Overview

- Data structures and associated operations

Data structures	*Associated operations*
Linked list	insert, delete, makenull
Stacks	push, pop
Queues	remove from head, insert from tail
Trees	insert, delete, traverse
Graphs	traverse, shortest path, strong components, etc.

- Data structures and associated operations are "tools" for building programs

Overview (cont.)

- Algorithm design
  - A sequence of operations which are
    - clearly defined (no ambiguity as to what to do next)
    - effective (component operations done in finite time)
    - terminate
  - E.g. *Sorting*
    - Data structures: an array of length n
    - Algorithms: comparing & swapping elements (bubble sort, insertion sort, selection sort, quick sort, merge sort, etc.)
  - Programs = Algorithms + Data structures

Overview (cont.)

- General principles
  - Divide-and-conquer
  - Greedy
  - Dynamic programming
  - Backtracking
  - Branch-and-bound
  - Randomized algorithms

Caveats

- There are a lot more principles for algorithm designs that we do not cover
  - Numerical algorithms
  - Graph algorithms
  - Geometrical algorithms (e.g., vision, graphics)
  - Probabilistical algorithms
- Multi-stage, discrete, countably many, unique

Divide-and-Conquer

Divide-and-Conquer

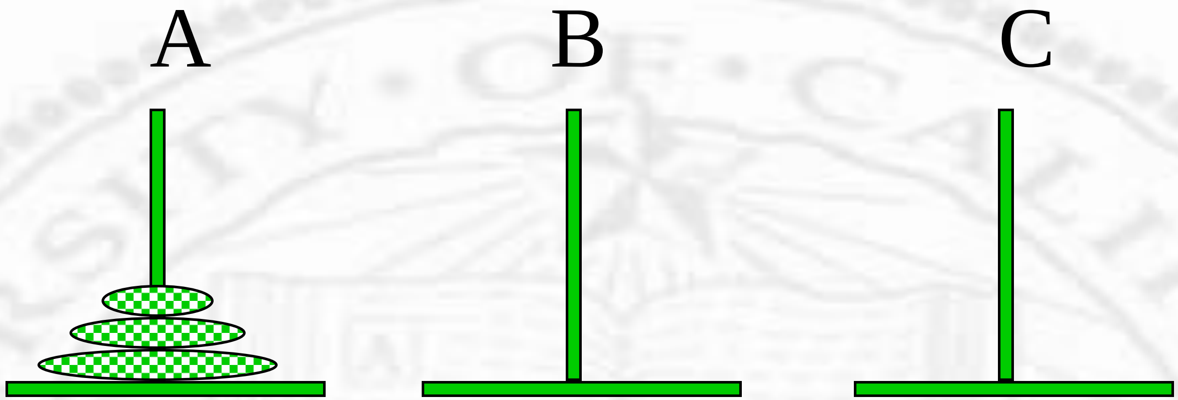
- Input *A*(*1:n*): *n* elements stored in an array

```
Procdure DandC(p,q)
  if Small(p,q) then
    return (G(p,q))
  else m ← Divide(p,q)
    return Combine(DandC(p,m), DandC(m+1,q))
  end if
end DandC
```

Divide-and-Conquer (cont.)

- *Divide*: split a larger problem into sub-problems of smaller size
- *Combine*: merge the solutions of sub-problems into that of a larger problem
- *Small*: is the problem small enough?
- *G(p,q)*: easy solutions to small problems

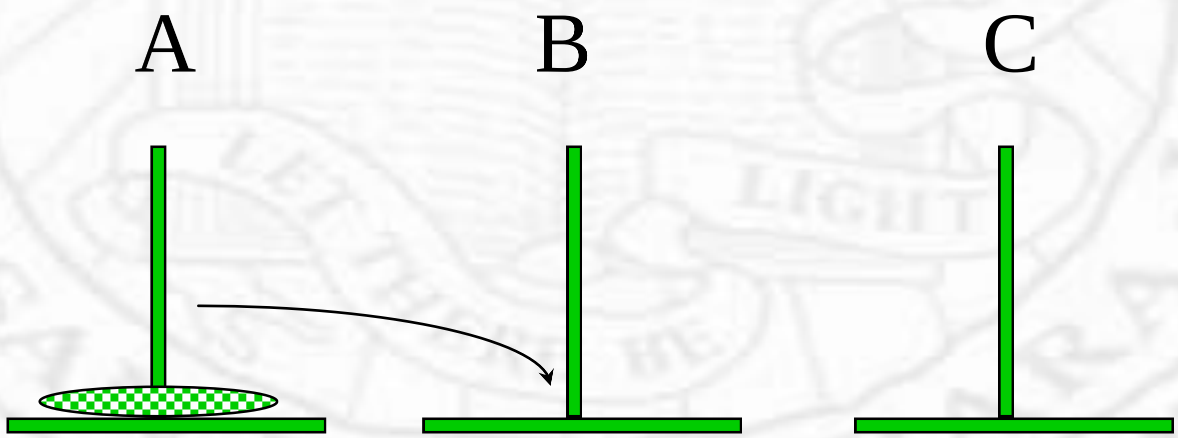
Hanoi Towers

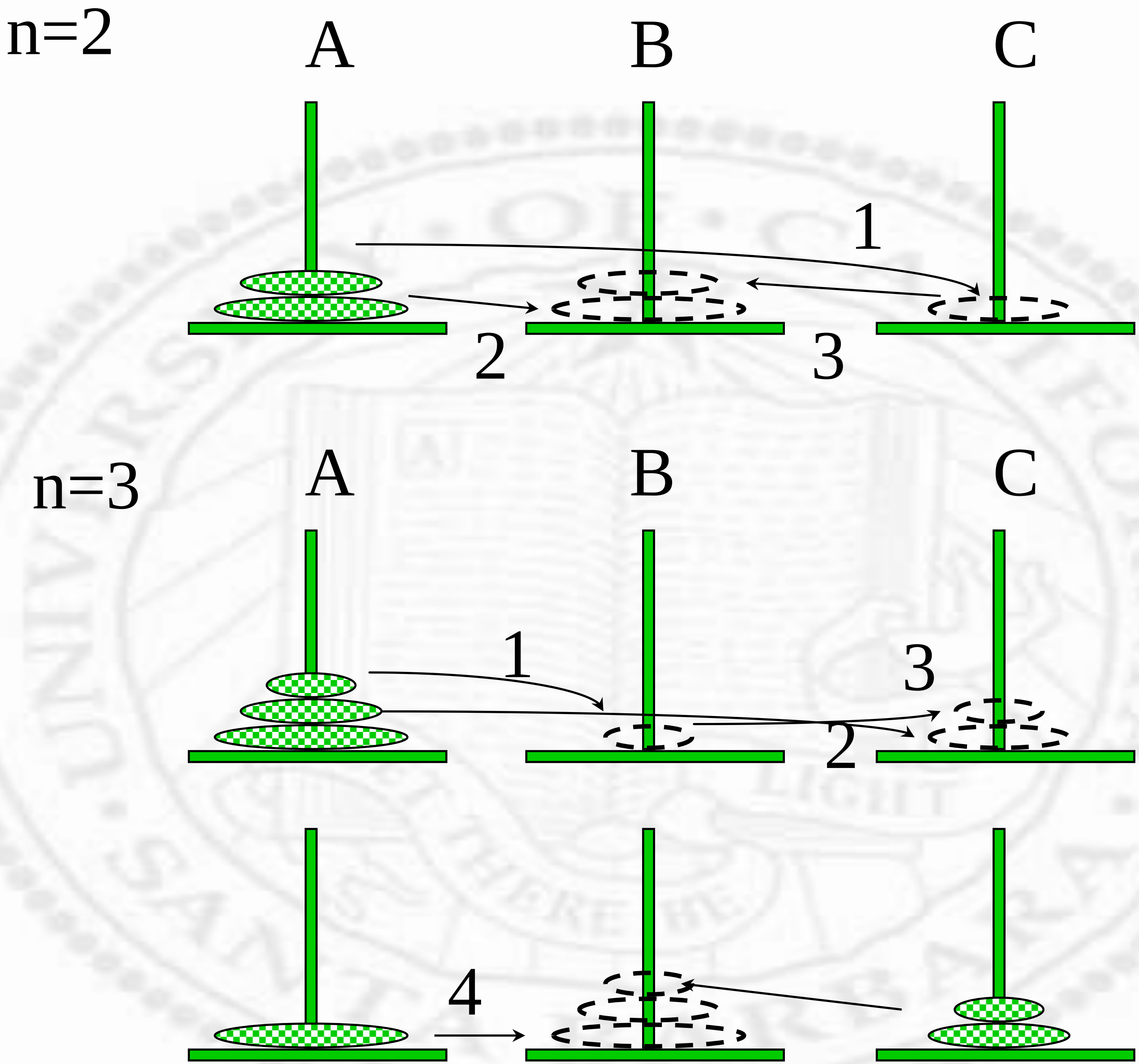


- Three pegs, A has *n* disks of different sizes stacked with smaller ones on top of bigger ones
- Move disks one at a time
- Never place a larger disk on top of a smaller one
- Move all disks onto B
- Trivial problem, the rule of the game dictates divide-and-conquer

- Hanoi(n, A, B, C)
  - n: number of disks
  - A: starting peg
  - B: end peg
  - C: temporary peg

