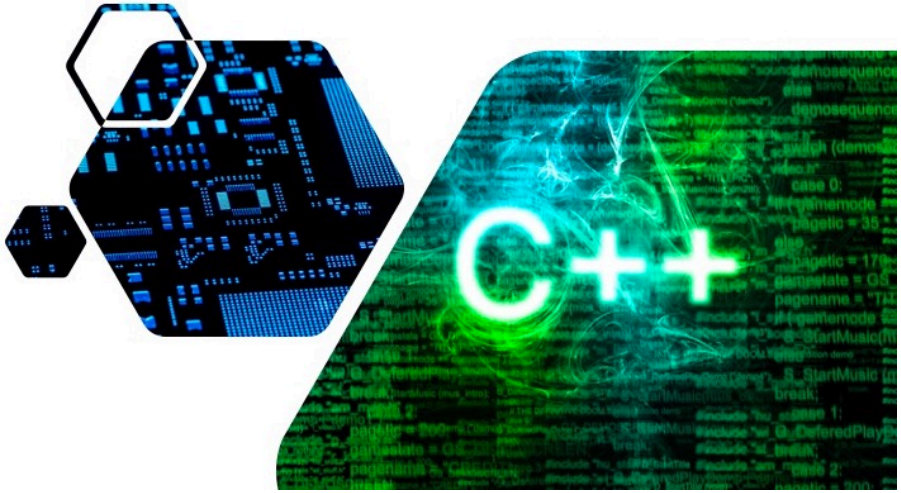
Lecture 2: Divide And Conquer (D&C) Algorithms

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

For those who didn't attend the course last time …



The course teaches you how to write "good" programs with a specific focus on solving computer design problems. You will gain "hands-on" experience in writing C++/Python code for implementing important data structures and algorithms …

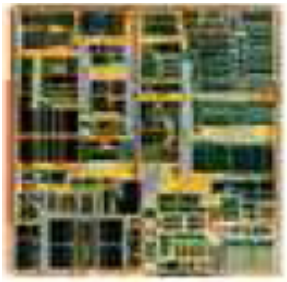
Design Automation is Driving Chip Evolution

1982 – Intel 80286
134K transistors
12MHz; 68.7 mm^2

1985 – Intel 80386
275K transistors
33MHz; 104 mm^2

1989 – Intel 80486
1.2M transistors
50MHz; 163 mm^2

1993 – Intel Pentium
3.1M transistors
66MHz; 264 mm^2



1997 – Intel Pentium II
7.5M transistors
300MHz; 209 mm^2

1999 – Intel Pentium III
28M transistors
733MHz; 140 mm^2

2000 – Intel Pentium4
42M transistors
1.5MHz; 224 mm^2

~~*Advanced*~~ *Applied Programming for Computer Design Problems*

More information can be found in lecture 1

Some Questions about the Course

- **How can I find more materials about C++ learning?**
  - Cpp-now: https://www.youtube.com/user/BoostCon
  - Cpp-conf: https://www.youtube.com/user/CppCon
  - Cpp-reference: https://en.cppreference.com/w/
- **How to know my code is good?**
  - Ask experienced coder to read your code
  - Dive into the assembly: https://godbolt.org/
- **What hardware knowledge do I need?**
- **How does this course help me find a job in AI?**

Reflect on Big Mod Problem

- **Naïve method**
  - $2^{16} = 2*2*2*2*2*2*\ldots*2$ total 15 calculation
- **Can we do better?**
  - $2^{16} = (2^8)^2$ total 1 calculation
  - $2^8 = (2^4)^2$ total 1 calculation
  - $2^4 = (2^2)^2$ total 1 calculation
  - $2^2 = (2^1)^2$ total 1 calculation
  - $2^1 = (2^0)$ total 1 calculation

Refined Solution

```
#include <iostream>
using namespace std;

#define SQUARE(x) (x*x)
int bigmod(int a, int b, int c)
{
   if(b==0) return 1;

   else if(b%2==0)
      return SQUARE(bigmod(a,b/2,c)%c)%c;

   else return ((a%c)*bigmod(a,b-1,c))%c;
}

int main()
{
   int a,b,c;

   while(std::cin >> a >> b >> c)  std::cout<<bigmod(a,b,c)<<endl;
   return 0;
}
```

DO NOT globalize the namespace!!!

In fact, there is a name for the refined method:

Divide and Conquer algorithm

(used to solve >50% computer design problems, including Google's distributed system solutions, e.g., MapReduce)

An Iterative Version

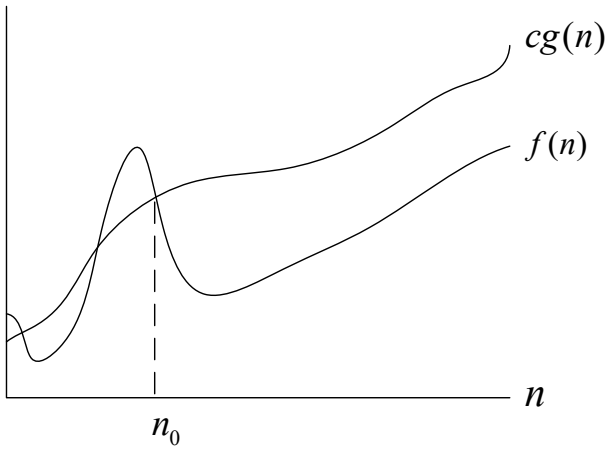
- **All recursive problems can be rewritten iteratively**

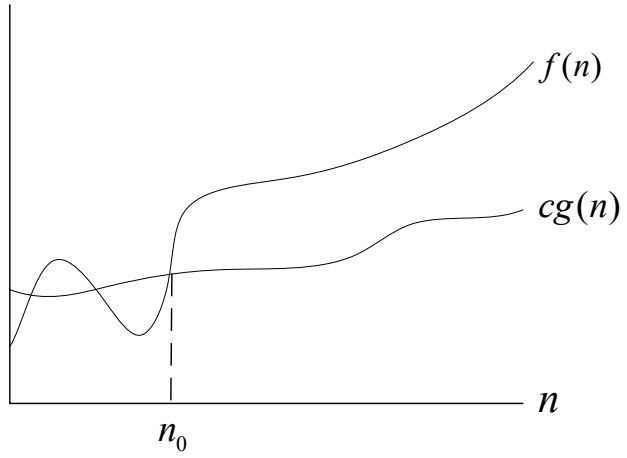
```
int64_t bigmod(int64_t a, int64_t b, int64_t c) {
  int64_t result = 1;
  while(b > 0) {
    if(b & 1) {
      result = (result * a) % c;
    }
    b = b >> 1;
    a = a*a%c;
  }
  return result;
}
```

See: https://godbolt.org/z/RVxZRN

Get a Sense of Runtime Performance

- **Time complexity analysis**
  - Runtime plays the most important role in performance
  - In algorithm theory, we use big O worst case analysis
    - Upper bound of the runtime with respect to input size N





In this course, we will use TW's empirical analysis as with modern computer architectures

Empirical Analysis

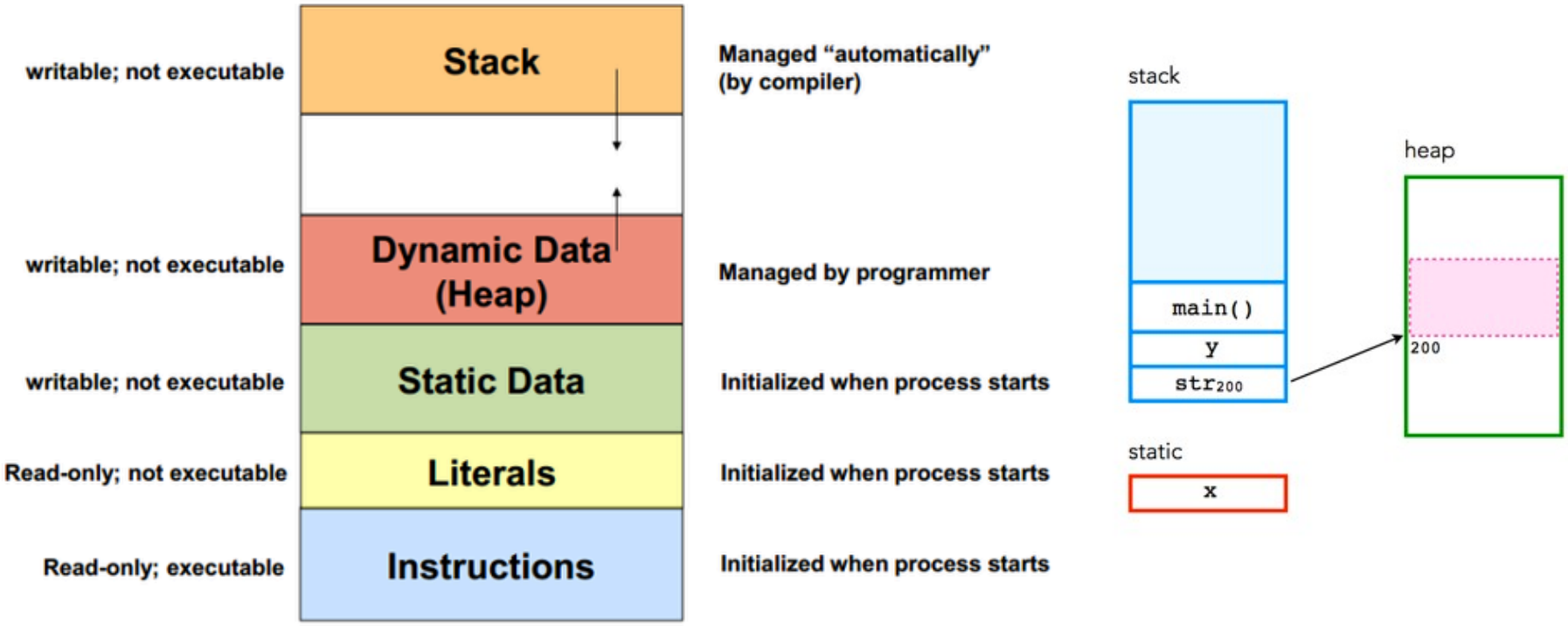
- **Technical Analysis(Input Data Size = N)**
  - O(N)
    - for(int i=1; i<=N; i++) some_constant_work();
  - $O(N^2)$
    - for(int i=1; i<=N; i++)
      for(int j=1; j<=N; j++)
        some_constant_time_work();
  - $O(N^3)$, $O(N^4)$…
- **On modern computers, input Data Size = N**
  - Timing Constraint
    - O(N) = 1000000 ~ 8000000 equals to Run Time < 1s
    - Generally true on modern PCs

Empirical Analysis Examples

- **A Problem with N = 1000, Time Limit = 1s**
  - Could a O(N) algorithm pass the time limit constraint?
  - What about the O(NlogN)?
  - What about the $O(N^2)$?
  - What about the $O(N^2 \log N)$?
  - What about the O(1) time ?
- **A Problem with N = 1000000, Time Limit = 1s**
  - Could a O(N) algorithm pass the time limit constraint?
  - What about the O(logN) ?
  - What about the O(logN logN) ?
  - What about the $O(N^2)$?

