Divide And Conquer

- Distinguish between small and large instances.
- Small instances solved differently from large ones.

Small And Large Instance

- Small instance.
  - Sort a list that has n <= 10 elements.
  - Find the minimum of n <= 2 elements.
- Large instance.
  - Sort a list that has n > 10 elements.
  - Find the minimum of n > 2 elements.

Solving A Small Instance

- A small instance is solved using some direct/simple strategy.
  - Sort a list that has n <= 10 elements.
    - Use count, insertion, bubble, or selection sort.
  - Find the minimum of n <= 2 elements.
    - When n = 0, there is no minimum element.
    - When n = 1, the single element is the minimum.
    - When n = 2, compare the two elements and determine which is smaller.

Solving A Large Instance

- A large instance is solved as follows:
  - Divide the large instance into k >= 2 smaller instances.
  - Solve the smaller instances somehow.
  - Combine the results of the smaller instances to obtain the result for the original large instance.

Sort A Large List

- Sort a list that has n > 10 elements.
  - Sort 15 elements by dividing them into 2 smaller lists.
    - One list has 7 elements and the other has 8.
  - Sort these two lists using the method for small lists.
  - Merge the two sorted lists into a single sorted list.

Find The Min Of A Large List

- Find the minimum of 20 elements.
  - Divide into two groups of 10 elements each.
  - Find the minimum element in each group somehow.
  - Compare the minimums of each group to determine the overall minimum.

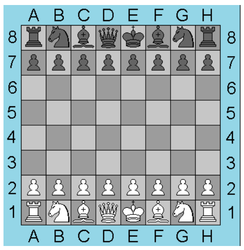
Recursion In Divide And Conquer

- Often the smaller instances that result from the divide step are instances of the original problem (true for our sort and min problems). In this case,
  - If the new instance is a small instance, it is solved using the method for small instances.
  - If the new instance is a large instance, it is solved using the divide-and-conquer method recursively.
- Generally, performance is best when the smaller instances that result from the divide step are of approximately the same size.

Recursive Find Min

- Find the minimum of 20 elements.
  - Divide into two groups of 10 elements each.
  - Find the minimum element in each group recursively. The recursion terminates when the number of elements is <= 2. At this time the minimum is found using the method for small instances.
  - Compare the minimums of the two groups to determine the overall minimum.

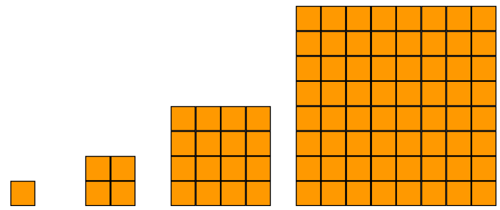
Tiling A Defective Chessboard



A real chessboard!

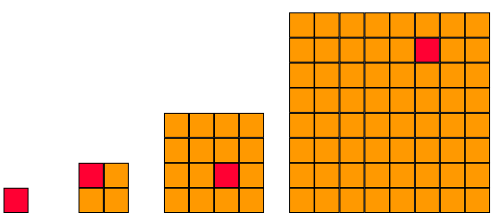
Our Definition Of A Chessboard

A chessboard is an n x n grid, where n is a power of 2.



A Defective Chessboard

A defective chessboard is a chessboard that has one unavailable (defective) position.



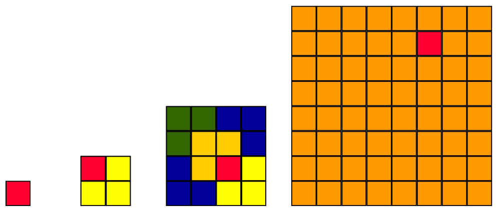
A Triomino

A triomino is an L shaped object that can cover three squares of a chessboard.

A triomino has four orientations.