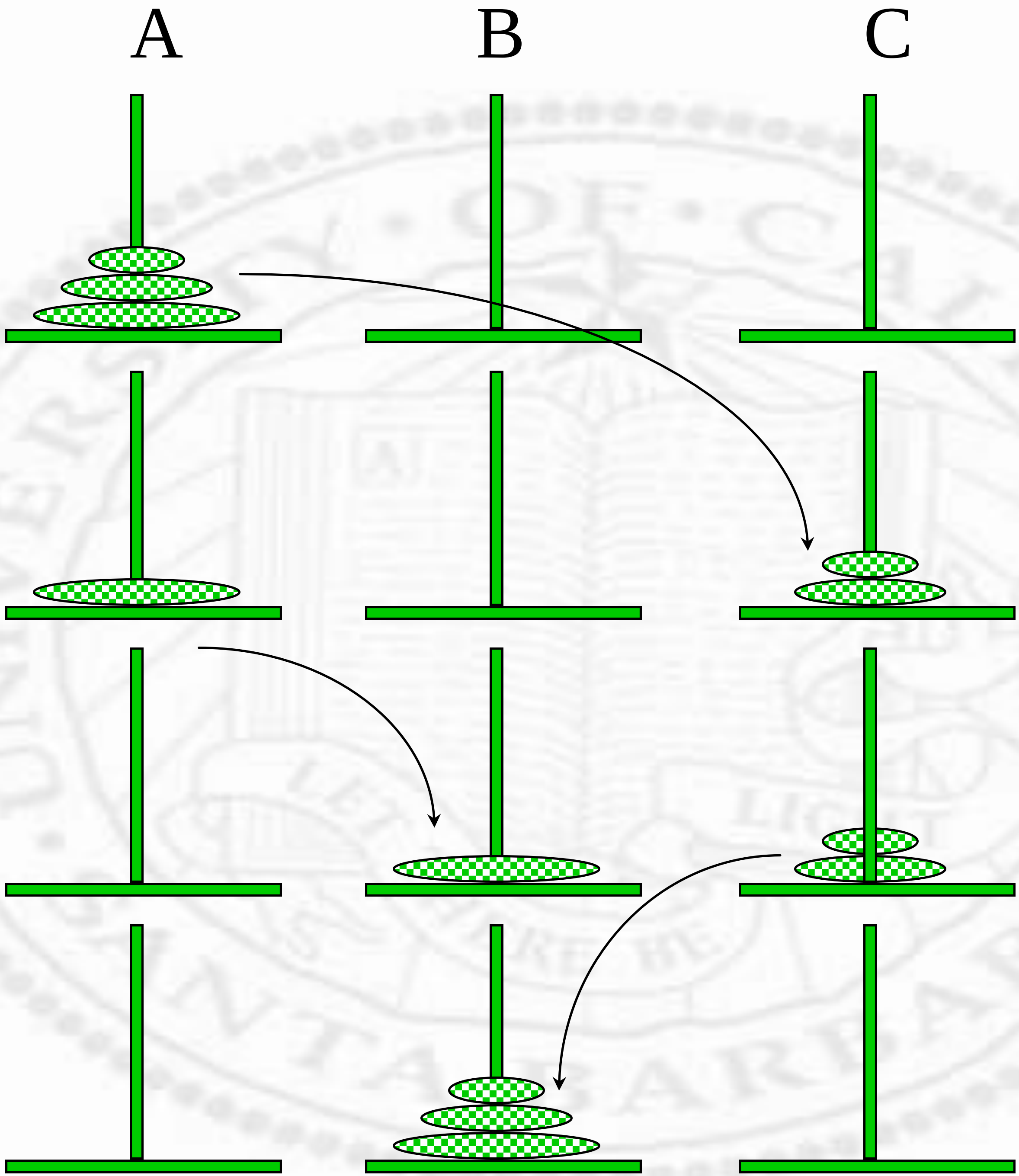
n=1

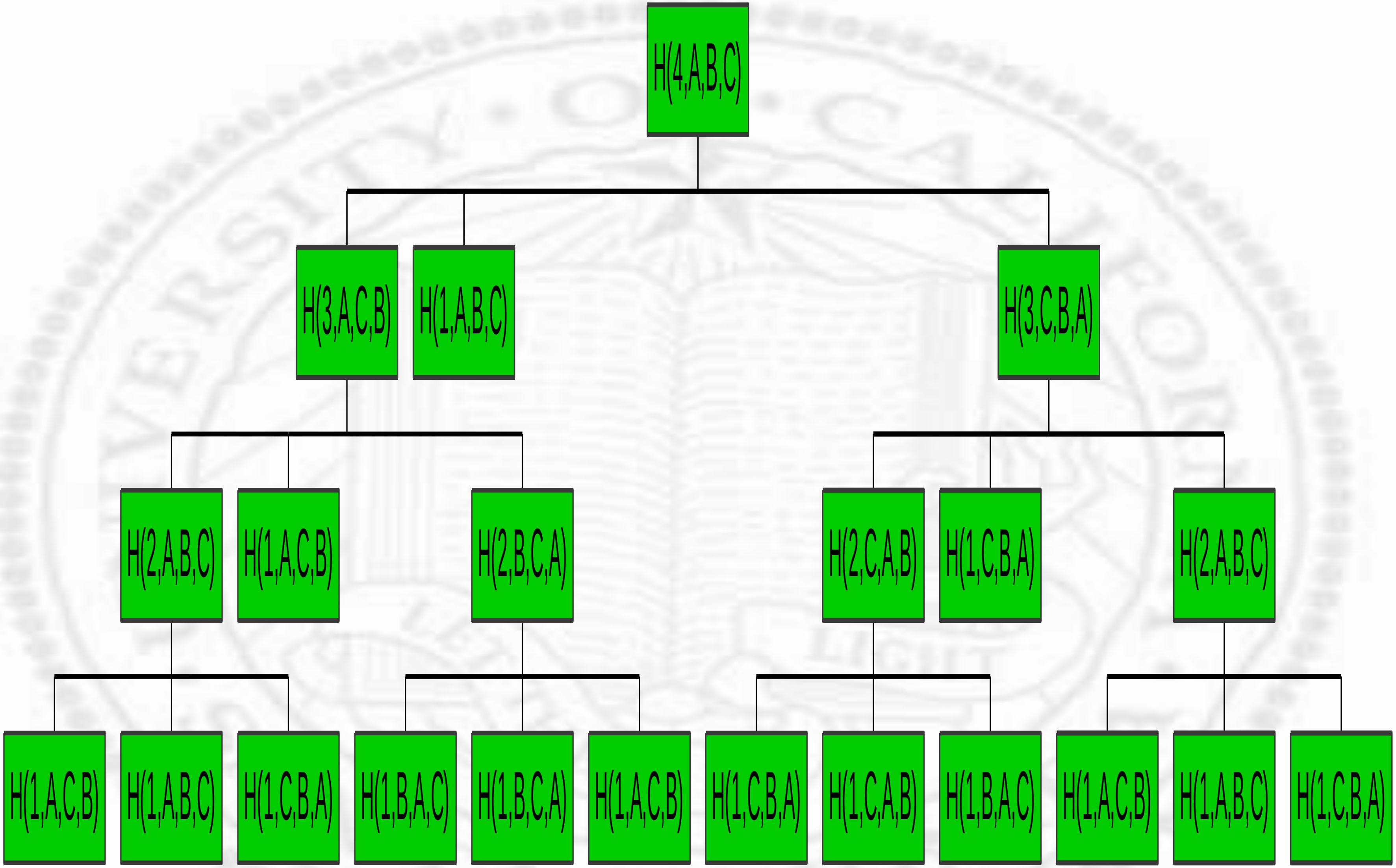



n=2
A
B
C
1
2
3
n=3
A
B
C
1
3
2
4

- Movement steps of m disks will be used in moving n disks ($n>m$)
- Problem is decomposable

$$
\begin{aligned}
&Hanoi(n,A,B,C) = \\
&\quad Hanoi(n-1,A,C,B) \\
&\quad + Hanoi(1,A,B,C) \\
&\quad + Hanoi(n-1,C,B,A)
\end{aligned}
$$


A
B
C


H(4,A,B,C)
H(3,A,C,B)
H(1,A,B,C)
H(3,C,B,A)
H(2,A,B,C)
H(1,A,C,B)
H(2,B,C,A)
H(2,C,A,B)
H(1,C,B,A)
H(2,A,B,C)
H(1,A,C,B)
H(1,A,B,C)
H(1,C,B,A)
H(1,B,A,C)
H(1,B,C,A)
H(1,A,C,B)
H(1,C,B,A)
H(1,C,A,B)
H(1,B,A,C)
H(1,A,C,B)
H(1,A,B,C)
H(1,C,B,A)

- *Divide*: two sub-problems of size *n-1* and one sub-problem of size *1*
- *Small(p,q)*: when the problem size is *1*
- *G(p,q)*: move a disk from peg to peg
- *Combine*: sequential concatenation of moves

- Time complexity

$$
\begin{aligned}
T(n) &= T(n-1)+c+T(n-1) && = 2T(n-1)+c \\
&= 2\{2T(n-2)+c\}+c && = 2^2T(n-2)+(1+2)c \\
&= 2^2\{2T(n-3)+c\}+(1+2)c && = 2^3T(n-3)+(1+2+2^2)c \\
&\ldots && \ldots \\
&= 2^{n-1}T(1)+(1+2+\ldots+2^{n-2})c \\
&= (1+2+\ldots+2^{n-2}+2^{n-1})c \\
&= O(2^n)
\end{aligned}
$$

Binary Search

- Input:
  - a list of elements sorted in *nondecreasing* order
  - an element x
- Output
  - determine whether x is present
  - if so, the position index j

- *Divide:*

$$BS(n, a_1, a_2, \ldots, a_n, x) =$$

$$BS\left(\left\lfloor \frac{n+1}{2} \right\rfloor - 1, a_1, a_2, \ldots, a_{\left\lfloor \frac{n+1}{2} \right\rfloor - 1}, x\right) +$$

$$BS\left(1, a_{\left\lfloor \frac{n+1}{2} \right\rfloor}, x\right) +$$

$$BS\left(n - \left\lfloor \frac{n+1}{2} \right\rfloor, a_{\left\lfloor \frac{n+1}{2} \right\rfloor + 1}, \ldots, a_n, x\right)$$

– two problems of size approximately *n/2*, and one problem of size *1*

- *Small(p,q): when* the size of problem is 1
- *G(p,q):* compare the single element in the list with the search element
- *Combine:*

$$x = a_{\left\lfloor \frac{n+1}{2} \right\rfloor} \qquad j = \left\lfloor \frac{n+1}{2} \right\rfloor$$

$$x < a_{\left\lfloor \frac{n+1}{2} \right\rfloor} \qquad \text{solve the first sub - problem}$$

$$x > a_{\left\lfloor \frac{n+1}{2} \right\rfloor} \qquad \text{solve the third sub - problem}$$

x=101

1	2	3	4	5	6	7	8	9
-15	-6	0	7	9	23	54	82	101
low				mid				high
				$mid+1 \rightarrow$	low	mid		high
						$mid+1 \rightarrow$	low/ med	high
							$mid+1 \rightarrow$	low, mid, high

$$mid = \left\lfloor \frac{low+high}{2} \right\rfloor$$

found j=9

x=30

1	2	3	4	5	6	7	8	9
-15	-6	0	7	9	23	54	82	101

low mid high

$mid + 1 \rightarrow$ low mid high

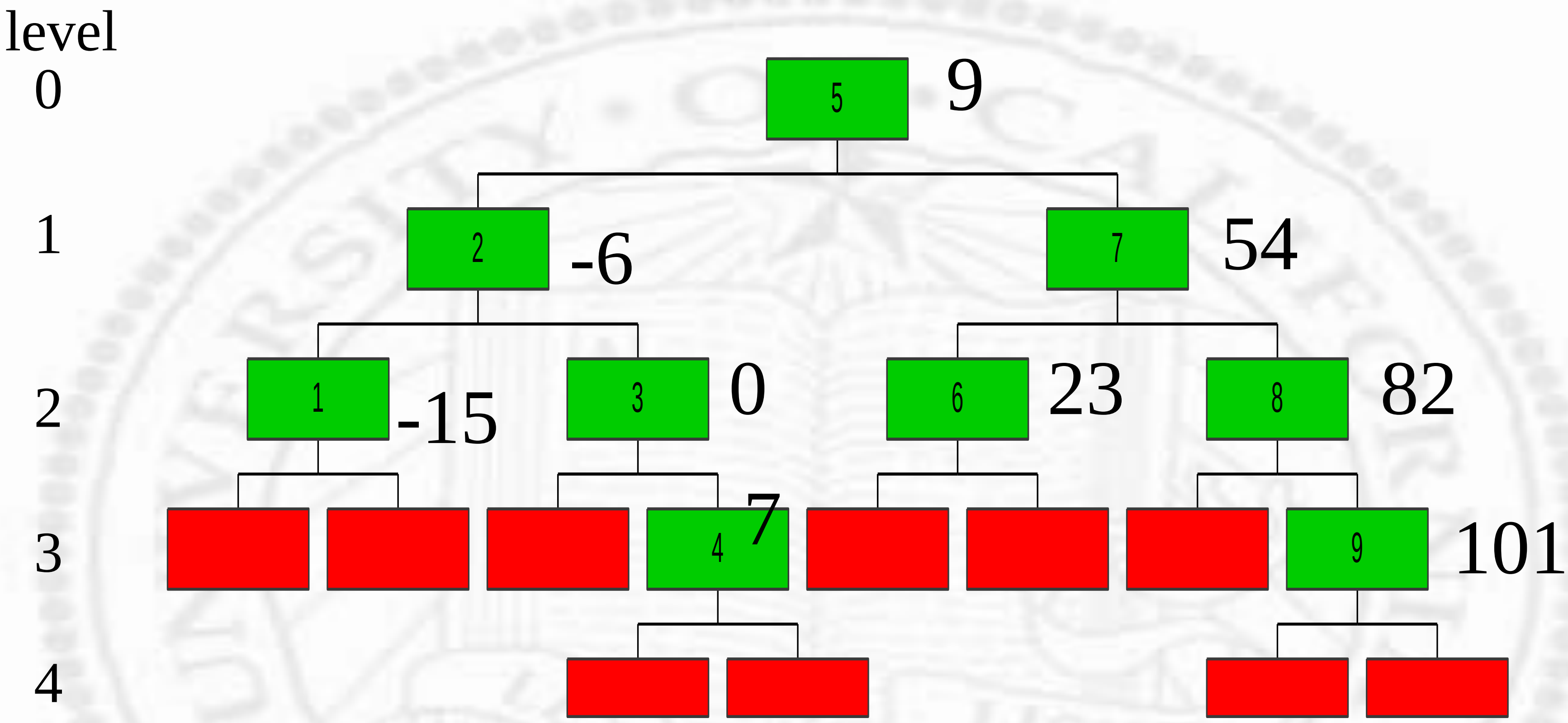
$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

low,
mid,
high $\leftarrow mid - 1$

high

$mid + 1 \rightarrow$ low>high, not found

- Time complexity



internal nodes: successful search

external nodes: failed search

- Properties of binary search trees
  - balanced (root corresponds to the middle element, two subtrees are of approximately equal size)
  - with *n* elements $2^{k-1} \le n < 2^k$

Tree depth (k)	Min capacity (2^{k-1})	Max capacity (2^k)
1	1	2
2	2	4
3	4	8
4	8	16

    - internal nodes at levels of 0 to *k-1*

(successful searches make at most *k* comparisons)

    - external nodes at levels *k-1* and *k*

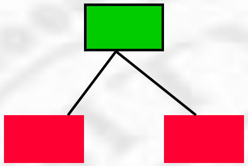
(failed searches make at least *k-1* ad at most *k* comparisons)