What about Space Complexity?

- **Same as the empirical runtime analysis**
  - However, it depends on your RAM size
- **Stack vs Heap**



Before we talk about D&C …

- Recursive program is the backbone of D&C
- Fibonacci number

1,1,2,3,5,8,13,21,34,55,89,144,233,377…

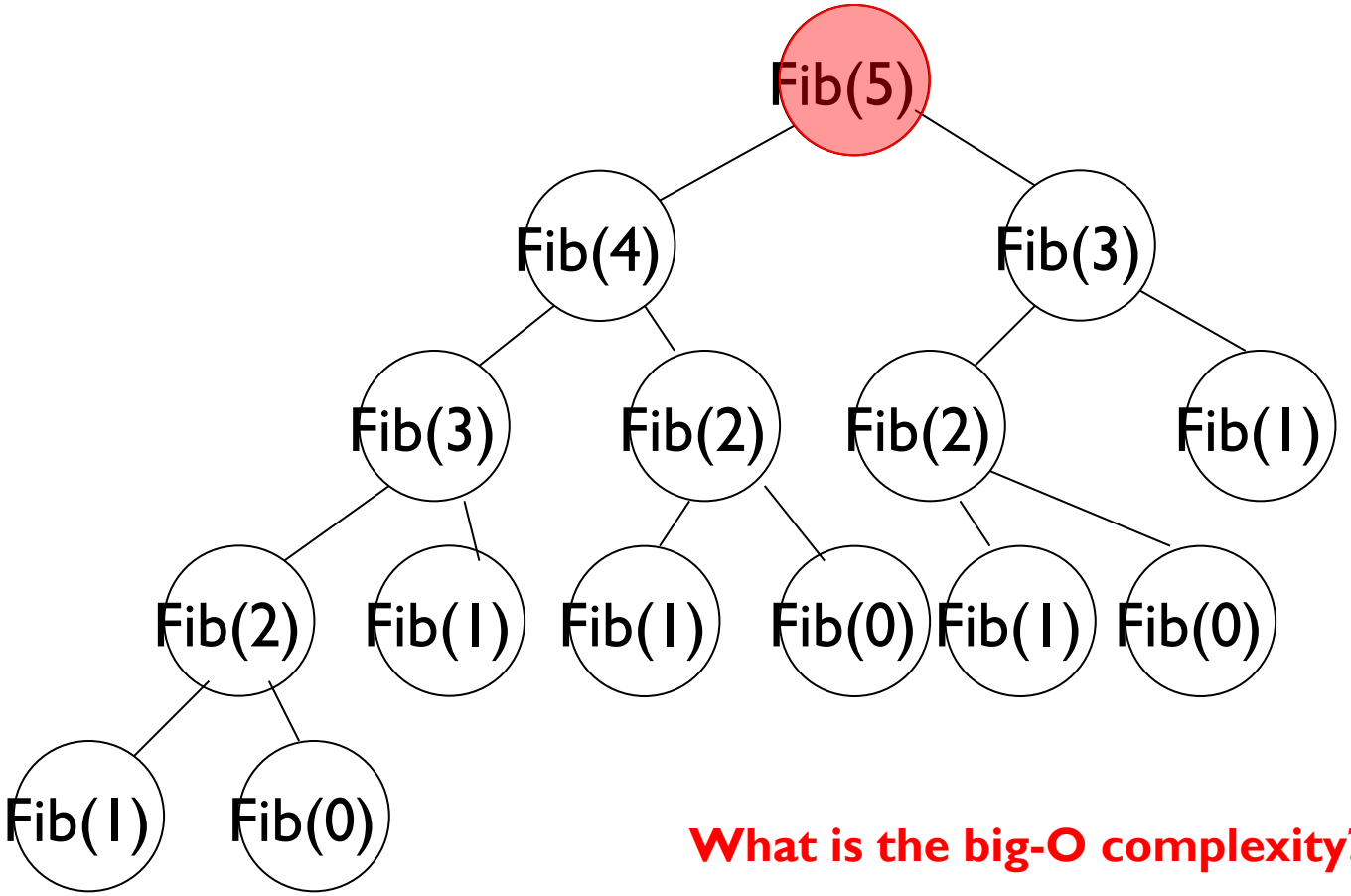
1+1=2
1+2=3
2+3=5
3+5=8
5+8=13
8+13=21
13+21=34
21+34=55
34+55=89
55+89=144
89+144=233
144+233=377

A Simple Recursive Solution

```
int fib_top_down(int n)
{
        if(n<=2) return 1;
        return fib_top_down(n-1)+fib_top_down(n-2);
}
```

What is the problem?

Duplicate Computations …



What is the big-O complexity?

Tabular Method

- **Problems**
  - Duplicate calculation
  - F(30) will blow up the runtime
- **Solve by table Weapon**
  - **Four Steps**
    - **(1) Basic and Valid Condition**
    - **(2) Is Found Condition**
    - **(3) Recursive Part**
    - **(4) Return Value Part**

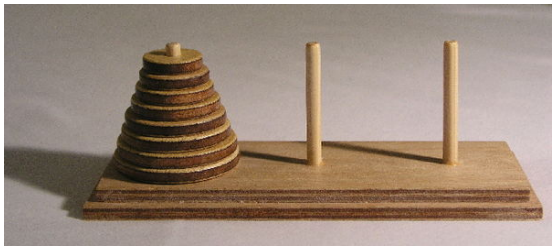
```
int fib_top_down(int n)
{
        if(n<=2) return 1;
        return fib_top_down(n-1)+fib_top_down(n-2);
}
```

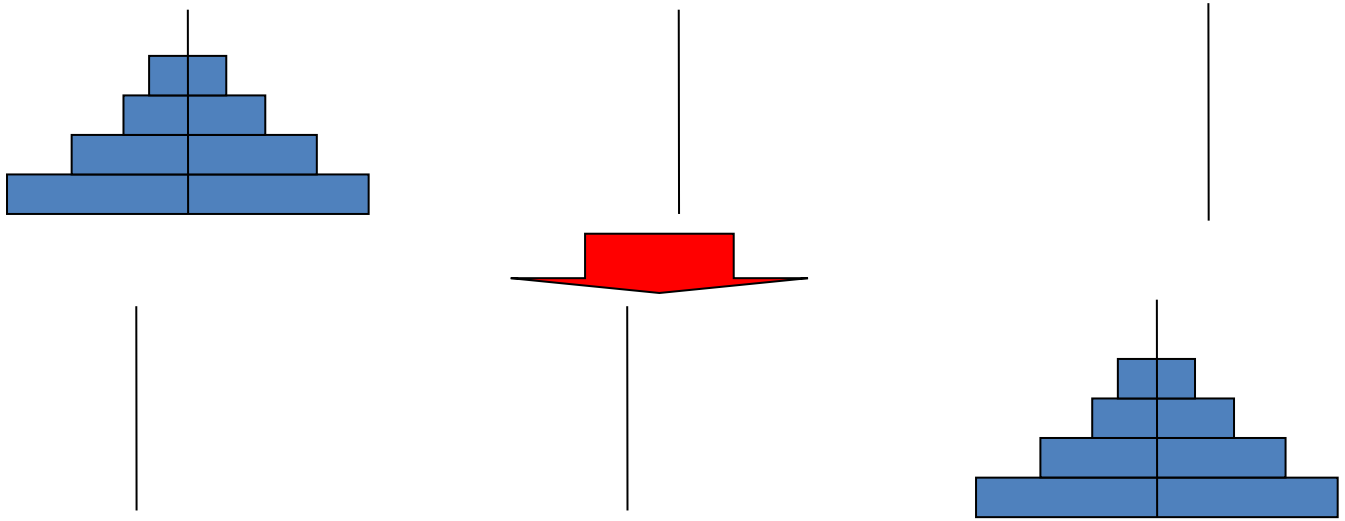
```
bool isfind[MAXN];   //all initialized to be false
int fib[MAXN];       //all initialized to be zero

int tabular(int n)
{
        if(n<=2) return 1;
        if(isfind[n]) return fib[n];
        fib[n] = tabular(n-1) + tabular(n-2);
        isfind[n] = true;
        return fib[n];
}
```