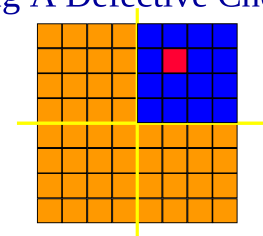
Tiling A Defective Chessboard

Place $(n^2 - 1)/3$ triominoes on an n x n defective chessboard so that all $n^2 - 1$ nondefective positions are covered.



1x1 2x2 4x4 8x8

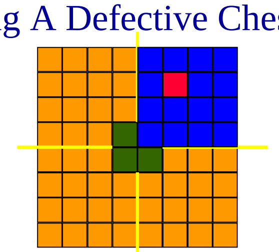
Tiling A Defective Chessboard



Divide into four smaller chessboards. 4 x 4

One of these is a defective 4 x 4 chessboard.

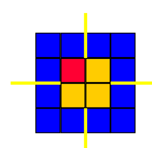
Tiling A Defective Chessboard



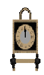
Make the other three 4 x 4 chessboards defective by placing a triomino at their common corner.

Recursively tile the four defective 4 x 4 chessboards.

Tiling A Defective Chessboard



Complexity



- Let $n = 2^k$.
- Let $t(k)$ be the time taken to tile a $2^k \times 2^k$ defective chessboard.
- $t(0) = d$, where d is a constant.
- $t(k) = 4t(k-1) + c$, when $k > 0$. Here c is a constant.
- Recurrence equation for $t()$.

Substitution Method

$$\begin{aligned}
t(k) &= 4t(k-1) + c \\
&= 4[4t(k-2) + c] + c \\
&= 4^2\, t(k-2) + 4c + c \\
&= 4^2[4t(k-3) + c] + 4c + c \\
&= 4^3\, t(k-3) + 4^2c + 4c + c \\
&= \ldots \\
&= 4^k\, t(0) + 4^{k-1}c + 4^{k-2}c + \ldots + 4^2c + 4c + c \\
&= 4^k\, d + 4^{k-1}c + 4^{k-2}c + \ldots + 4^2c + 4c + c \\
&= \text{Theta}(4^k) \\
&= \text{Theta(number of triominoes placed)}
\end{aligned}$$

Min And Max

Find the lightest and heaviest of n elements using a balance that allows you to compare the weight of 2 elements.

Minimize the number of comparisons.

Max Element

- Find element with max weight from w[0:n-1].

```
maxElement = 0;
for (int i = 1; i < n; i++)
   if (w[maxElement] < w[i]) maxElement = i;
```

- Number of comparisons of w values is $n-1$.

Min And Max

- Find the max of n elements making n-1 comparisons.
- Find the min of the remaining n-1 elements making n-2 comparisons.
- Total number of comparisons is 2n-3.

Divide And Conquer

- Small instance.
  - n <= 2.
  - Find the min and max element making at most one comparison.

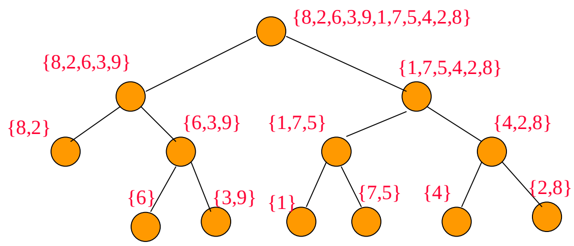
Large Instance Min And Max

- n > 2.
- Divide the n elements into 2 groups A and B with floor(n/2) and ceil(n/2) elements, respectively.
- Find the min and max of each group recursively.
- Overall min is min{min(A), min(B)}.
- Overall max is max{max(A), max(B)}.

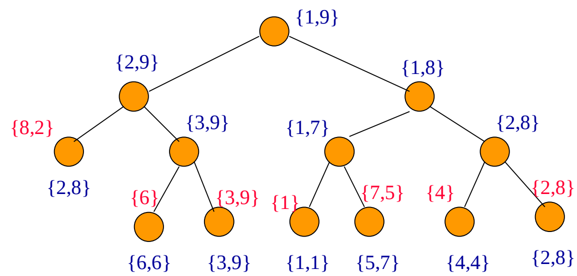
Min And Max Example

- Find the min and max of {3,5,6,2,4,9,3,1}.
- Large instance.
- A = {3,5,6,2} and B = {4,9,3,1}.
- min(A) = 2, min(B) = 1.
- max(A) = 6, max(B) = 9.
- min{min(A),min(B)} = 1.
- max{max(A), max(B)} = 9.