  - worst case is *O(k)* or *O(logn)* for both successful and failed searches
  - best case is *O(1)* for successful and *O(logn)* for failed searched

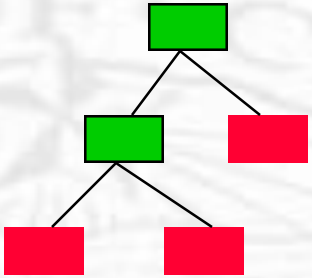
- Average case - slightly more complicated

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{Average performance} =} \\
\text{prob(S)} \times \text{average performance} & \text{prob(F)} \times \text{average performance} \\
\text{of successful searches} + & \text{of failed searches} \\
\Downarrow & \Downarrow \\
\text{average \# of comparisons} & \text{average \# of comparisons} \\
\text{in successful searches} & \text{in failed searches} \\
\Downarrow & \Downarrow \\
\text{average internal path length} + 1 & \text{average external path length} \\
\Downarrow & \Downarrow \\
\dfrac{\text{total internal path length (I)}}{\text{\# of internal nodes (i)}} + 1 & \dfrac{\text{total external path length (E)}}{\text{\# of external nodes (e)}}
\end{array}
$$

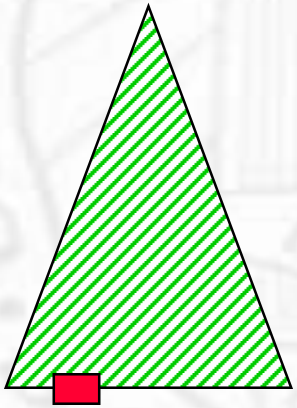
❖ # of internal (i) and external (e) nodes



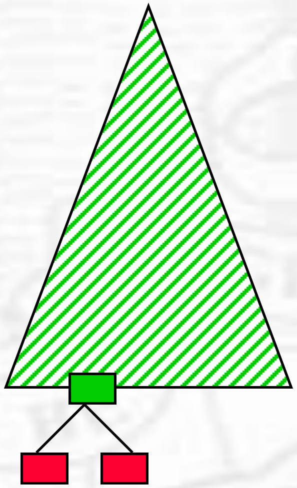
(i=1, e=2)



(i=2, e=3)

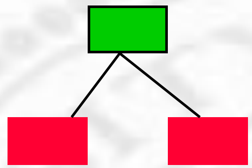


(i=n, e=n+1)

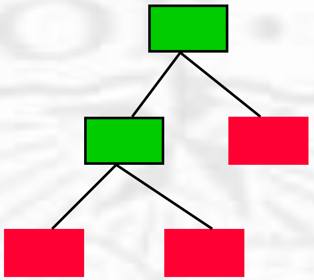


(i=n+1, e=n+2)

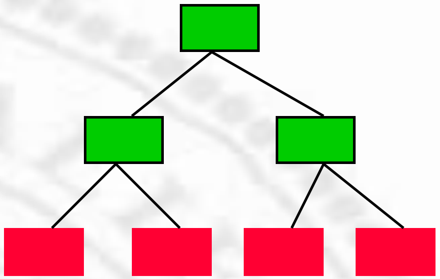
- total internal (I) and external (E) path length



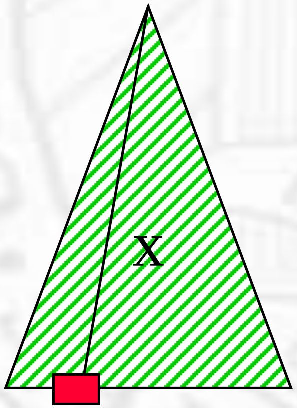
(I=0, E=2)



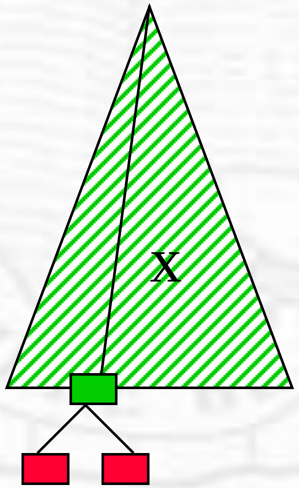
(I=1, E=5)



(I=2, E=8)



$$E = I + 2i$$



$$\begin{aligned} I' &= I + x \\ E' &= E + 2(x+1) - x \\ &= (I + 2i) + 2(x+1) - x \\ &= (I + x) + 2(i+1) \\ &= I' + 2i' \end{aligned}$$

$$If\ \ i\ is\ n$$

$$\Downarrow$$

$$\Downarrow$$

$$I = E - 2i \approx (n+1)\log n - 2n$$

$$\Downarrow$$

$$\frac{I}{n} + 1 \approx \frac{(n+1)\log n - 2n}{n} + 1$$

$$\approx \log n - 1 = O(\log n)$$

$$\Rightarrow$$

$$Then\ e\ is\ n{+}1$$

$$\Downarrow$$

$$E \approx (n+1)\log n = O(n\log n)$$

$$\Downarrow$$

$$\Downarrow$$

$$\frac{E}{n+1} \approx \frac{(n+1)\log n}{n+1} = O(\log n)$$

average performanc e is

O(logn) regardless

More Examples: Sorting

- brute force methods
  - bubble sort
  - selection sort
  - insertion sort

  $O(n^2)$

- “smart” methods
  - quick sort
  - merge sort

  $O(n \log n)$

  - based on Divide-and-Conquer

Bubble sort

```
for i=1 to n-1 do
    for  j=n downto i+1 do
        if a[j]<a[j-1] then
            swap(a[j-1], a[j])
```

$O(1)$

$O(n-i)$

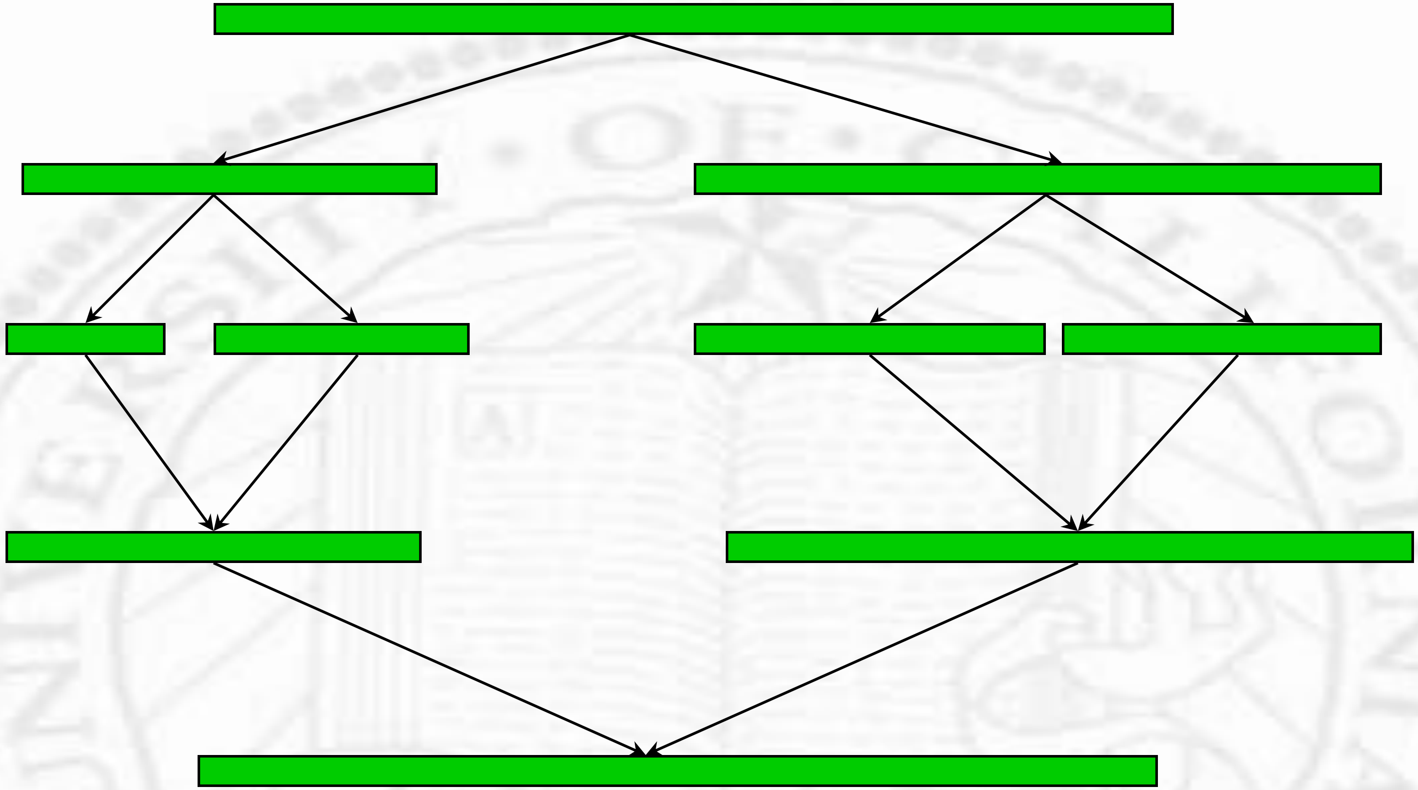
$O(\sum_{i=1}^{n-1} n-i) =$

$O(n^2)$

$i =$	1	2	3	4	5	6	7	8	9
1	65	85	70	75	80	60	55	50	45
2	45	65	85	70	75	80	60	55	50
3	45	50	65	85	70	75	80	60	55
4	45	50	55	65	85	70	75	80	60
5	45	50	55	60	65	85	70	75	80
6	45	50	55	60	65	70	85	75	80
7	45	50	55	60	65	70	75	85	80
8	45	50	55	60	65	70	75	80	85
9	45	50	55	60	65	70	75	80	85

- Each iteration places one element correctly
- Many elements are involved in many iterations
- Size of subproblems decrease very slowly through iterations

❖ Sorting based on Divide-and-Conquer



- Quick sort
  - uneven division
  - simple concatenation
- Merge sort
  - even division
  - elaborate merge

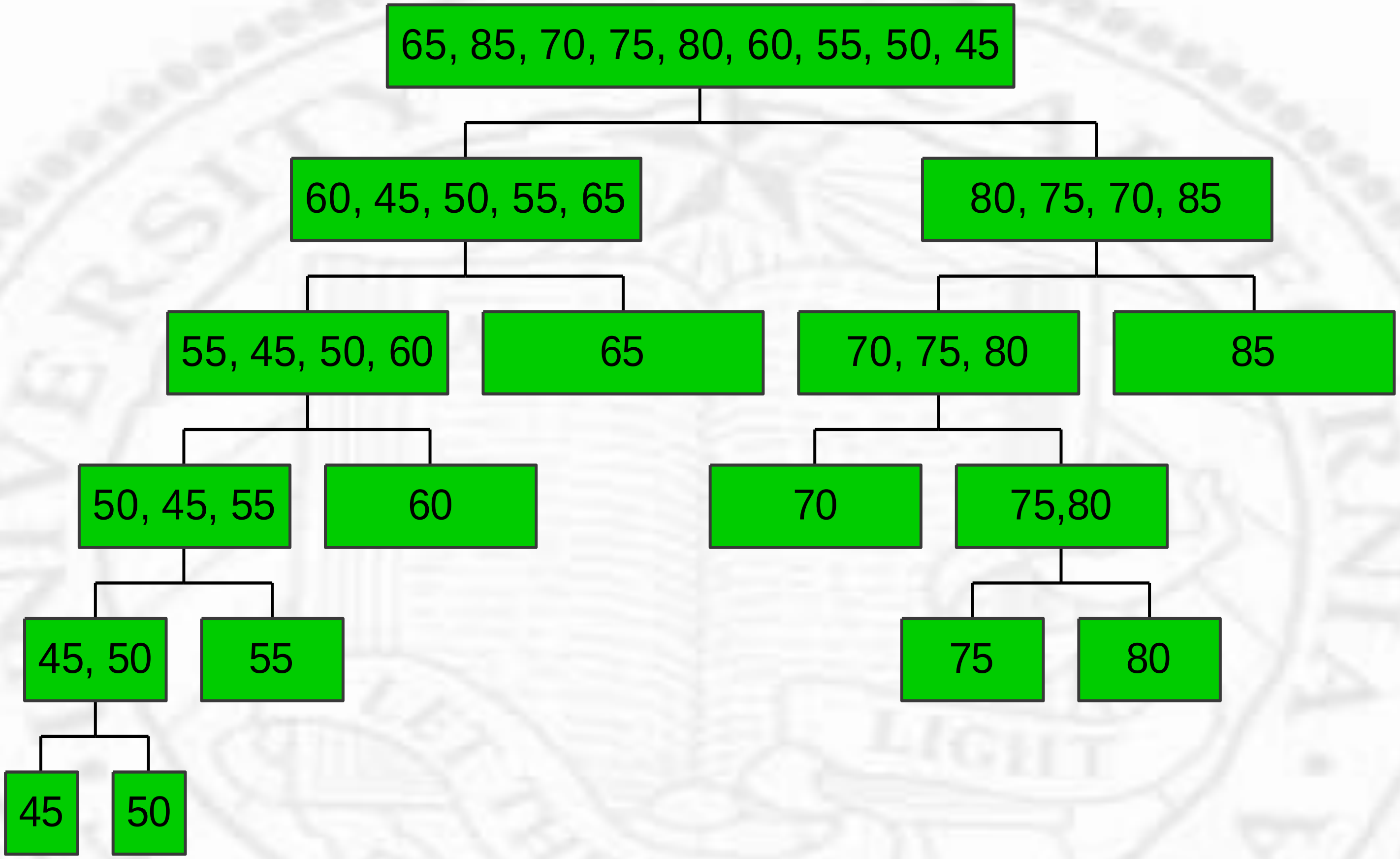
Quick Sort

- Input: a list of n elements
- Output: a list of the same elements sorted in nondecreasing order