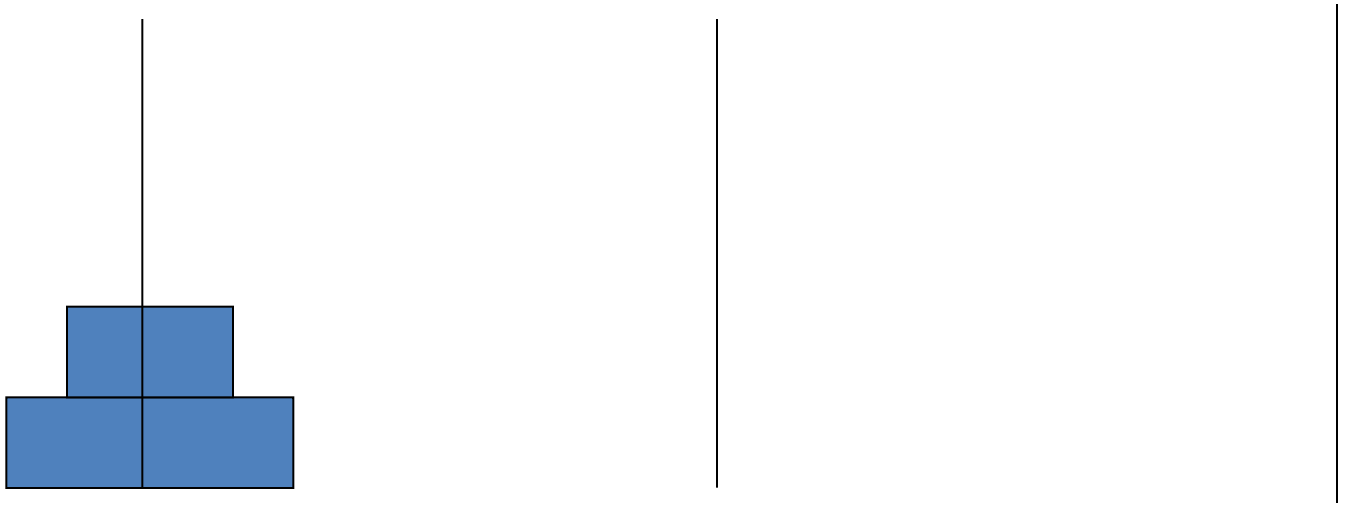
The Hanoi Problem

- Given three sticks, what's the number of steps required to move the tower of n blocks from the first peg to the last peg. You cannot put a large block on the small block



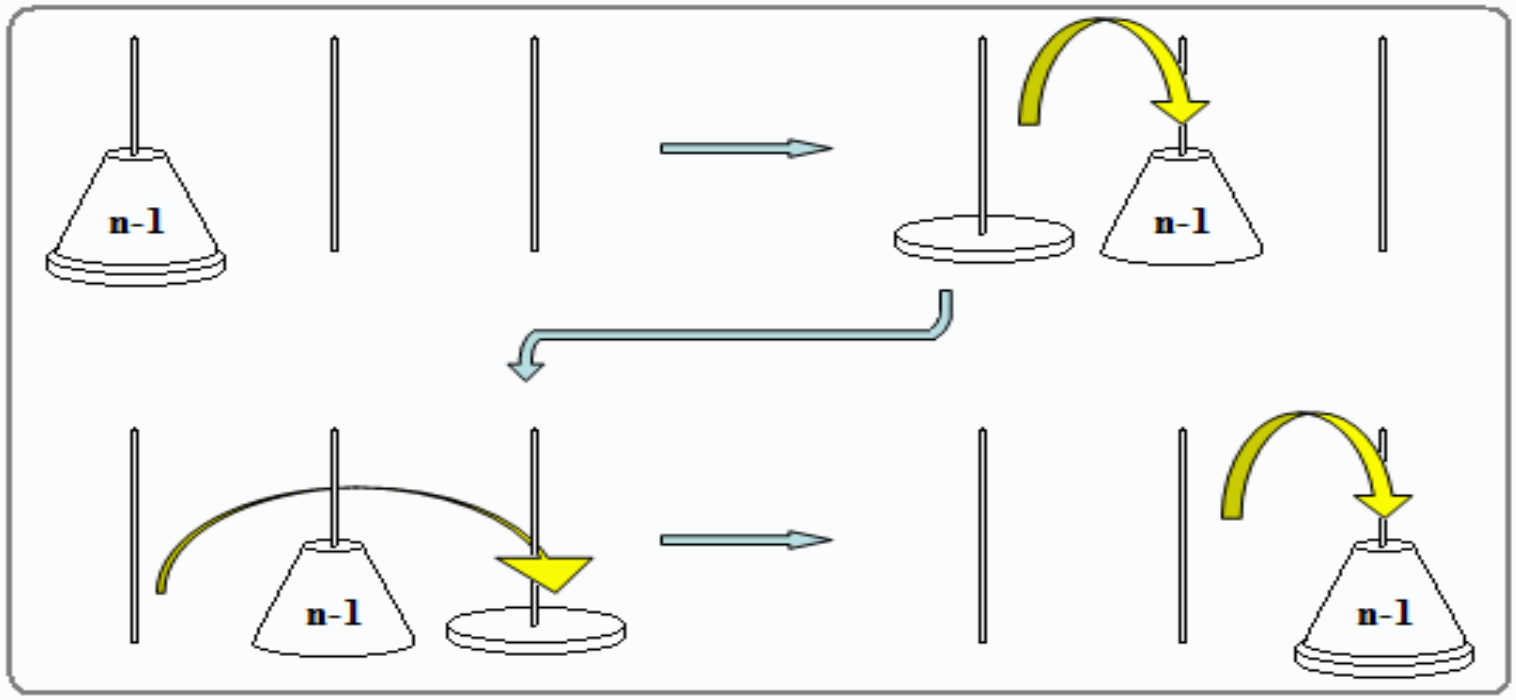


Example of Two Blocks



Example with a 2-block tower.

Recursion Formula

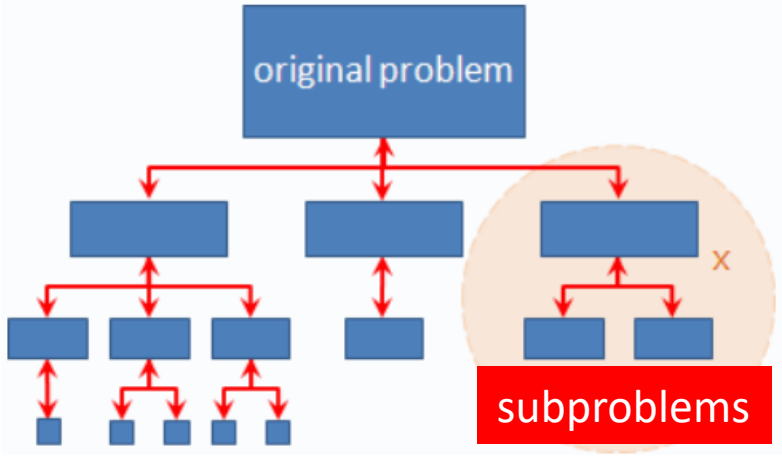


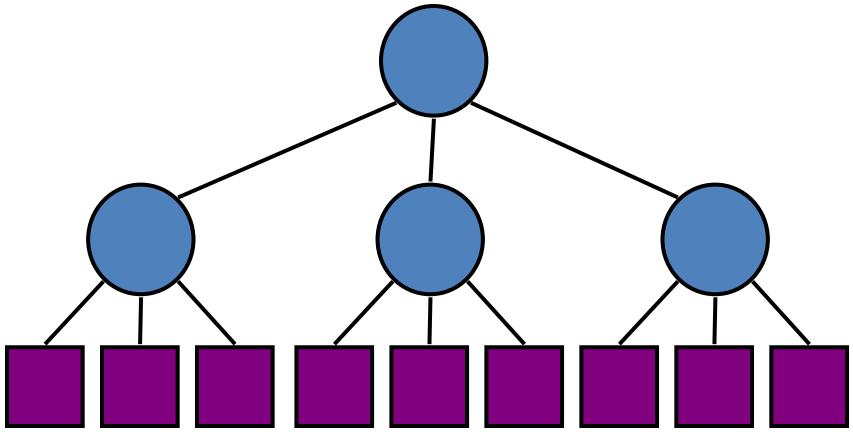
Tower of Hanoi
Counting the movements

$$f(n) = f(n-1) + 1 + f(n-1)$$
$$= 2 * f(n-2) + 1$$

Divide and Conquer

- **Divide & Conquer**
  - Divide: Divide the Original Problem Size into Small Problem Set
  - Recursive: Solve the Problem Recursively
  - Conquer: Combine these Problem Set
- **Illustration**