Dividing Into Smaller Instances



Solve Small Instances And Combine



Time Complexity

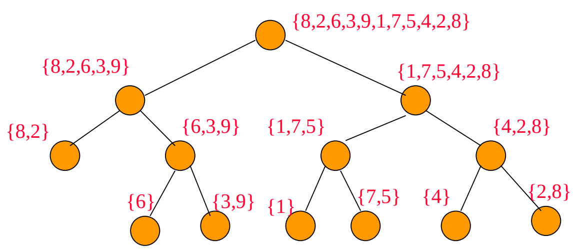
- Let c(n) be the number of comparisons made when finding the min and max of n elements.
- c(0) = c(1) = 0.
- c(2) = 1.
- When n > 2,

  c(n) = c(floor(n/2)) + c(ceil(n/2)) + 2
- To solve the recurrence, assume n is a power of 2 and use repeated substitution.
- c(n) = ceil(3n/2) - 2.

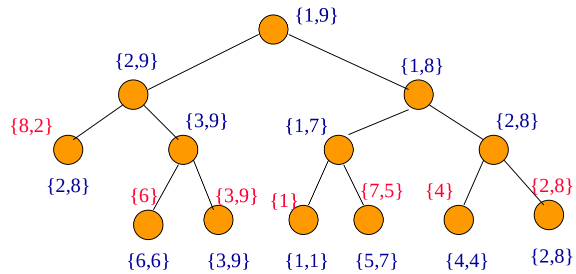
Interpretation Of Recursive Version

- The working of a recursive divide-and-conquer algorithm can be described by a tree—recursion tree.
- The algorithm moves down the recursion tree dividing large instances into smaller ones.
- Leaves represent small instances.
- The recursive algorithm moves back up the tree combining the results from the subtrees.
- The combining finds the min of the mins computed at leaves and the max of the leaf maxs.

Downward Pass Divides Into Smaller Instances



Upward Pass Combines Results From Subtrees



Iterative Version

- Start with n/2 groups with 2 elements each and possibly 1 group that has just 1element.
- Find the min and max in each group.
- Find the min of the mins.
- Find the max of the maxs.

Iterative Version Example

- {2,8,3,6,9,1,7,5,4,2,8 }
- {2,8}, {3,6}, {9,1}, {7,5}, {4,2}, {8}
- mins = {2,3,1,5,2,8}
- maxs = {8,6,9,7,4,8}
- minOfMins = 1
- maxOfMaxs = 9

Comparison Count

- Start with n/2 groups with 2 elements each and possibly 1 group that has just 1element.
  - No compares.
- Find the min and max in each group.
  - floor(n/2) compares.
- Find the min of the mins.
  - ceil(n/2) - 1 compares.
- Find the max of the maxs.
  - ceil(n/2) - 1 compares.
- Total is ceil(3n/2) - 2 compares.

Initialize A Heap

- n > 1:
  - Initialize left subtree and right subtree recursively.
  - Then do a trickle down operation at the root.
- t(n) = c, n <= 1.
- t(n) = 2t(n/2) + d * height, n > 1.
- c and d are constants.
- Solve to get t(n) = O(n).
- Implemented iteratively in Chapter 13.

Initialize A Loser Tree

- n > 1:
  - Initialize left subtree.
  - Initialize right subtree.
  - Compare winners from left and right subtrees.
  - Loser is saved in root and winner is returned.
- t(n) = c, n <= 1.
- t(n) = 2t(n/2) + d, n > 1.
- c and d are constants.
- Solve to get t(n) = O(n).
- Implemented iteratively in Chapter 14.