❖ *Divide*

$$QS(n, a_1, a_2, \ldots, a_n) =$$
$$partition(1, n) +$$
$$QS(i, a_1, a_2, \ldots, a_i) +$$
$$QS(n - i, a_{i+1}, \ldots, a_n)$$

- *Small(p,q):* when the problem size becomes *1*
- *G(p,q):* nothing
- *Combine:* simple concatenation of solutions of two sorted lists

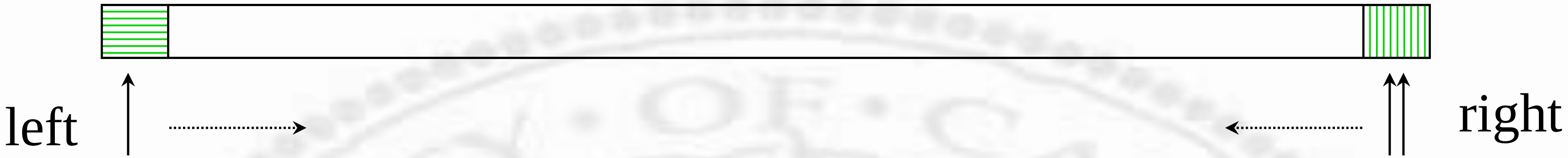

65, 85, 70, 75, 80, 60, 55, 50, 45
60, 45, 50, 55, 65
80, 75, 70, 85
55, 45, 50, 60
65
70, 75, 80
85
50, 45, 55
60
70
75,80
45, 50
55
75
80
45
50

$$partition\,(p,q) \rightarrow i, such\ that\ a_i\ is\ the\ pivot$$

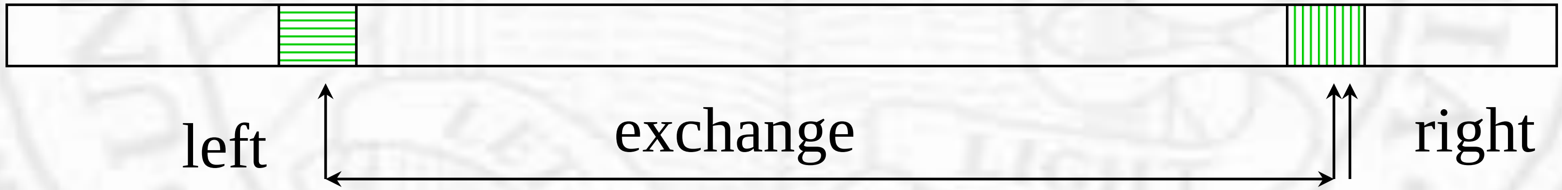
$$a_p, a_{p+1}, \ldots, a_i \leq a_i$$

$$a_{i+1}, a_{i+2}, \ldots, a_q > a_i$$

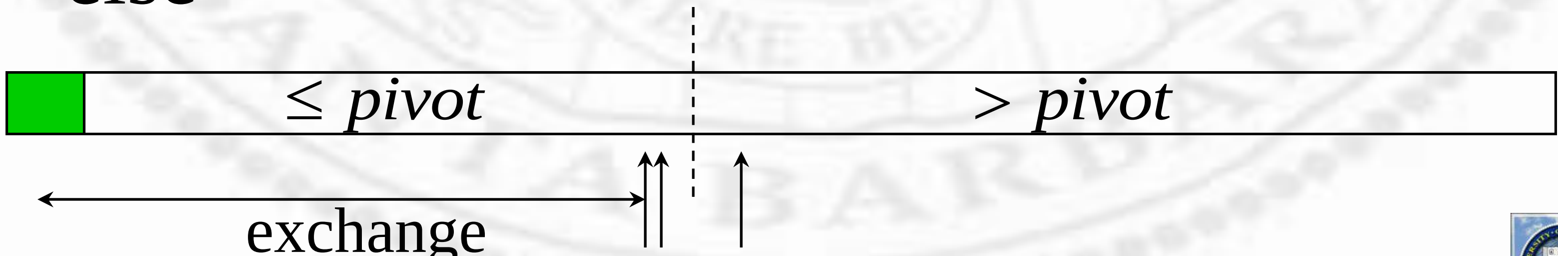
65	85	70	75	80	60	55	50	45
65	45	70	75	80	60	55	50	85
65	45	50	75	80	60	55	70	85
65	45	50	55	80	60	75	70	85
65	45	50	55	60	80	75	70	85
60	45	50	55	65	80	75	70	85



- left pointer moves right, until $*(left) > pivot$
- right point moves left, until $*(right) \le pivot$
- if *left<right*, swap **(left)* and **(right)*

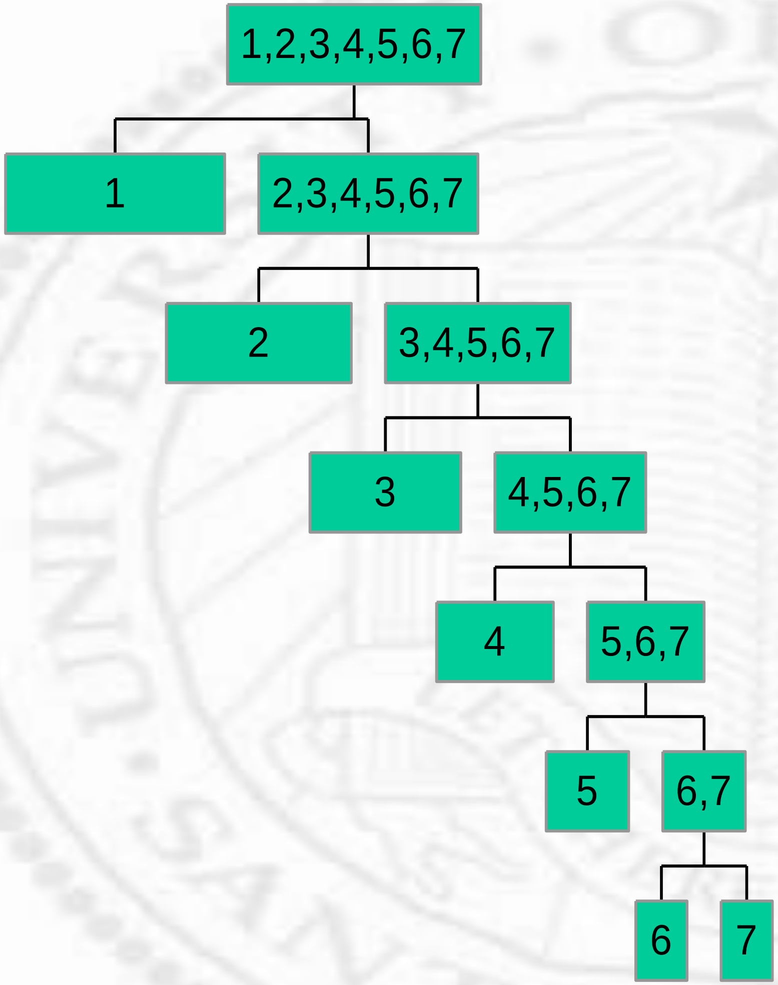


- else



- Array is scanned only once, at a particular location
  - no action is taken (advance pointer), or
  - swap elements and advance pointer
  - partition is *O(array length)*

❖ Time complexity - worst case



$$\sum_{1}^{n-1}(\text{\# of elements in the array})$$

$$= n + (n-1) + (n-2) + \ldots + 2$$

$$= O(n^2)$$

- Time complexity - average case
  - Assumptions:
    - the n elements are distinct
    - the pivot element can be equally likely the *ith* element in the sorted array