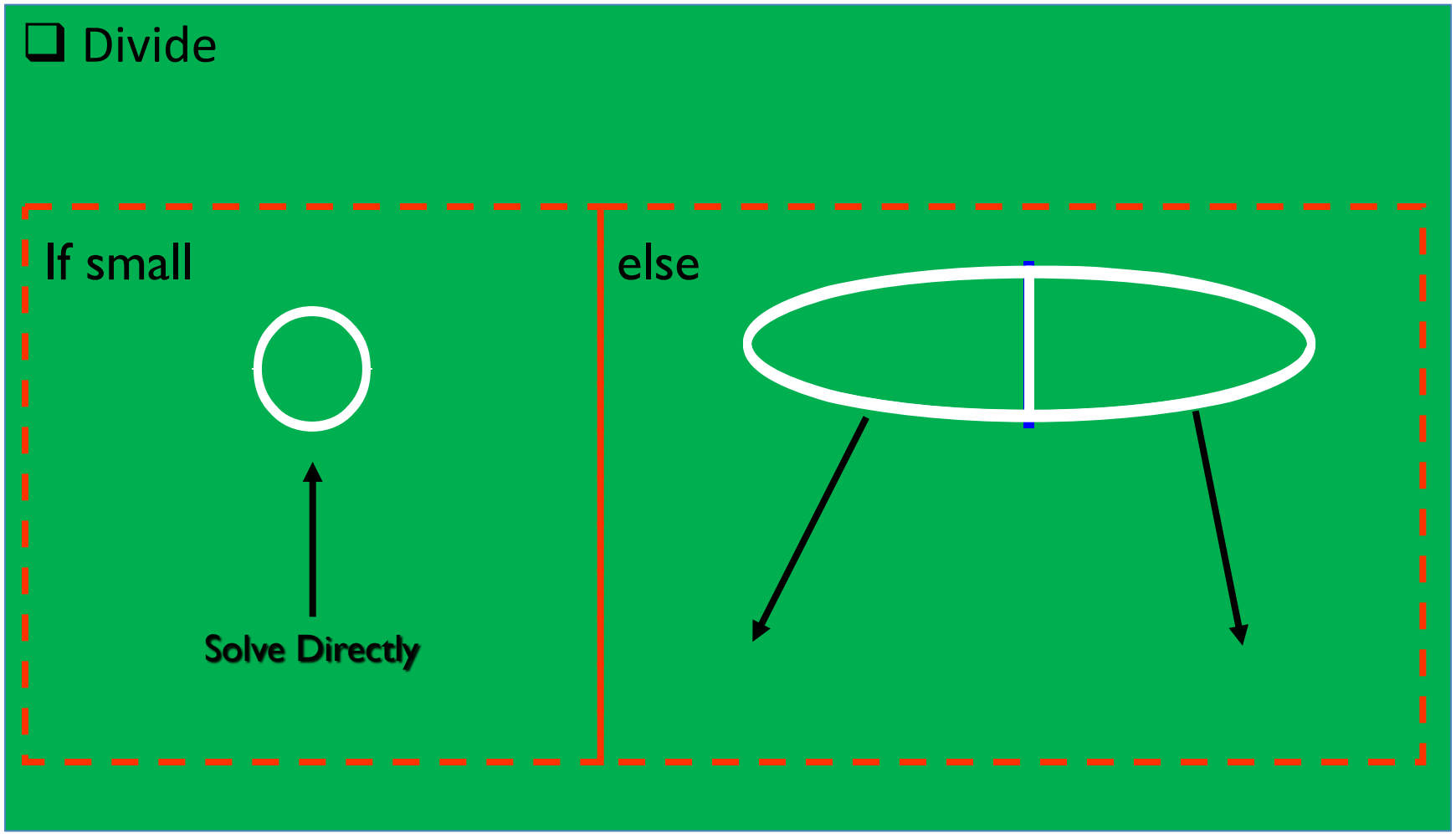
Time Complexity of Divide and Conquer

- **Time Complexity Analysis of D&C**
  - Usually Nlog(N) time

$$T(n) = \begin{cases} k * T(\frac{n}{k}) + time(s) + time(m), n > given\_size \\ b, n < given\_size \end{cases}$$

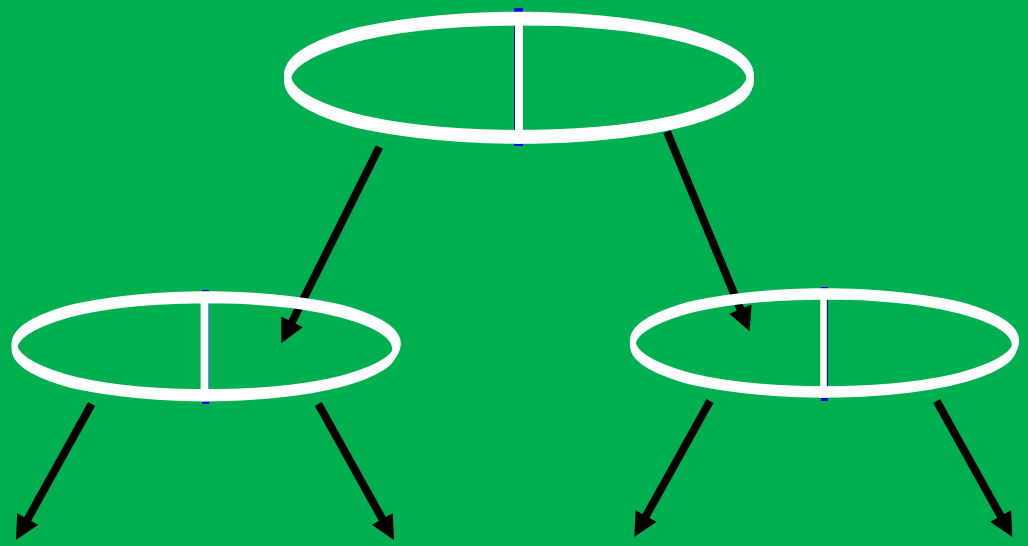
  - k: the divided size
  - s: split
  - m: merge

