L28. Divide and Conquer Algorithms

- Binary Search
- Merge Sort
- Mesh Generation
- Recursion

**Question**

The Manhattan phone book has 1,000,000+ entries. How is it possible to locate a name by examining just a tiny, tiny fraction of those entries?

**Answer: Repeated Halving**

To find the page containing Derek Jeter’s number...

```plaintext
while (Phone book is longer than 1 page)
    Open to the middle page.
    if “Jeter” comes before the first name,
        Rip and throw away the 2nd half.
    else
        Rip and throw away the 1st half.
    end
end
```

**What Happens to Phone Book Length...**

- Original: 3000 pages
- After 1 rip: 1500 pages
- After 2 rips: 750 pages
- After 3 rips: 375 pages
- After 4 rips: 188 pages
- After 5 rips: 94 pages
- After 12 rips: 1 page

**Binary Search**

The idea of repeatedly halving the size of the “search space” is the main idea behind the method of binary search.

An item in a sorted array of length \( n \) can be located with just \( \log_2 n \) comparisons.

**Problem**

Search a Sorted Array For a Given Value

(We assume that the array has no repeated elements.)
**Binary Search: $a = 70$**

<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>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x:</td>
<td>12</td>
<td>15</td>
<td>33</td>
<td>35</td>
<td>42</td>
<td>45</td>
<td>51</td>
<td>62</td>
<td>73</td>
<td>75</td>
<td>86</td>
</tr>
</tbody>
</table>

1. **L:** 1  
2. **Mid:** 6  
3. **R:** 12

$x(\text{Mid}) \leq a$

So throw away the left half...

**L:** 6

$a < x(\text{Mid})$

So throw away the right half...

---

**L:** 7

$x(\text{Mid}) \leq a$

So throw away the left half...

**L:** 7

$x(\text{Mid}) \leq a$

So throw away the left half...

---

```matlab
function L = BinarySearch(a,x)
% x is a row n-vector with x(1) < ... < x(n)
% x(1) <= a <= x(n)
% x(L) <= a <= x(L+1)
% x(L) <= a <= x(L+1)
% L = 1; R = length(x);
% x(L) <= a <= x(R)
while R-L > 1
    mid = floor((L+R)/2);
    if a < x(mid)
        x(L) <= a <= x(mid)
        R = mid;
    else
        x(mid) <= a <= x(R)
        L = mid;
    end
end
```
Next Problem
Sort the values in an array so that they are arranged from smallest to biggest.

Chosen Method: Merge Sort
This technique is an excellent example of a divide and conquer procedure.

Motivation
A plan if you have two “helpers”:
Split the array and have each helper sort his/her half.
Merge the two sorted subarrays.

And what if those two helpers each had two sub-helpers? Etc

Subdivide the Sorting Task

Subdivide Again

And Again

And One Last Time
Now Merge

And Merge Again

And Again

And One Last Time

Done!

function y = MergeSort(x)
% x is a column n-vector.
% y is a column n-vector consisting % of the values in x sorted % from smallest to largest.

n = length(x);
if n==1
    y = x;
else
    m = floor(n/2);
    y1 = MergeSort(x(1:m));
    y2 = MergeSort(x(m+1:n));
    y = Merge(y1,y2);
end
Important Sub-Problem

Merging Two Sorted Arrays
Into a
Single Sorted Array

Example

\[
\begin{align*}
\text{x:} & \quad 12 \quad 33 \quad 35 \quad 45 \\
\text{y:} & \quad 15 \quad 42 \quad 55 \quad 65 \quad 75 \\
\text{z:} & \quad 12 \quad 15 \quad 33 \quad 35 \quad 42 \quad 45 \quad 55 \quad 65 \quad 75
\end{align*}
\]

Merge

\[
\begin{align*}
x: & \quad 12 \quad 33 \quad 35 \quad 45 \\
y: & \quad 15 \quad 42 \quad 55 \quad 65 \quad 75 \\
z: & \quad \text{merge} \\
x(ix) \leq y(iy) & \quad ?? \\
\end{align*}
\]