$$T(n) = \frac{1}{n}\sum_{i=1}^{n}\{T(i) + T(n-i)\} + cn$$

$$= O(n \log n)$$

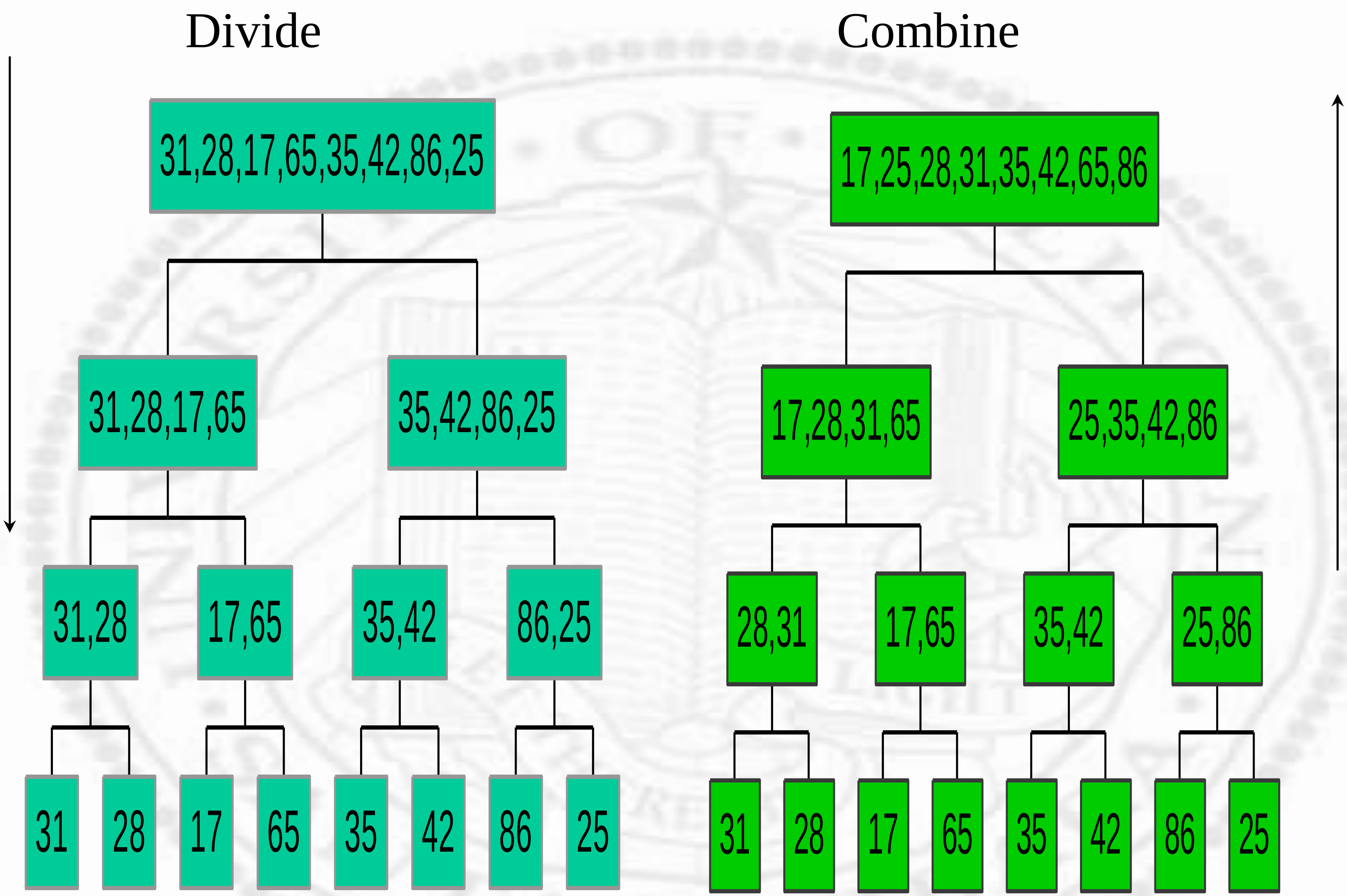
Merge Sort

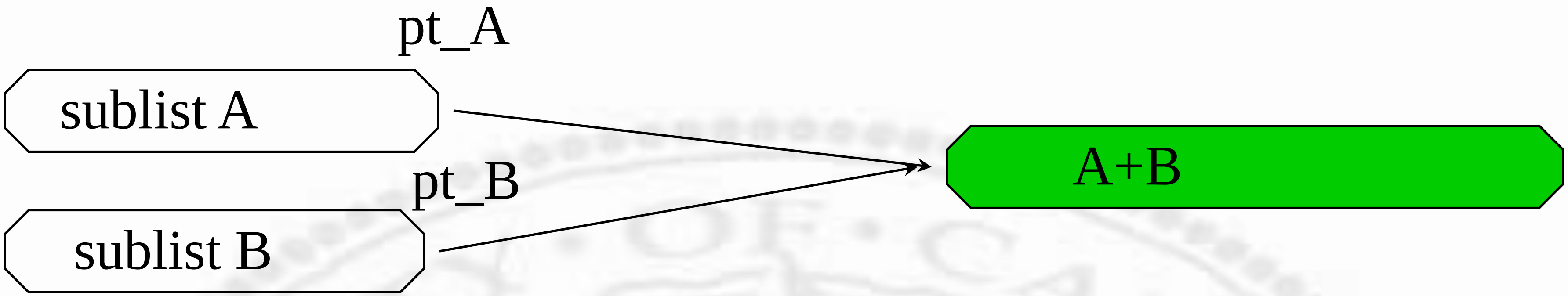
- Input: a list of n elements
- Output: a list of the same elements sorted in nondecreasing order

❖ *Divide*

$$MS(n, a_1, a_2, \ldots, a_n) =$$
$$MS\left(\left\lfloor \frac{n+1}{2} \right\rfloor, a_1, a_2, \ldots, a_{\left\lfloor \frac{n+1}{2} \right\rfloor}\right) +$$
$$MS\left(n - \left\lfloor \frac{n+1}{2} \right\rfloor, a_{\left\lfloor \frac{n+1}{2} \right\rfloor + 1}, \ldots, a_n\right) +$$
$$\text{merge the two sublists properly}$$

- *Small(p,q):* when the problem size becomes *1*
- *G(p,q):* nothing
- *Combine:* trace down the two sublists and merge them properly


Divide
Combine
31,28,17,65,35,42,86,25
31,28,17,65
35,42,86,25
31,28
17,65
35,42
86,25
31
28
17
65
35
42
86
25
17,25,28,31,35,42,65,86
17,28,31,65
25,35,42,86
28,31
17,65
35,42
25,86
31
28
17
65
35
42
86
25



if $*(pt\_A)$ is $NULL$, append B to $A+B$

else if $*(pt\_B)$ is $NULL$, append A to $A+B$

else if $*(pt\_A) < *(pt\_B)$,

append $*(pt\_A)$ to $A+B$, increment pt_A

else

append $*(pt\_B)$ to $A+B$, increment pt_B

end if

$O(|A|+|B|)$ operations

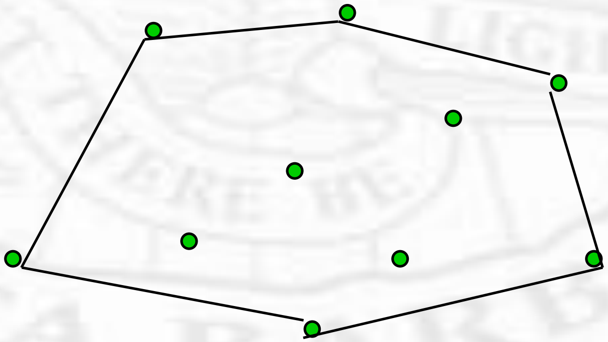
❖ Time complexity

$$MS(n, a_1, a_2, \ldots, a_n)$$
$$= MS\left(\left\lfloor \frac{n+1}{2} \right\rfloor, a_1, a_2, \ldots, a_{\left\lfloor \frac{n+1}{2} \right\rfloor}\right)$$
$$+ MS\left(n - \left\lfloor \frac{n+1}{2} \right\rfloor, a_{\left\lfloor \frac{n+1}{2} \right\rfloor + 1}, \ldots, a_n\right)$$
$$+ \text{merge}$$

$$\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + cn = 2T\left(\frac{n}{2}\right) + cn \\
&= 2\left(2T\left(\frac{n}{4}\right) + c\frac{n}{2}\right) + cn = 2^2 T\left(\frac{n}{4}\right) + 2cn \\
&= 2^2\left(2T\left(\frac{n}{8}\right) + c\frac{n}{4}\right) + 2cn = 2^3 T\left(\frac{n}{8}\right) + 3cn \\
&\ldots \\
&= 2^k T(1) + kcn \qquad n = 2^k \\
&= an + cn\log n \\
&= O(n\log n)
\end{aligned}$$

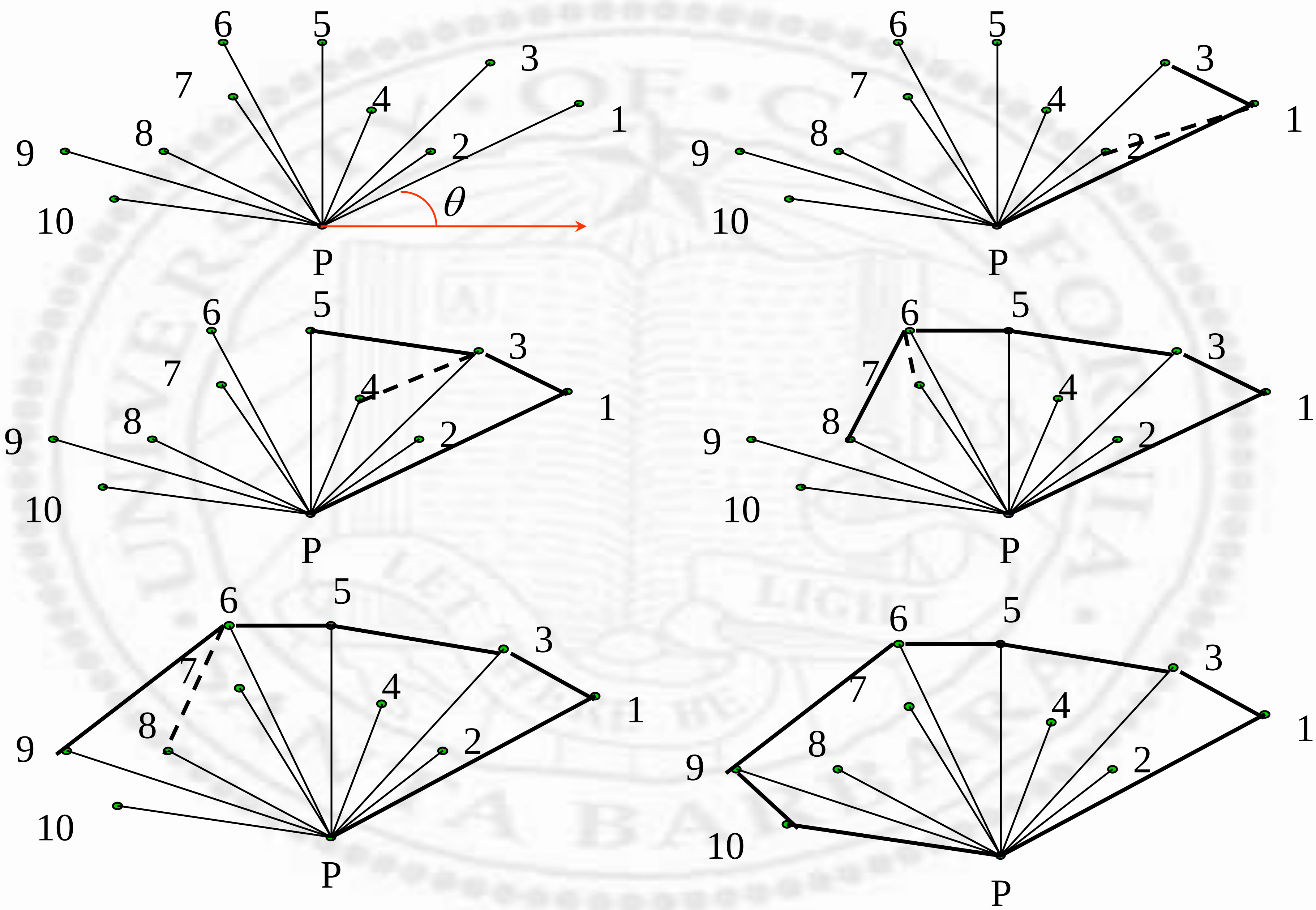
Convex Hull

- Input: a collection of *n* points
- Output: the smallest convex polygon that encloses the set of points
  - 2D case: points as nails sticking out on a table, put a rubberband around them



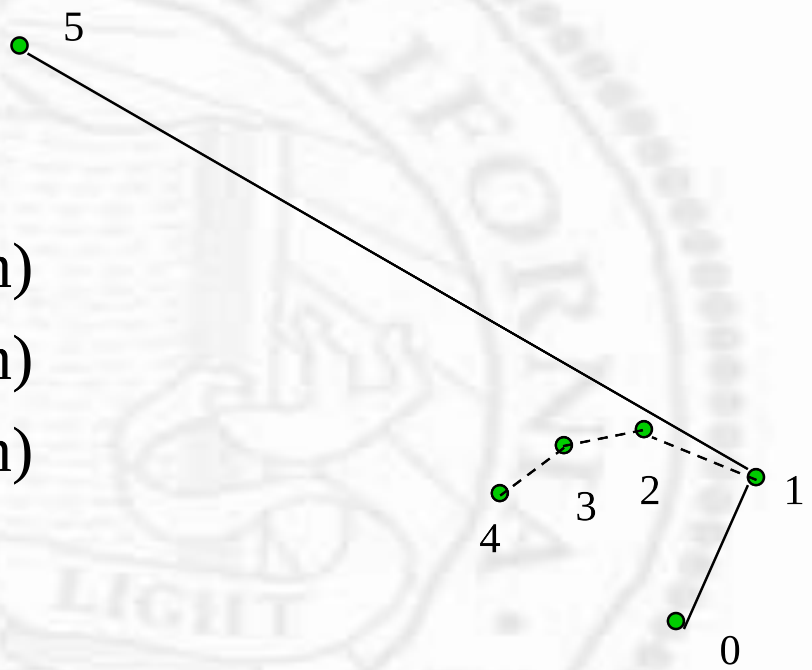
- Properties
  - Use given points as vertices
  - Contain all extreme points in the set
  - Points of smallest and largest x and y coordinates are included
  - Traverse the edge of the hull
    - counterclockwise, all points must be on the left
    - clockwise, all points must be on the right

❖ Graham's Scan (package wrapping)



- Start at some point that guaranteed to be on the convex hull (e.g., point with smallest y coordinate)
- From that point, compute theta (see previous slide) for all remaining points
- Sort by theta and consider each point in turn
- After examining *i-1* points
  - p[1..M] are on the convex hull
- After examining *i* points
  - p[M] is *recursively* eliminated if p[M], p[M-1] and P[i] make the wrong turn