Visualization of Divide and Conquer



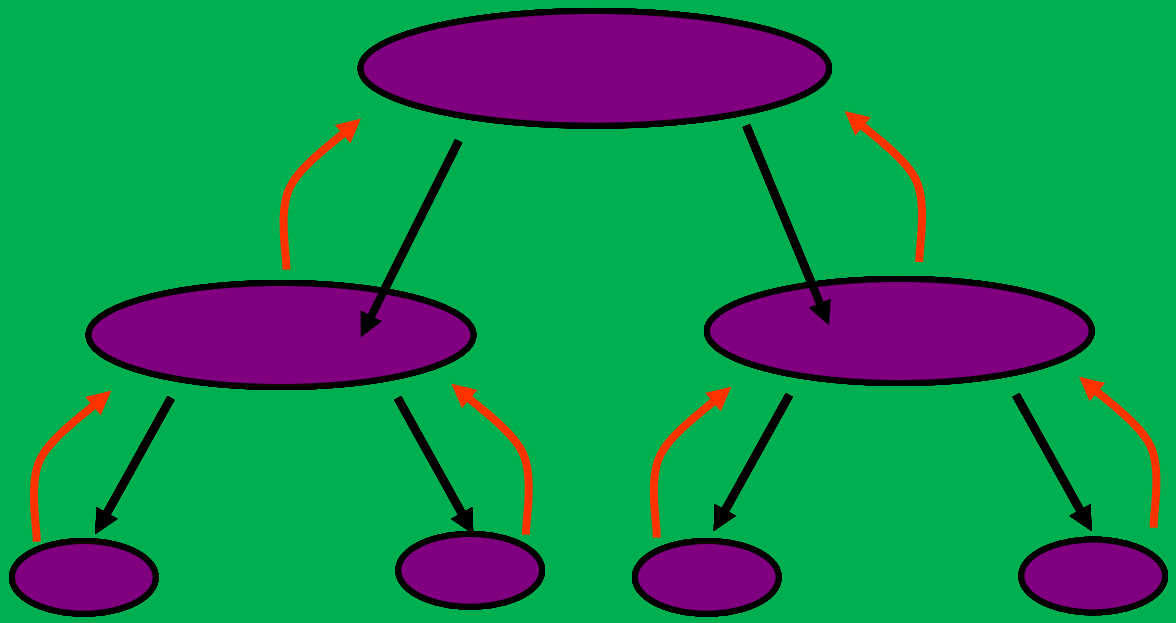
Visualization of Divide and Conquer

- Recursive



Visualization of Divide and Conquer

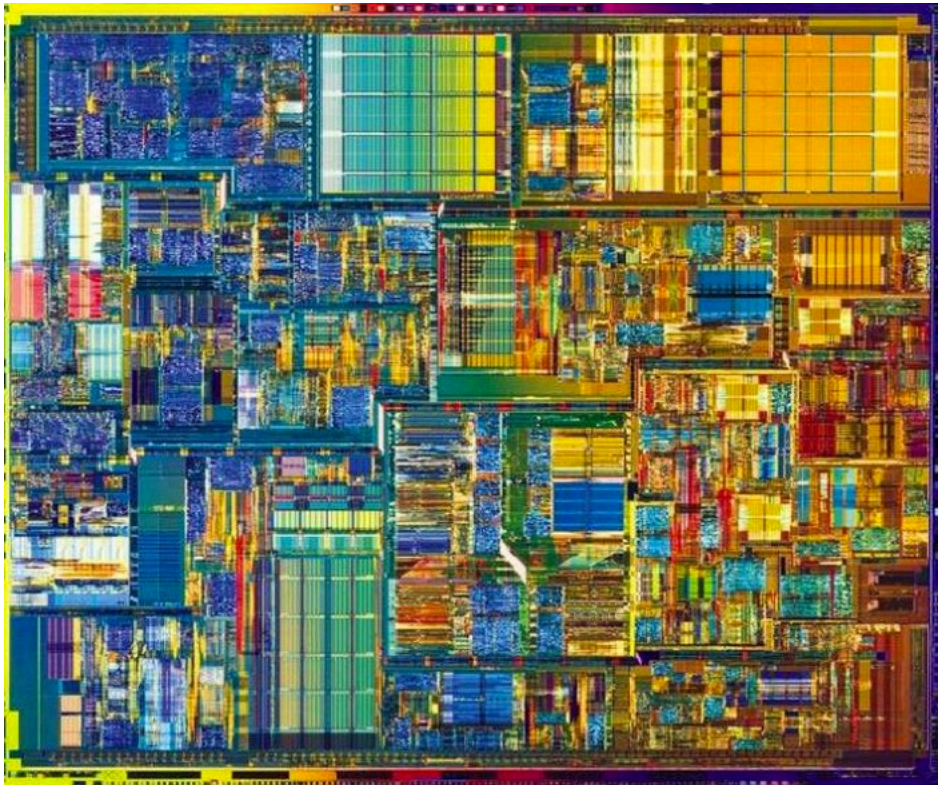
- Solve and Conquer



Divide and Conquer is Heavily used in CAD

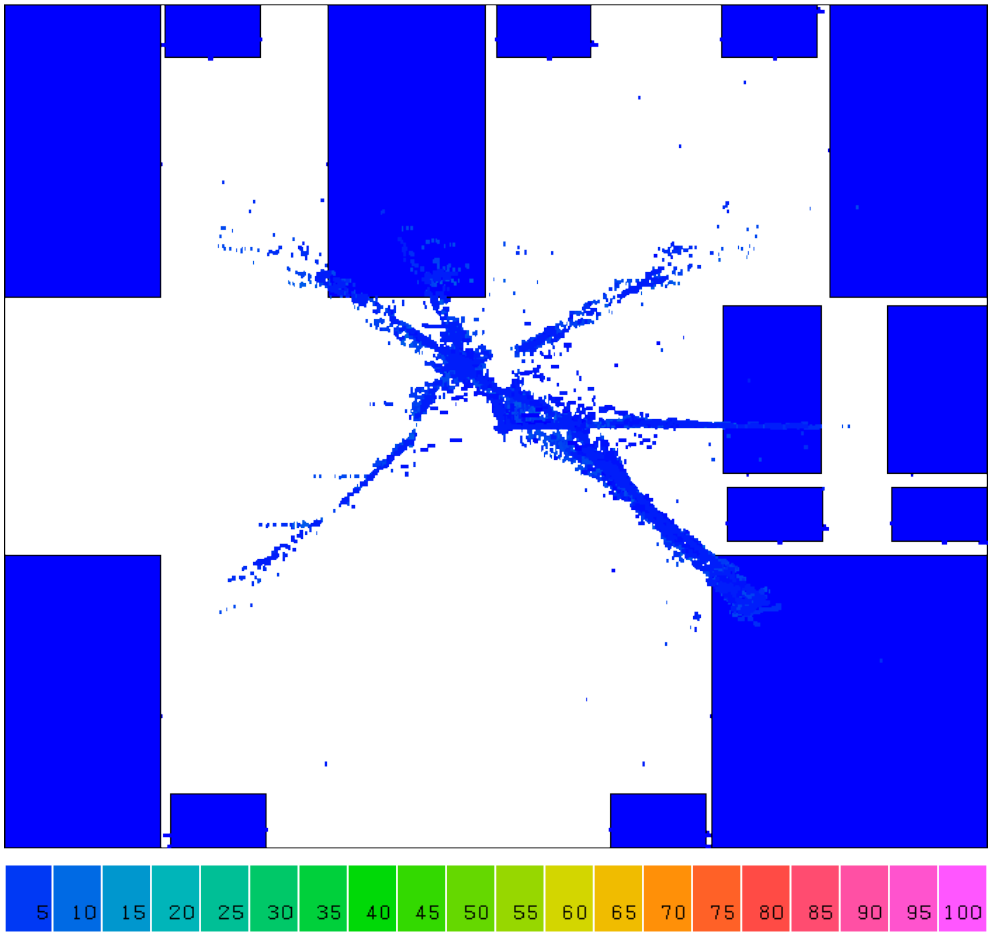
- **Modern circuits sizes are too large to handle in flat**

