndex + 1, last)
    }
}
Partition(inout array:ItemArray,
   in first:integer, in last:integer,
   out pivotIndex:integer)

\[
p = array[\text{first}]
\]
\[
lastS1 = \text{first}
\]
\[
\text{firstUnknown} = \text{first} + 1
\]

\[
\text{while (firstUnknown} \leq \text{last}) \{ \\
   \text{if (array[firstUnknown} < p) } \\
   \quad \text{Move array[firstUnknown] into S1} \\
   \text{else} \\
   \quad \text{Move array[firstUnknown] into S2} \\
\} \\
\text{swap array[first] with array[lastS1]} \\
\text{pivotIndex} = \text{lastS1}
\]
Initial state

lastS1 = first
firstUnknown = first + 1

Unknown

<table>
<thead>
<tr>
<th>p</th>
<th>?</th>
</tr>
</thead>
</table>

↑↑firstUnknown
first,
last,
lastS1
SI: swap
S2: increment

- S1 (<p)
- S2 (>=p)

First: first
Last: lastS1
First Unknown: firstUnknown

53
Swap pivot

\[ p \quad S1 (<p) \quad <p \quad S2 (>=p) \]

\[ \text{first} \quad \text{lastS1} \quad \text{last} \]

\[ p \quad S1 (<p) \quad p \quad S2 (>=p) \]

\[ \text{first} \quad \text{lastS1} \quad \text{last} \]
Radix sort

• Each number has the same number of digits
• Distribute in bins by digit (right to left)
• Collect the bins
• Repeat with next digit
• Requires more space (n-numbers * 10 bins)
Worst Case: $O(n)$
Avg Case: $O(n)$
Best Case: $O(n)$
RadixSort(inout array:ItemArray, in n:integer, in d:integer)
for (j = d down to 1) {
    Initialize 10 groups to empty
    Initialize a counter for each group to 0

    for (i = 0 through n-1) {
        k = jth digit of array[i]
        Place array[i] at the end of group k
        Increase kth counter by 1
    }

    Replace the items in array with all items in group 0, followed by all the items in group 1 and so on.
}
<table>
<thead>
<tr>
<th>0</th>
<th>100</th>
<th>004</th>
<th>009</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>123</td>
<td>023</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>034</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>240</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Summary

<table>
<thead>
<tr>
<th>Sort</th>
<th>Worst</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>$n^2$</td>
<td>$n^2$</td>
</tr>
<tr>
<td>Bubble</td>
<td>$n^2$</td>
<td>$n^2$</td>
</tr>
<tr>
<td>Insertion</td>
<td>$n^2$</td>
<td>$n^2$</td>
</tr>
<tr>
<td>Merge</td>
<td>$n\log N$</td>
<td>$n\log N$</td>
</tr>
<tr>
<td>Quick</td>
<td>$n^2$</td>
<td>$n\log N$</td>
</tr>
<tr>
<td>Radix</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Tree</td>
<td>$n\log N$</td>
<td>$n\log N$</td>
</tr>
<tr>
<td>Heap</td>
<td>$n\log N$</td>
<td>$n\log N$</td>
</tr>
</tbody>
</table>