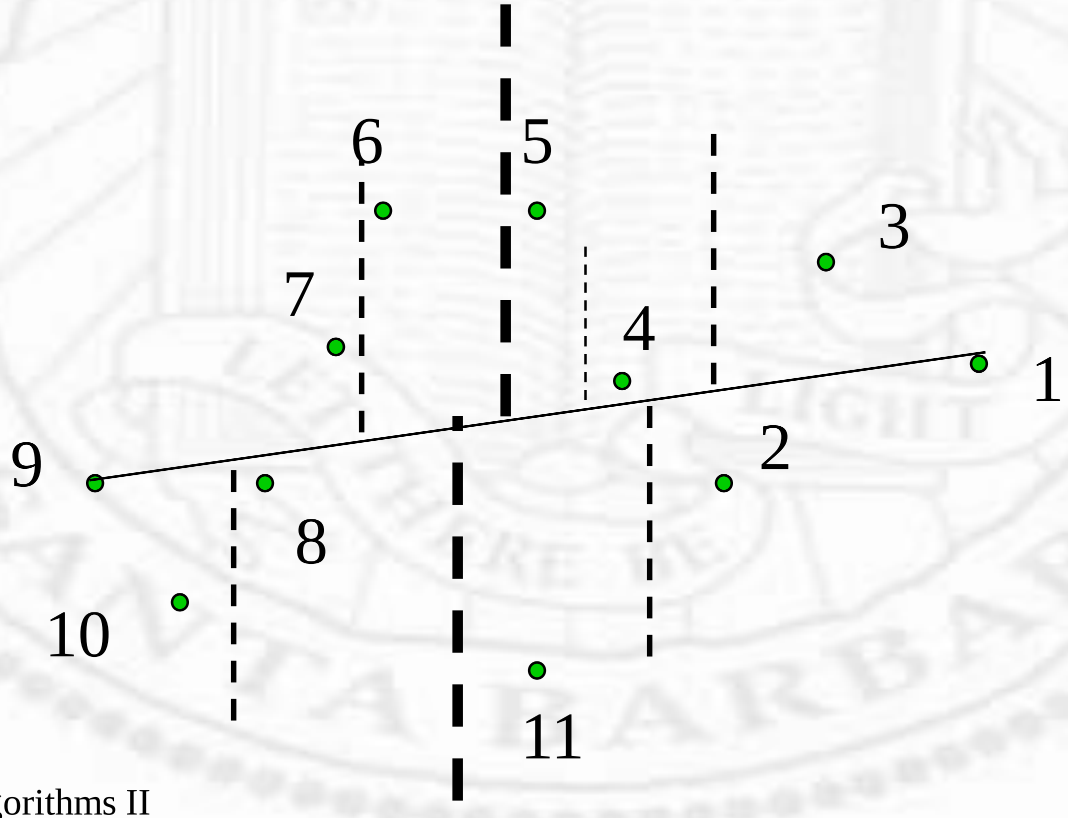
- Example
  - 0 is the base
  - 1,2,3,4 will be included in the hull (all make left turns)
  - when 5 is considered
    - 4 is eliminated (3,4,5 right turn)
    - 3 is eliminated (2,3,5 right turn)
    - 2 is eliminated (1,2,5 right turn)
    - 1 is kept (0, 1,5 left turn)
    - 5 is added



Complexity

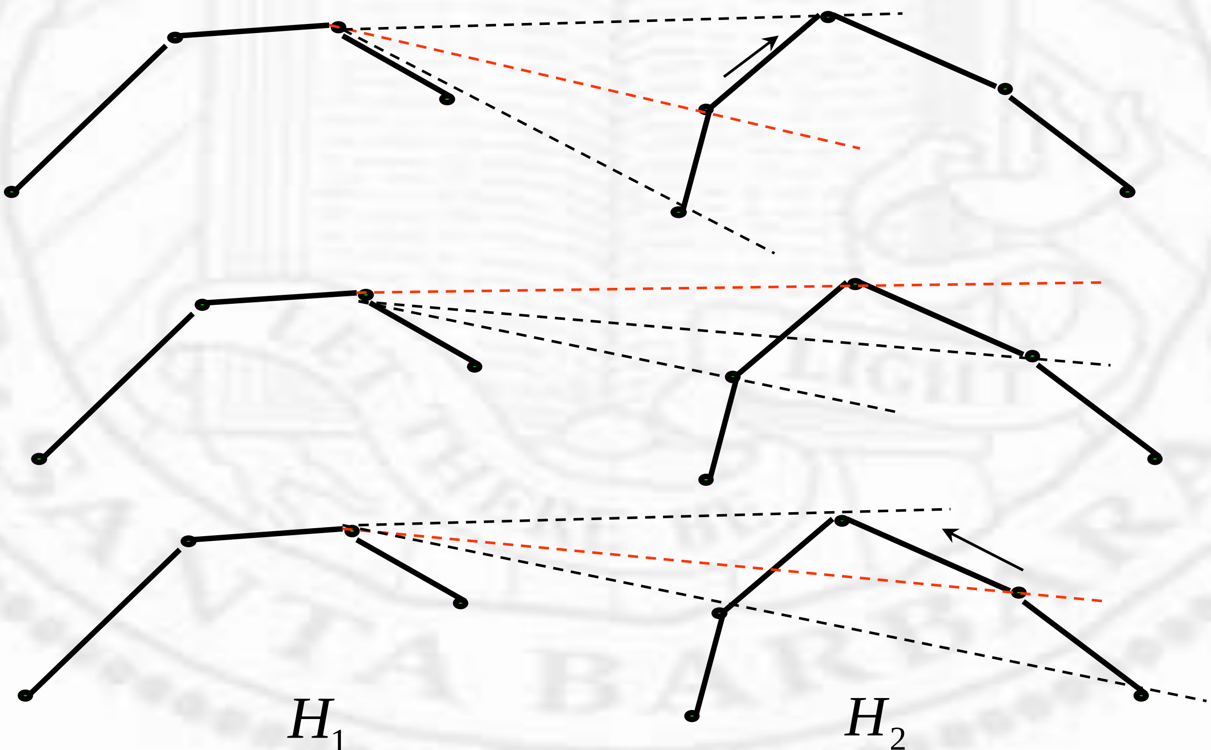
- Angular sorting O(nlogn)
- With n vertices
  - Loop: add vertices to the CH
  - Loop: delete vertices from the CH
  - Each vertex can be added and/or deleted only once
  - Each add/delete operation takes constant time (inner product)
  - O(n) total
- Whole operation: O(nlogn)

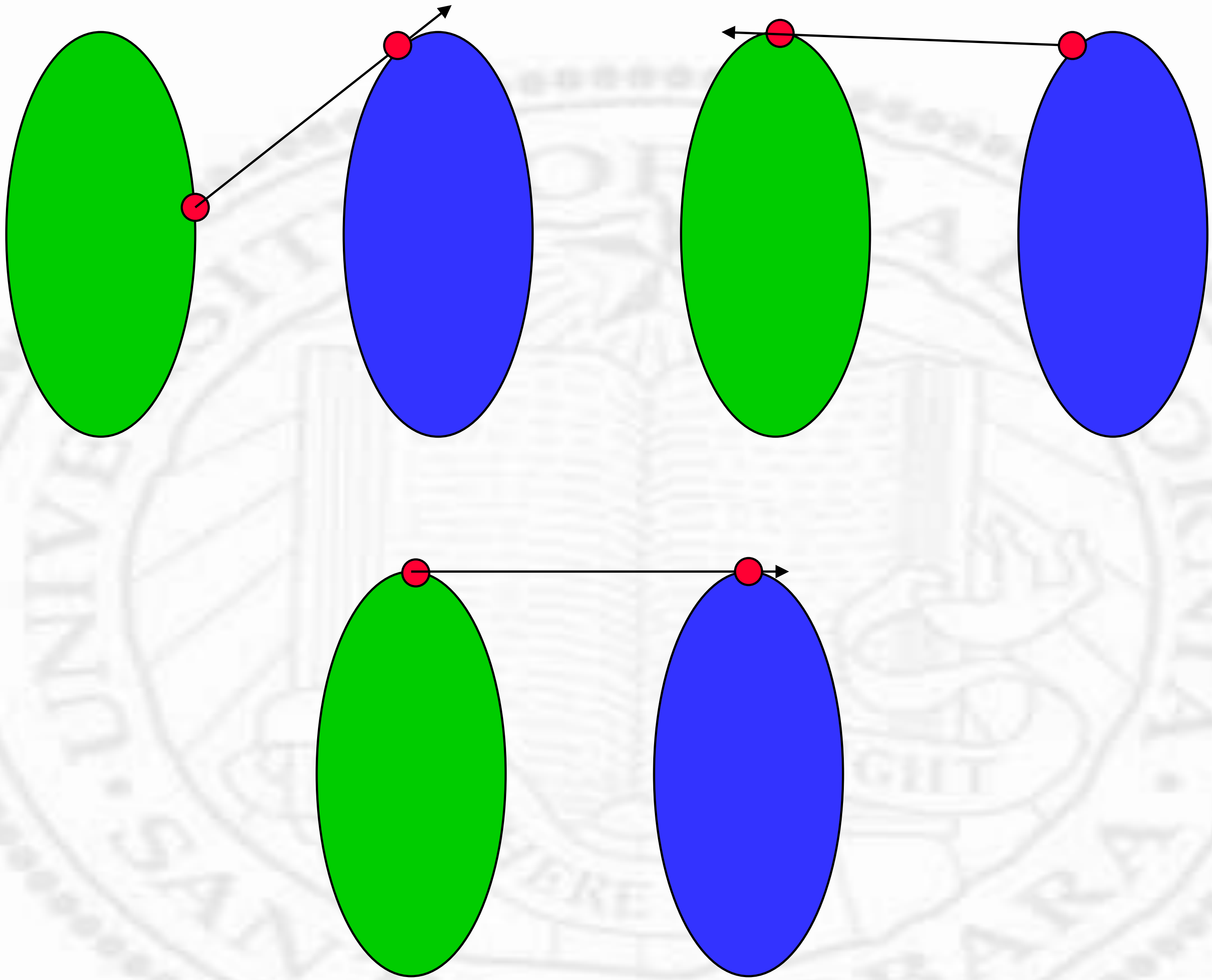
- Divide-and-Conquer
  - Upper hull and lower hull division (not essential)
  - Recursive division



Merge

- Intuition: connecting extreme points (points with the largest *y* coordinate on two hulls)
- Or more precisely, move the connecting lines are high (low) as possible for upper hull (lower hull)
- sort by *y*, too expensive (*O(nlogn)*)
- hill climbing (binary search on sorted *x*)





- If H1 and H2 are two upper hulls with at most m points each. If *p* is any point on H1, its point of tangency, *q*, with H2 can be found on O(*logm*) time
- If H1 and H2 are two upper hulls with at most m points each, their common tangent can be found on O(*log^2 m*) time
- The Divide-and-Conquer convex hull algorithm has a complexity of O(*nlogn*)

UpperTangent(HA ; HB) :

(1)Let a be the rightmost point of HA .

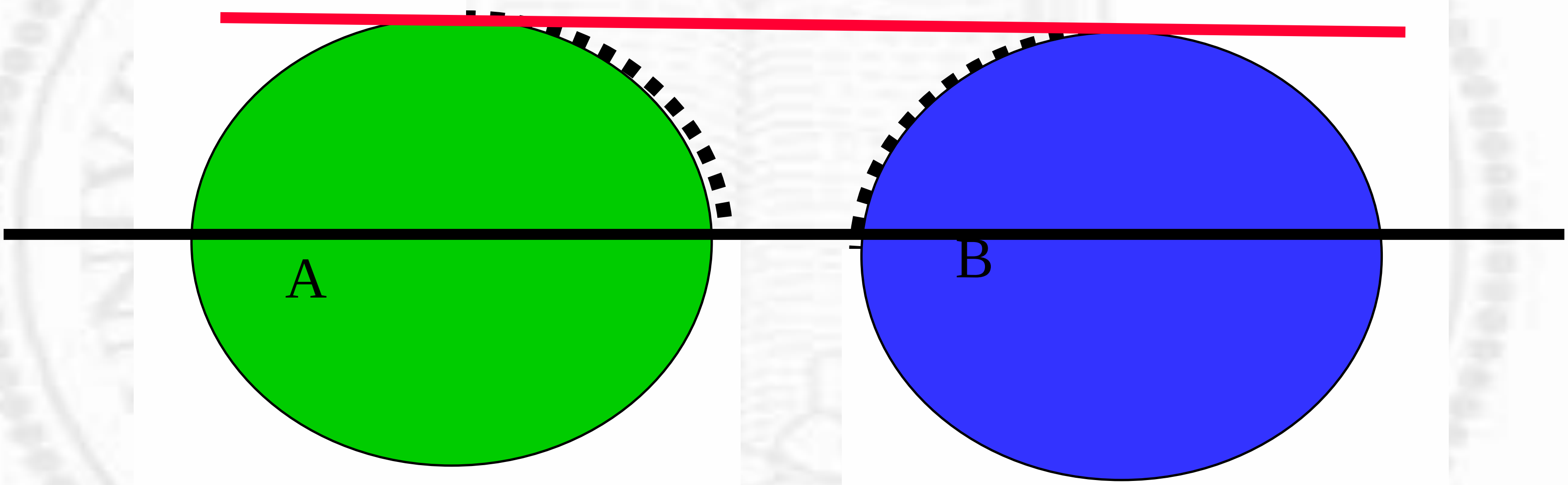
(2)Let b be the leftmost point of HB .