2000 – Intel Pentium4
42M transistors
1.5MHz; 224 mm^2

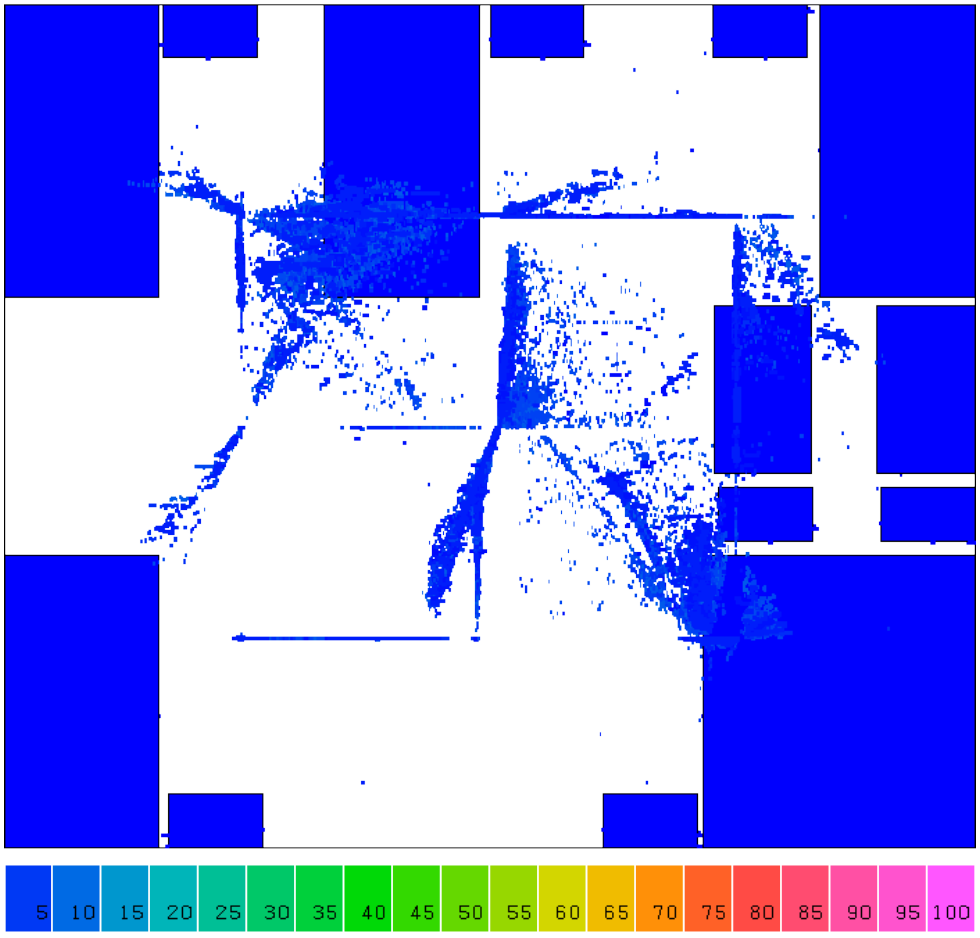


Directly solving the original problem takes forever to finish …

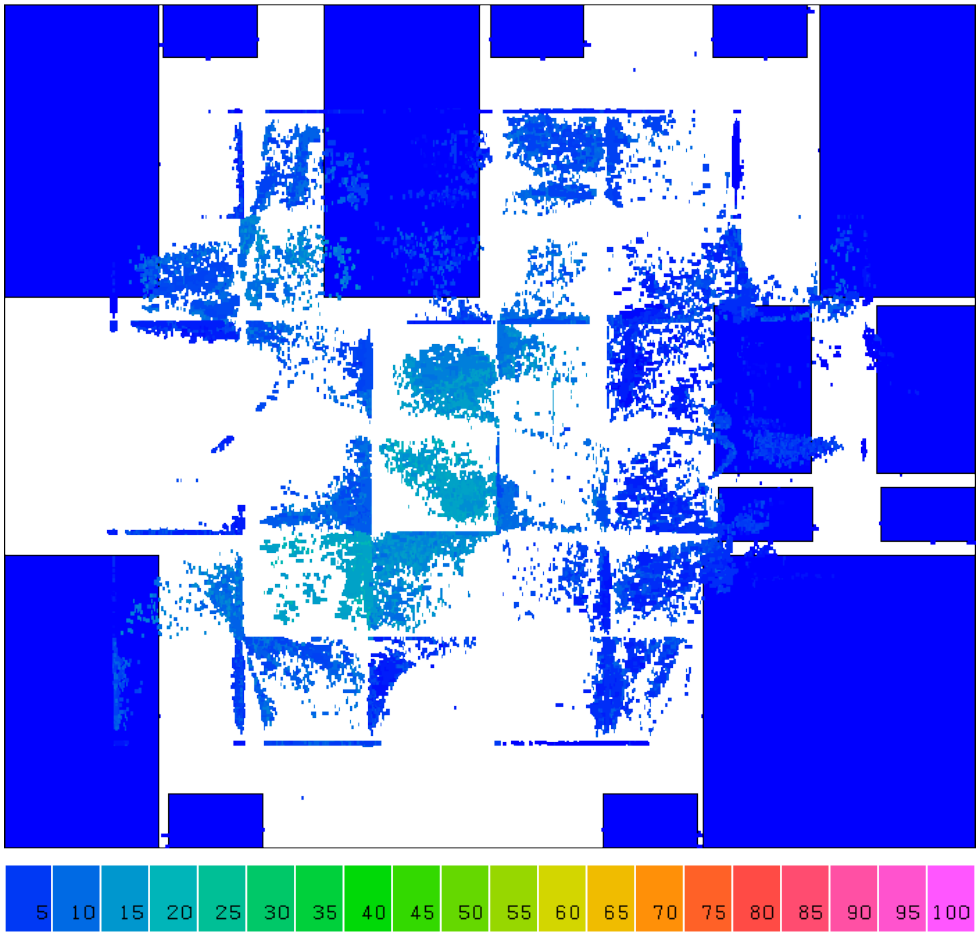
Example: Placement



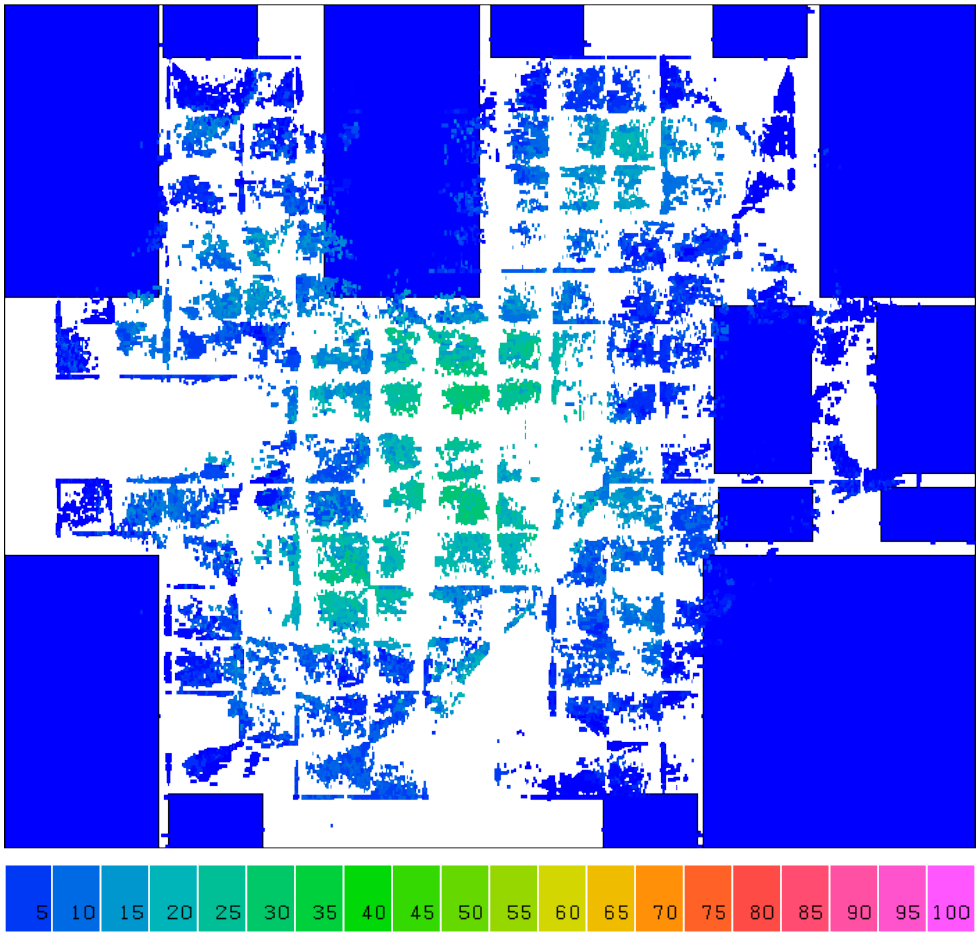
Example: Placement



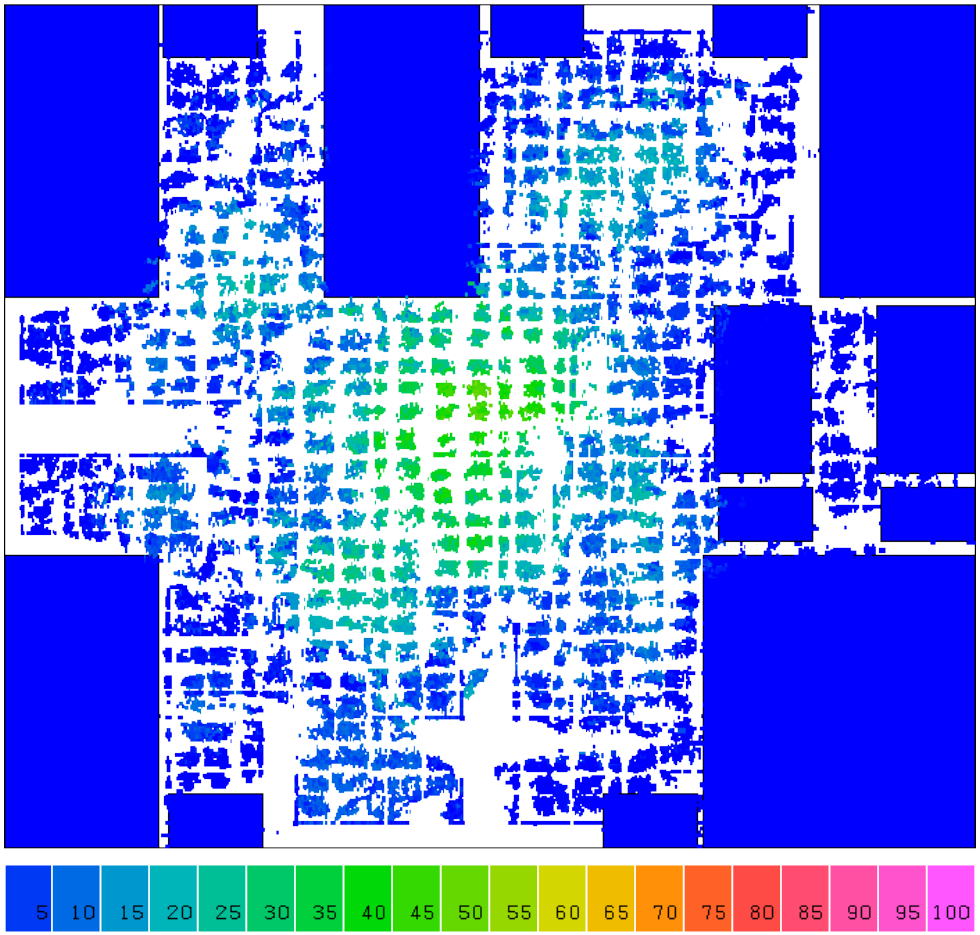
Example: Placement



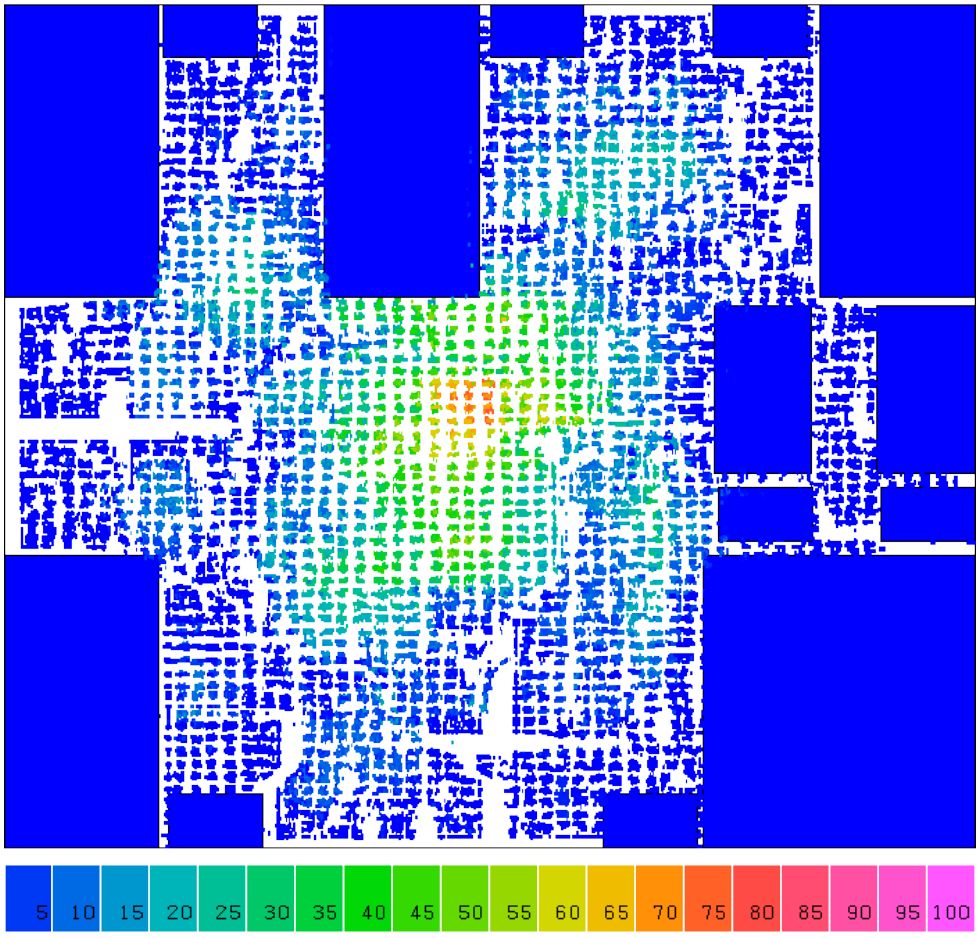
Example: Placement



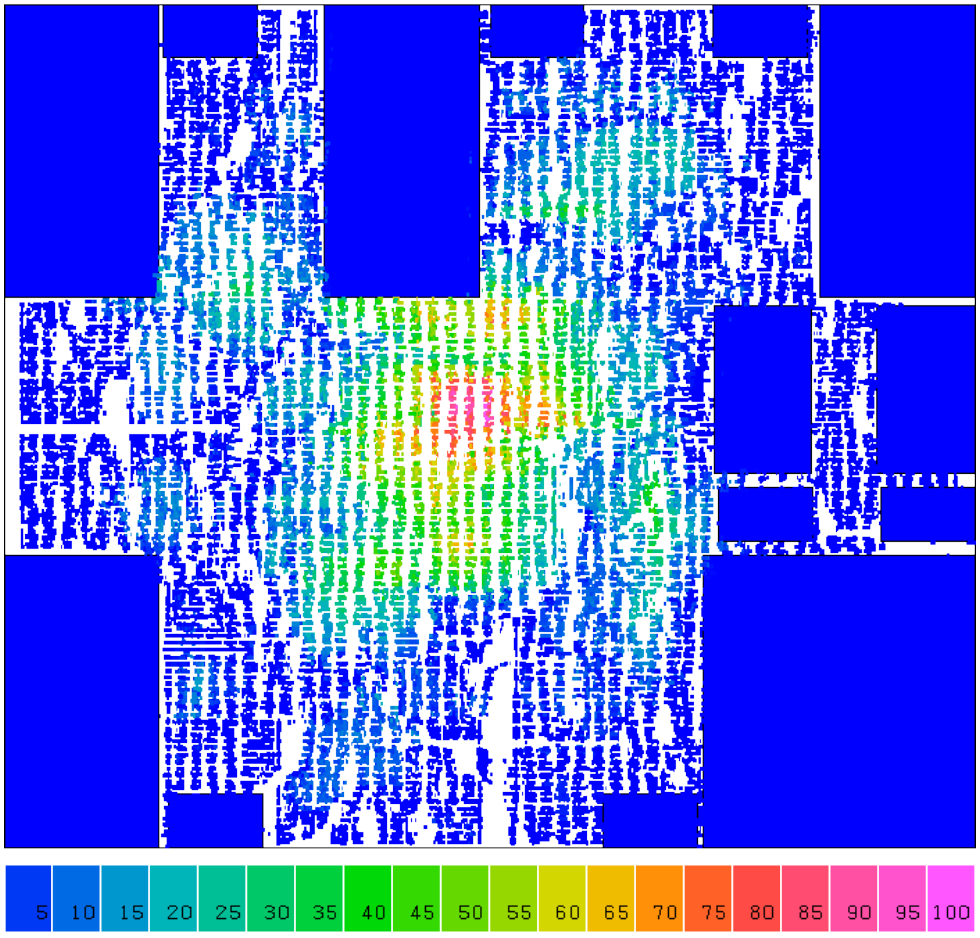