Merge

\[
\begin{align*}
x: & \quad 12 \quad 33 \quad 35 \quad 45 \\
y: & \quad 15 \quad 42 \quad 55 \quad 65 \quad 75 \\
z: & \quad \text{merge} \\
x(ix) \leq y(iy) & \quad \text{yes}
\end{align*}
\]

Merge

\[
\begin{align*}
x: & \quad 12 \quad 33 \quad 35 \quad 45 \\
y: & \quad 15 \quad 42 \quad 55 \quad 65 \quad 75 \\
z: & \quad \text{merge} \\
x(ix) \leq y(iy) & \quad ?? \\
\end{align*}
\]

Merge

\[
\begin{align*}
x: & \quad 12 \quad 33 \quad 35 \quad 45 \\
y: & \quad 15 \quad 42 \quad 55 \quad 65 \quad 75 \\
z: & \quad \text{merge} \\
x(ix) \leq y(iy) & \quad \text{no}
\end{align*}
\]
Merge

x: 12 33 35 45
ix: 2

y: 15 42 55 65 75
iy: 2

z: 12 15  
iz: 3

x[ix] <= y[iy]  ???

Merge

x: 12 33 35 45
ix: 2

y: 15 42 55 65 75
iy: 2

z: 12 15 33  
iz: 3

x[ix] <= y[iy]  yes

Merge

x: 12 33 35 45
ix: 3

y: 15 42 55 65 75
iy: 2

z: 12 15 33  
iz: 4

x[ix] <= y[iy]  ???

Merge

x: 12 33 35 45
ix: 3

y: 15 42 55 65 75
iy: 2

z: 12 15 33  
iz: 4

x[ix] <= y[iy]  yes

Merge

x: 12 33 35 45
ix: 4

y: 15 42 55 65 75
iy: 2

z: 12 15 33  
iz: 5

x[ix] <= y[iy]  ???

Merge

x: 12 33 35 45
ix: 4

y: 15 42 55 65 75
iy: 2

z: 12 15 33  
iz: 5

x[ix] <= y[iy]  no
Merge

x: 12 33 35 45
ty: 15 42 55 65 75
z: 12 15 33 35 42

ix: 4
iy: 3
iz: 5

x(ix) <= y(iy)   ???

ix > 4

ix > 4

ix > 4

ix > 4
function z = Merge(x,y)
n = length(x); m = length(y);
z = zeros(1,n+m);
ix = 1; iy = 1;
for iz=1:(n+m)
    if ix > n
        z(iz) = y(iy); iy = iy+1;
    elseif iy>m
        z(iz) = x(ix); ix = ix + 1;
    elseif x(ix) <= y(iy)
        z(iz) = x(ix); ix = ix + 1;
    else
        z(iz) = y(iy); iy = iy + 1;
    end
end

New Problem
Recursive Mesh Generation

Divide and Conquer
Methods Also Show Up
in Geometric Situations

Mesh Generation

Chop a Region
up into triangles
with smaller
triangles
in “areas of interest”.

Step One in simulating flow around an airfoil is to generate a
mesh and (say) estimate velocity at each mesh point.
Mesh Generation in 3D

Why is Mesh Generation a Divide & Conquer Process?

Let Us Draw this Graphic:

The Basic Operation

if the triangle is big enough
   Connect the midpoints.
   Color the interior triangle mauve.
else
   Color the whole triangle yellow.
end

At the Start...

Recur On this Idea

Recur Again

Apply same idea to lower left triangle
function MeshTriangle(x,y,L)
    if L==0
        % No subdivision required...
        fill(x,y,'y','linewidth',1.5)
    else
        % A subdivision is called for. Get midpoints...
        a = [(x(1)+x(2))/2 (x(2)+x(3))/2 (x(3)+x(1))/2];
        b = [(y(1)+y(2))/2 (y(2)+y(3))/2 (y(3)+y(1))/2];
        % Color the interior triangle magenta...
        fill(a,b,'m','linewidth',1.5)
        % Apply the process to the three "outer" triangles...
        MeshTriangle([x(1) a(1) a(3)],[y(1) b(1) b(3)],L-1)
        MeshTriangle([x(2) a(2) a(1)],[y(2) b(2) b(1)],L-1)
        MeshTriangle([x(3) a(3) a(2)],[y(3) b(3) b(2)],L-1)
end