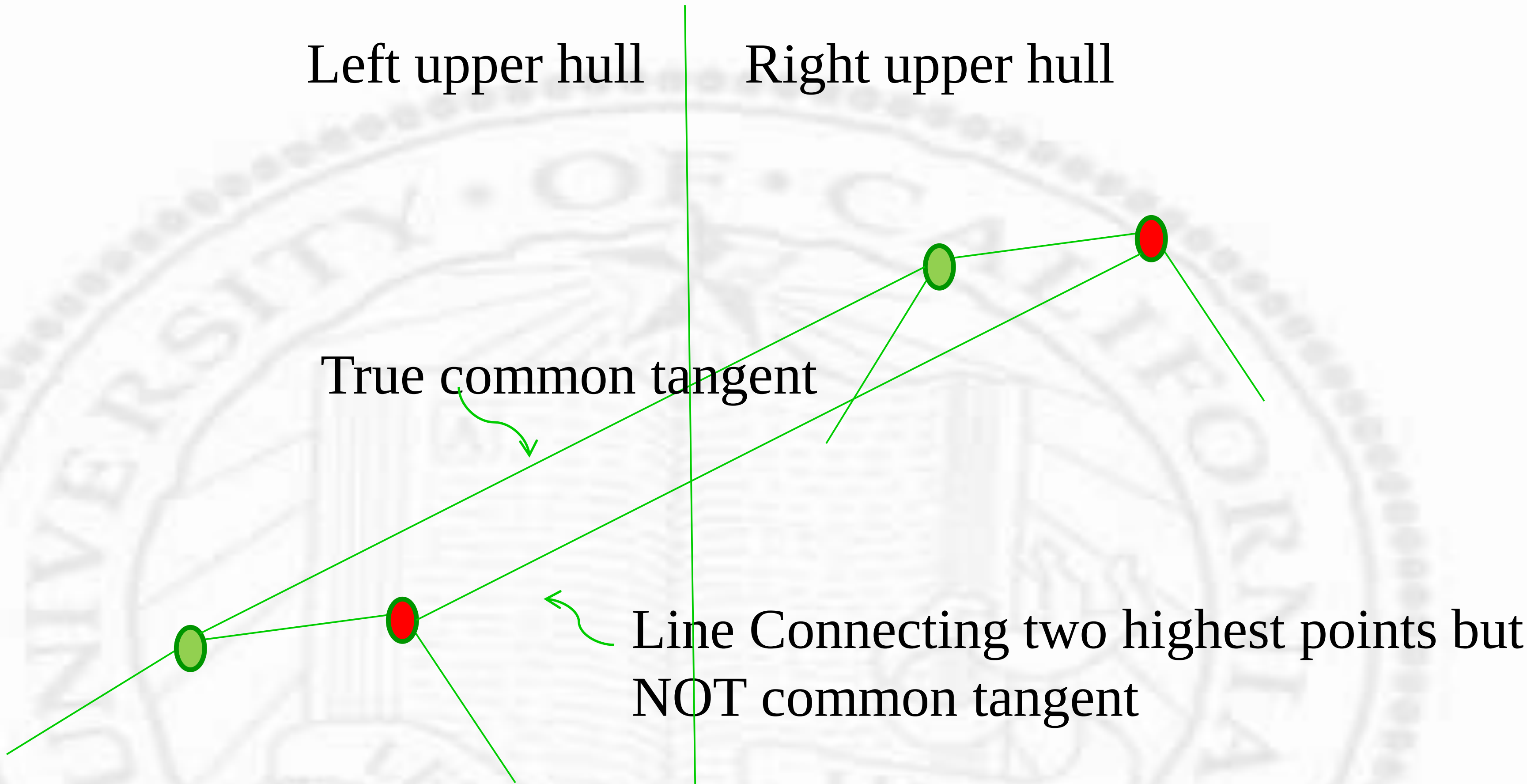
(3)While ab is not a upper tangent for HA and HB do

(a) While ab is not a upper tangent to HA do a = a - 1 (move a counterclockwise).

(b) While ab is not a upper tangent to HB do b = b + 1 (move b clockwise).

(4) Return ab.



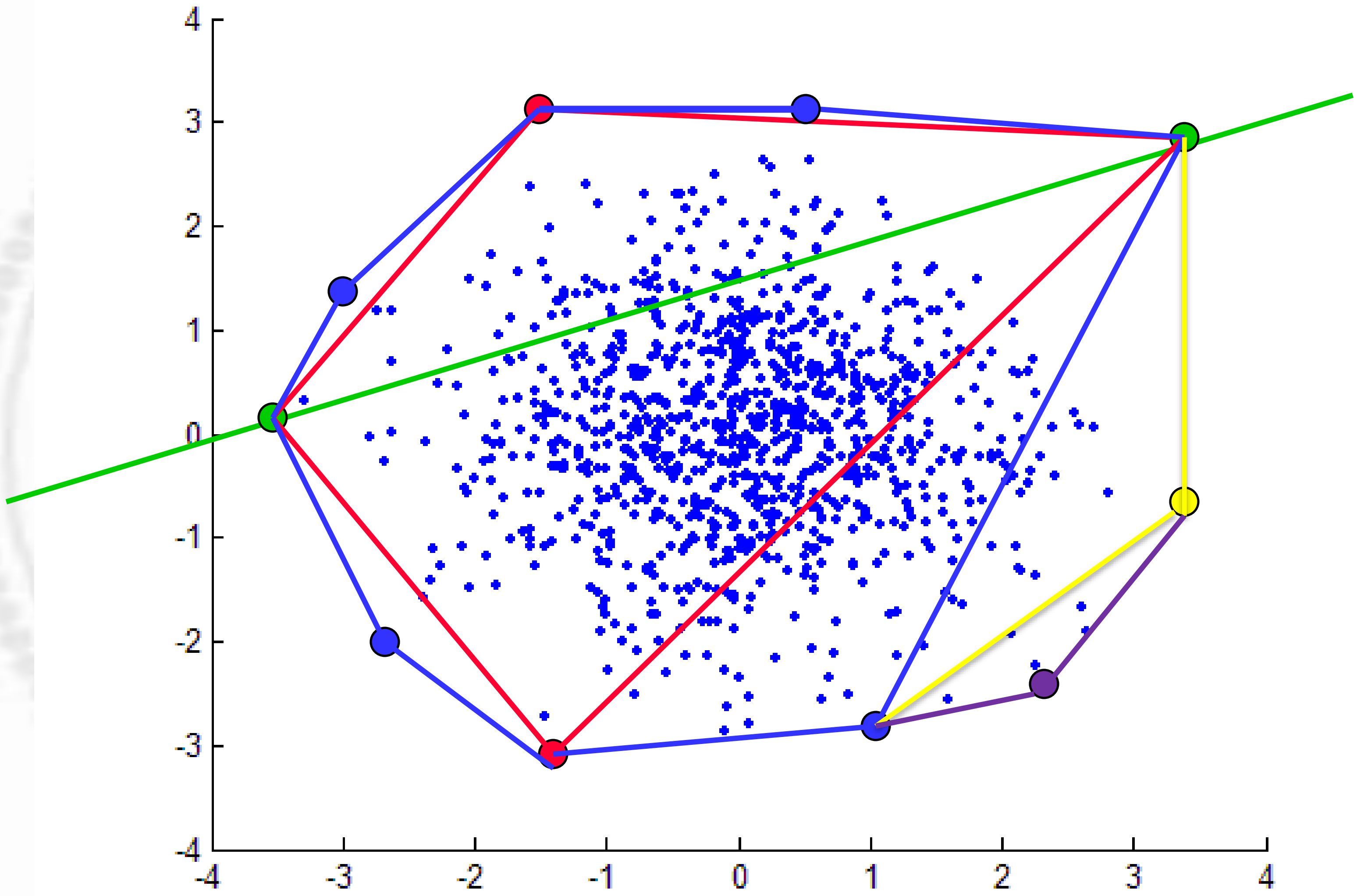


- Nitty-Gritty Details
  - Line connecting two highest points in component hulls is NOT necessarily the common tangent

- Time complexity
  - Upper and lower hulls division
    - largest and smallest x points *O(n)*
    - partition points into two halves *O(n)*
  - Recursive division
    - sort points by x *O(nlogn)*
    - main step

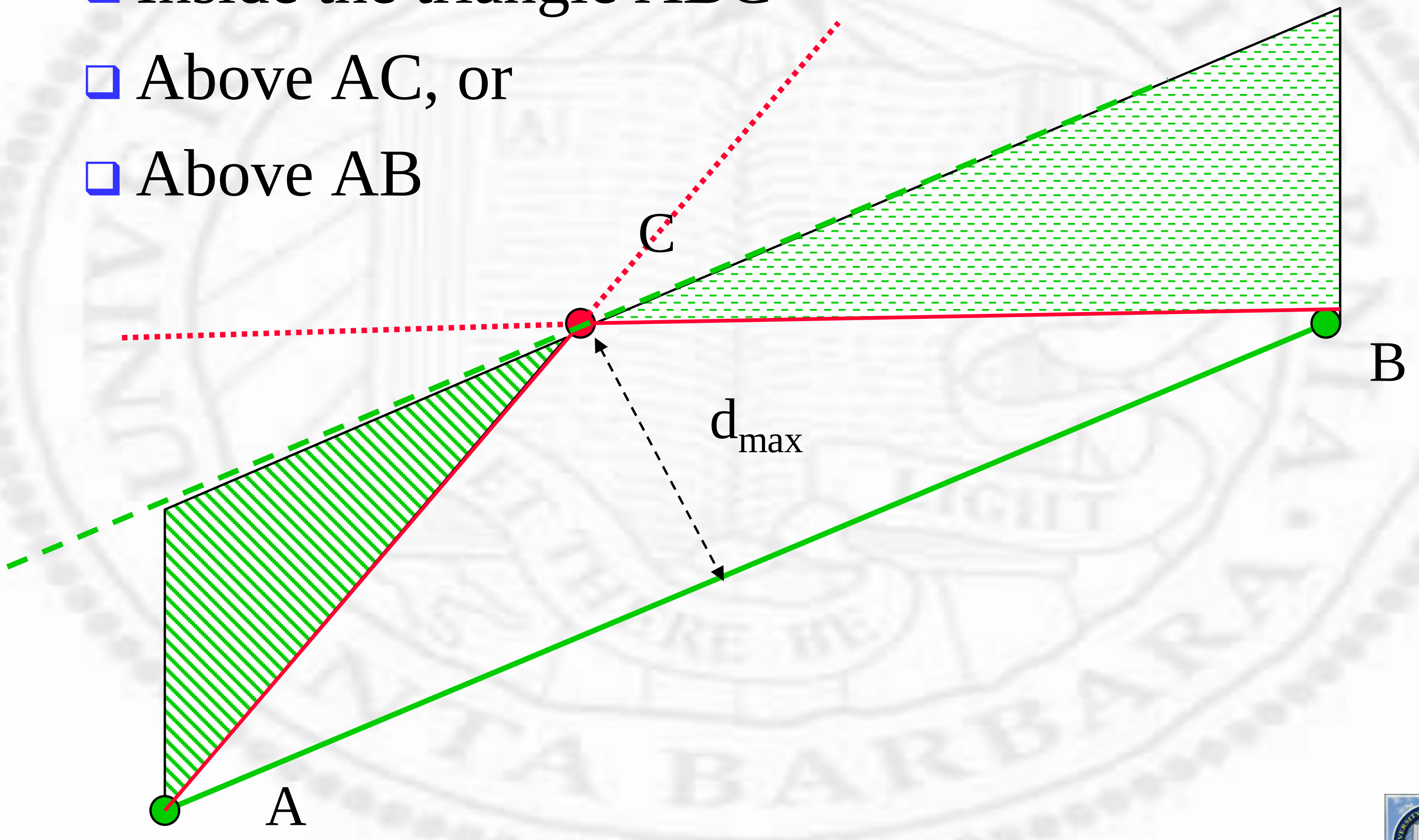
$$T(n) = 2T(\frac{n}{2}) + merge = 2T(\frac{n}{2}) + O(\log^2 n)$$
$$= O(n \log n)$$

Yet Another Divide-and-Conquer Algorithm (QuickHull)