Example: Placement



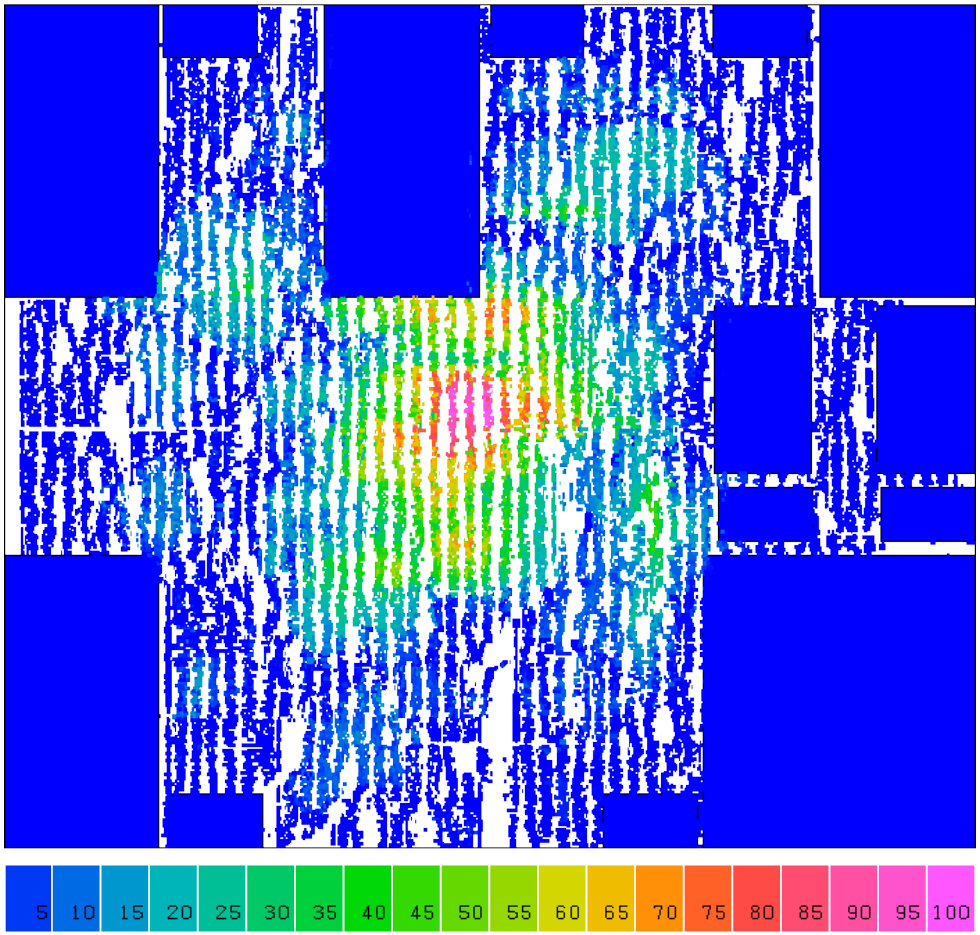
Example: Placement



Example: Placement

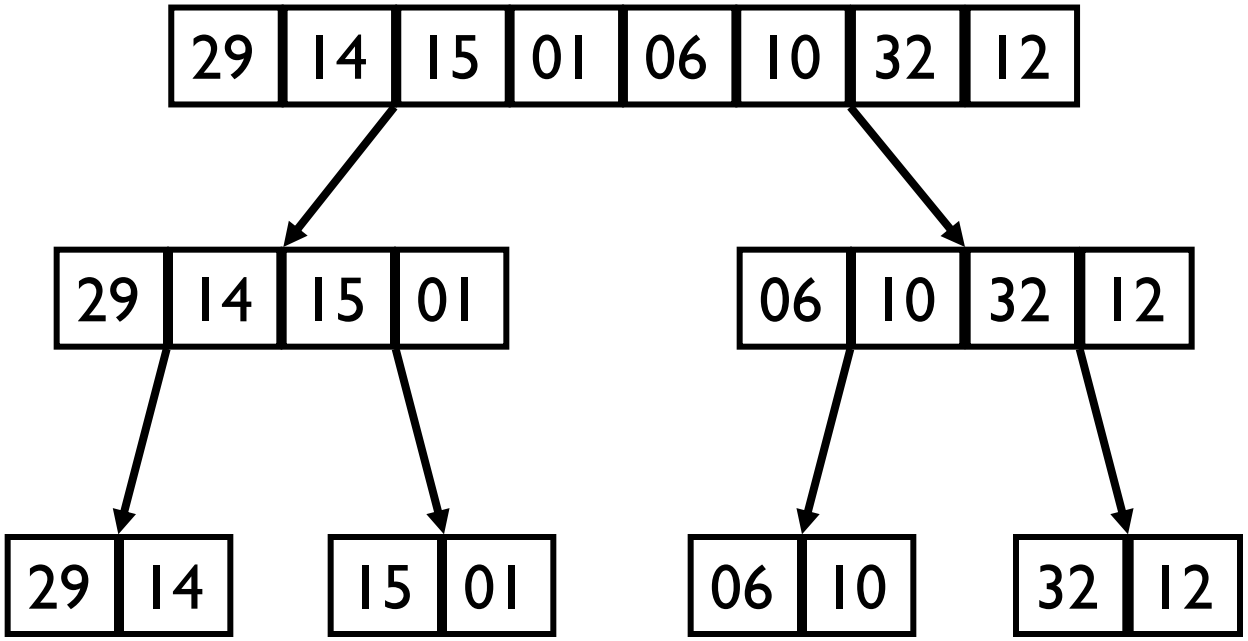


Example: Placement



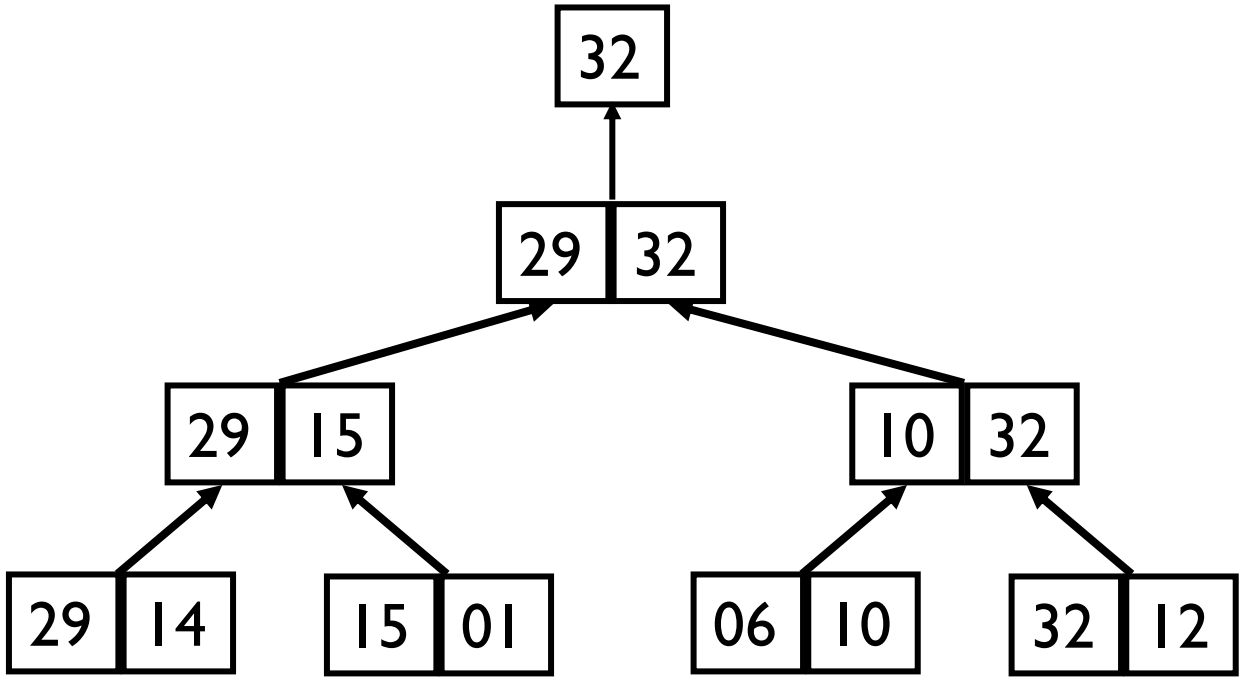
Example: Find the Minimum Value

- Divide



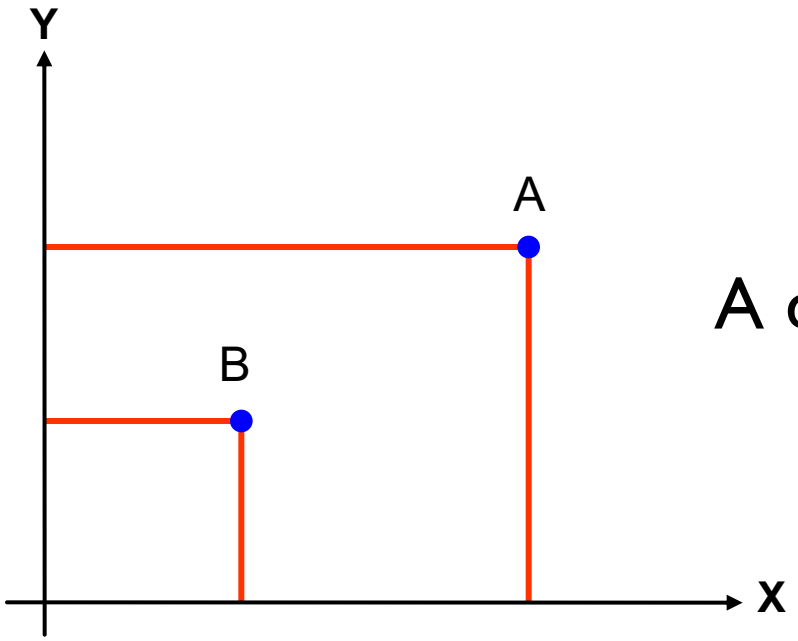
Example: Find the Minimum Value

- Conquer



Example: Find the 2D Maxima Points

A point (x1, y1) dominates (x2, y2) if x1 > x2 and y1 > y2.

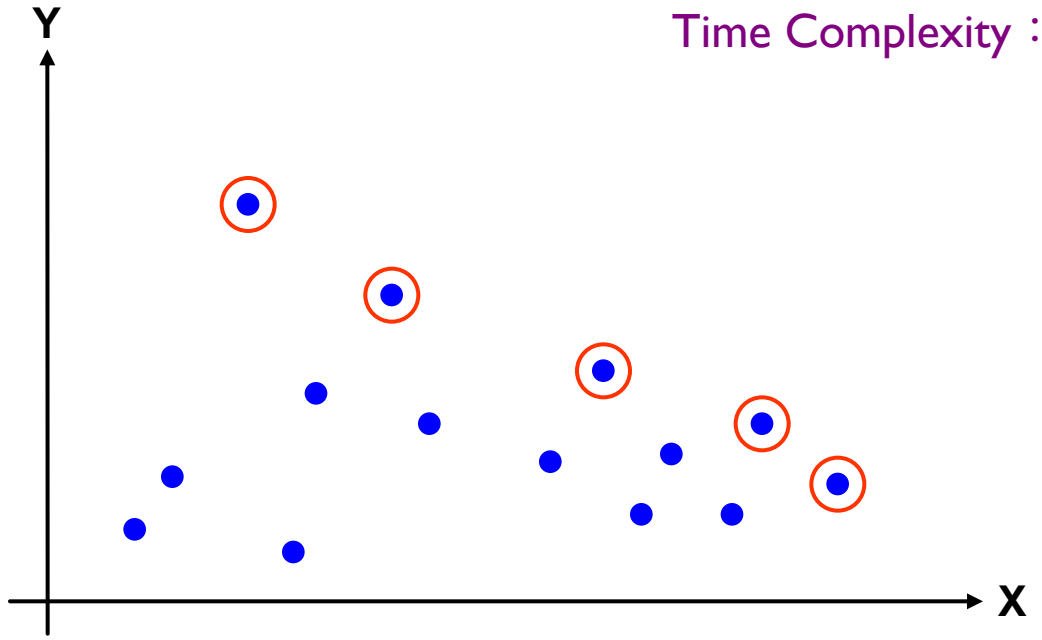


A dominates B

Example: Find the 2D Maximal Points

- Brute-force method
  - Write a two-level for-loop to compare all pair of points

Compare all pairs of points
Time Complexity : $O(n^2)$



Solve the 2D Maximal Points Problem

Input: A set S of n planar points.

Output: The maximal points of S.

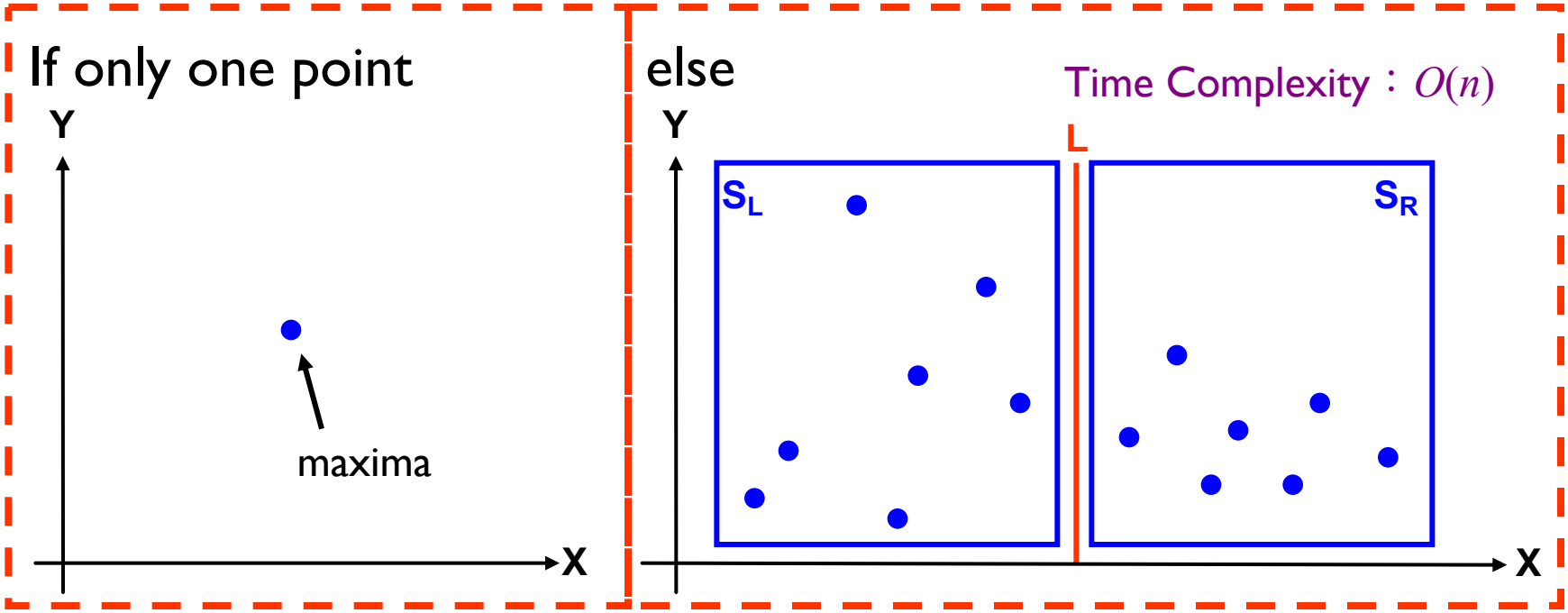
Step 1: If S contains only one point, return it as the maxima. Otherwise, find a line L perpendicular to the X-axis which separates S into S_L and S_R, with equal sizes.

Step 2: Recursively find the maximal points of S_L and S_R .

Step 3: Find the largest y-value in S_R, denoted as y_R. Discard each of the maximal points in S_L if its y-value is less than y_R.

Algorithm Illustration

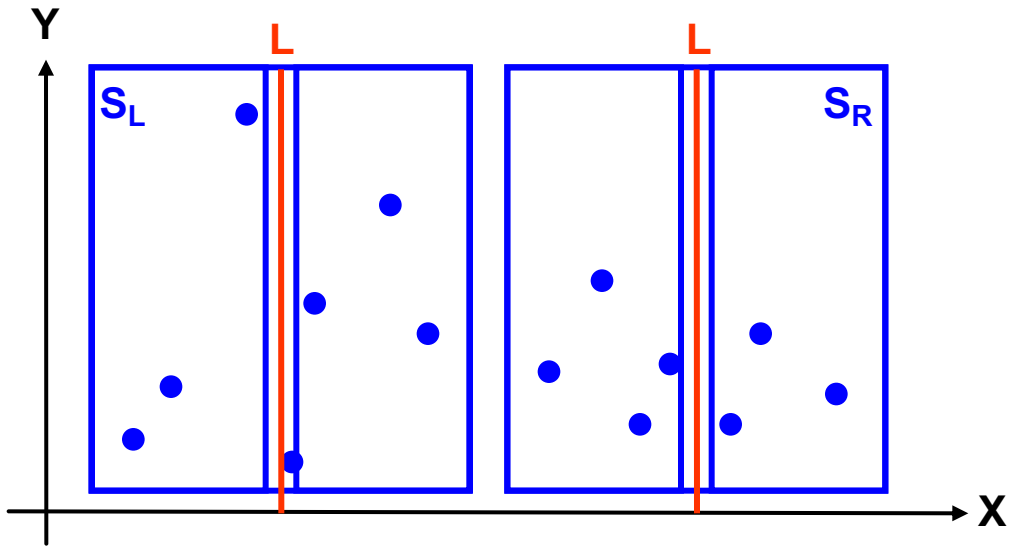
- Step 1



Algorithm Illustration

- Step 2

Time Complexity：$2T(n/2)$

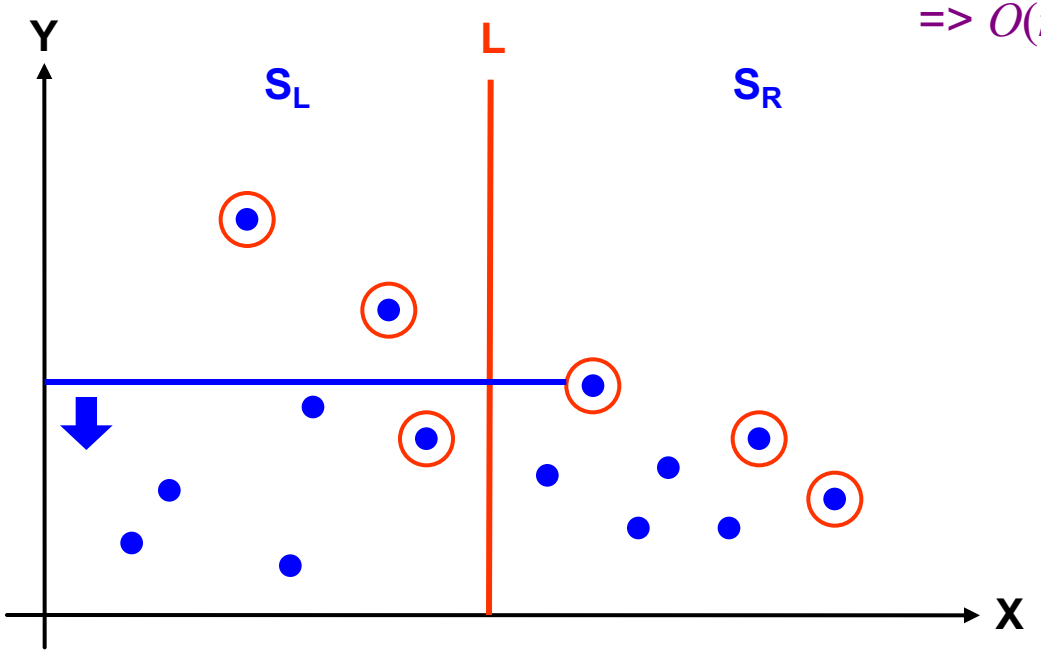


Algorithm Illustration

- Step 3

Time Complexity

=> $O(n\log n)$



Summary

- **Divide and Conquer is a recursive algorithm**
- **Three essential steps:**
  - **Divide**: If the input size is too large to deal with in a straightforward manner, divide the data into two or more disjoint subsets.
  - **Recur**: Use divide and conquer to solve the subproblems associated with the sub datasets.
  - **Conquer**: Take the solutions from the subproblems and “merge” them to a solution from the division point.

Note

- **To compile a “test.cpp” program to a binary “test”**
  - g++ test.cpp -O2 -o test
- **To feed a program with a test file from the standard output:**
  - ./simple < test.txt
- **To measure the runtime of the above program:**
  - time –p ./simple < test.txt