Graphical Illustration

- Three possibilities O(n) time:
  - Inside the triangle ABC
  - Above AC, or
  - Above AB



Complexity

- If points are uniformly distributed in a unit square, expected # of points on the hull is O(logn)
- Quickhull discards interior points very quickly and narrows in peripheral points
- Like Quicksort, average time is O(nlogn) but worst case performance is O(n^2)

Complexity

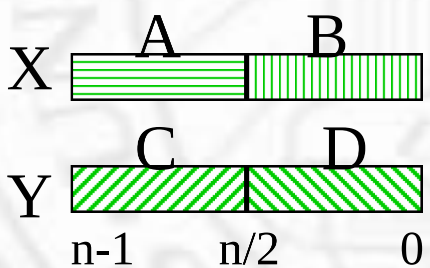
- Quick sort
  - Select pivot (O(1))
  - Partition into two parts O(n)
  - Recursive division
  - Trivial concatenation
  - T(n) = T(i) + T(n-i) + O(n)
- Quick hull
  - Select furthest point (O(n))
  - Partition into three parts O(n)
  - Recursive division
  - Trivial concatenation
  - T(n) = T(i) + T(n-i) + O(n)

Moral of the story

- Algorithm design is an art. We have seen three different convex hull algorithms
  - One based on domain knowledge only
  - Two based on divide-and-conquer

Multiplying Long Integers

- Input: two *n*-bit integer *x* and *y*
- Output: a *2n*-bit integer $x \times y$
- Divide-and-Conquer strategy



$$x = A2^{\frac{n}{2}} + B$$

$$y = C2^{\frac{n}{2}} + D$$

$$x \times y = (A2^{\frac{n}{2}} + B)(C2^{\frac{n}{2}} + D)$$

$$= AC2^{n} + (AD + BC)2^{\frac{n}{2}} + BD$$

- *Divide:* multiply 2 n-bit integers

  = 4 multiplies of 2 $n/2$-bit integers

  + 3 additions of integers ($2n$ bits)

  + 2 shifts
- *Small(p,q):* when the length becomes *1*
- *G(p,q): 1*-bit AND
- *Combine:* shift and addition

$$\overset{6}{\underbrace{\boxed{0}\boxed{1}}_{A}\underbrace{\boxed{1}\boxed{0}}_{B}} \times \overset{7}{\underbrace{\boxed{0}\boxed{1}}_{C}\underbrace{\boxed{1}\boxed{1}}_{D}}$$

$$\overset{A}{\boxed{0}\boxed{1}} \times \overset{C}{\boxed{0}\boxed{1}} \times 2^4 + \underline{\overset{A}{\boxed{0}\boxed{1}} \times \overset{D}{\boxed{1}\boxed{1}} + \overset{B}{\boxed{1}\boxed{0}} \times \overset{C}{\boxed{0}\boxed{1}}} \times 2^2 + \overset{B}{\boxed{1}\boxed{0}} \times \overset{D}{\boxed{1}\boxed{1}}$$

$$\boxed{0} \times \boxed{0} \times 2^2 + \underline{\boxed{0} \times \boxed{1} + \boxed{1} \times \boxed{0}} \times 2 + \boxed{1} \times \boxed{1}$$

$$\boxed{0} \times \boxed{1} \times 2^2 + \underline{\boxed{0} \times \boxed{1} + \boxed{1} \times \boxed{1}} \times 2 + \boxed{1} \times \boxed{1}$$

$$\boxed{1} \times \boxed{0} \times 2^2 + \underline{\boxed{1} \times \boxed{1} + \boxed{0} \times \boxed{0}} \times 2 + \boxed{1} \times \boxed{0}$$

$$\boxed{1} \times \boxed{1} \times 2^2 + \underline{\boxed{1} \times \boxed{1} + \boxed{0} \times \boxed{1}} \times 2 + \boxed{1} \times \boxed{0}$$

$$\boxed{0}\boxed{0}\boxed{0}\boxed{1} \times 2^4 + \underline{\boxed{0}\boxed{0}\boxed{1}\boxed{1} + \boxed{0}\boxed{0}\boxed{1}\boxed{0}} \times 2^2 + \boxed{0}\boxed{1}\boxed{1}\boxed{0}$$

00010000 + 00010100 + 00000110 = 00101010

=42

❖ Time complexity

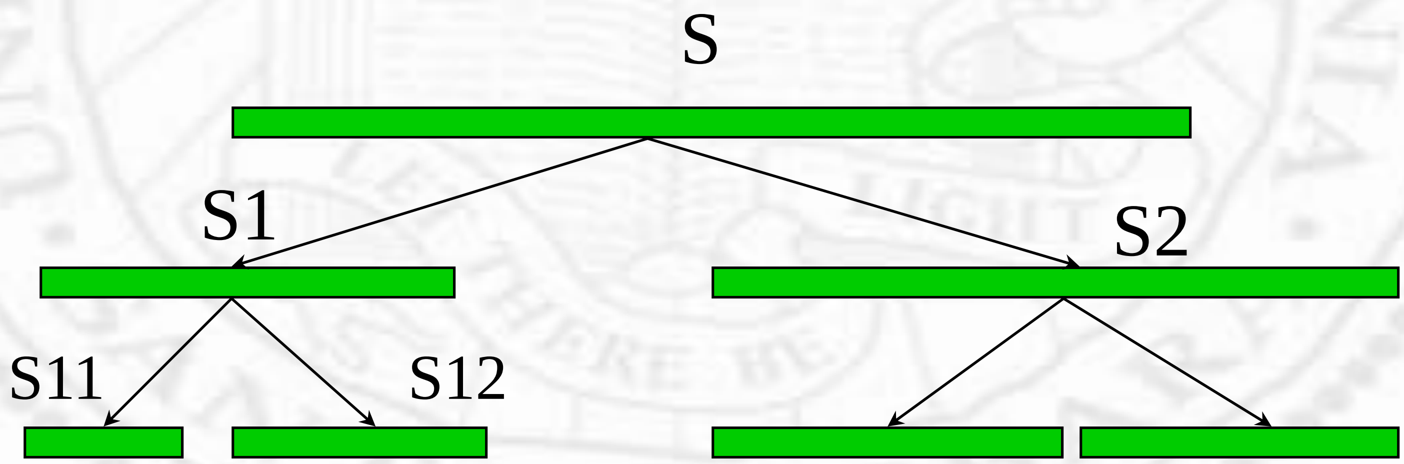
$$x \times y = (A2^{\frac{n}{2}} + B)(C2^{\frac{n}{2}} + D)$$

$$= AC2^n + (AD + BC)2^{\frac{n}{2}} + BD$$

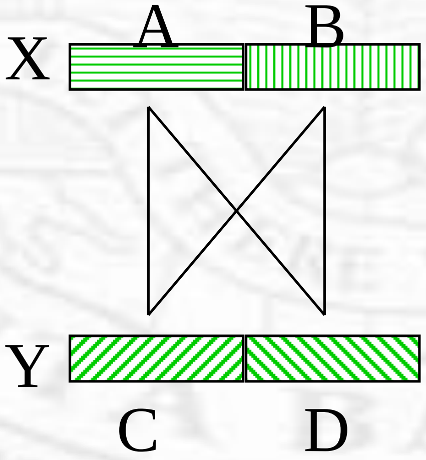
$$\begin{aligned}
T(n) &= 4T(\frac{n}{2}) + cn \\
&= 4(4T(\frac{n}{4}) + c\frac{n}{2}) + cn = 4^2 T(\frac{n}{4}) + cn(1+2) \\
&\ldots \\
&= 4^k T(1) + cn(1 + 2 + \ldots + 2^{k-1}) \\
&= 4^{\log n} a + cn(2^{\log n} - 1) \\
&= O(n^2)
\end{aligned}$$

- cf. brute force method $O(n^2)$

- Why no improvement using divide-and-conquer?
  - in quick sort
    - elements in *S1* do not compare with those in *S2*
    - elements in *S11* do not compare with those in *S12*
    - *problems are* ***decomposable*** *and* ***independent***



- In integer multiplication
  - problems are *decomposable* **but not** i*ndependent*
  - the number of multiplications ***is not*** reduced
  - The fancy way of decomposing the solution still requires every digit in one number to "touch" every digit in the other number (*no* sharing, *no* reuse)



- Possible improvements: through sharing

$$x \times y = (A2^{\frac{n}{2}} + B)(C2^{\frac{n}{2}} + D)$$

$$= AC2^{n} + (AD + BC)2^{\frac{n}{2}} + BD \qquad 4\times,3+,2 \leftarrow$$

$$= AC2^{n} + \{(A - B)(D - C) + AC + BD\}2^{\frac{n}{2}} + BD$$

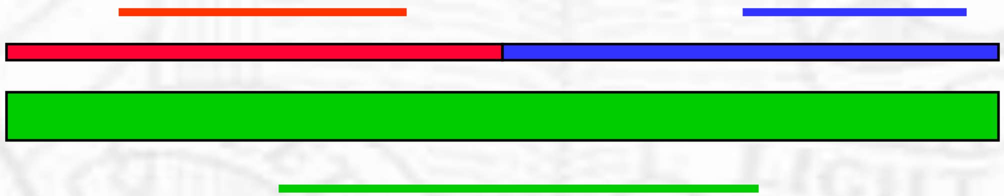
$$3\times,6+,2 \leftarrow$$

$$T(n) = 3T(\frac{n}{2}) + cn = O(n^{\log_2 3}) = O(n^{1.59})$$

Maximum Sum

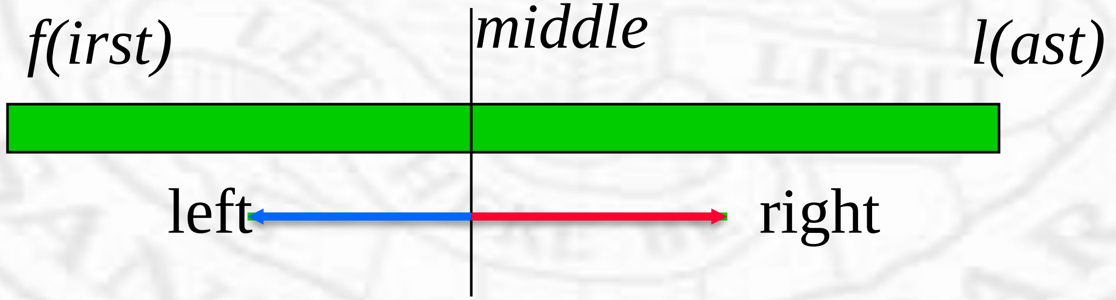
- Just to confuse you more, it is not to say that the subproblems must be totally independent for divide-and-conquer to work
- Given: an array of *n* numbers, possibly negative
- Find: maximum subsequence sum (if all numbers are negative, then the maximum sum is 0)
- -4, 10, 12, -5, -7, 8, 3, 1 is 22

- How does divide-and-conquer work?
  - Divide the array into two parts
  - Compute the maximum sum in each part
  - The global maximum sum is the largest of the two
  - but …
- What happens if the maximum sum sequence straddles the boundary?

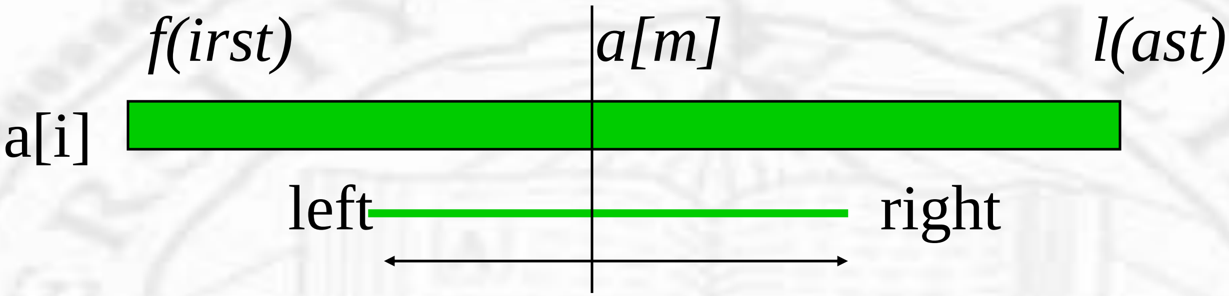


- 4, -3, 5, -2, -1, 2, 6, -2

- Start from the middle
  - Accumulate from middle moving leftward, keep the largest sum
  - Accumulate from middle moving rightward, keep the largest sum
  - The largest partial sum across two parts must be the sum of the above two



- Need a third term which captures the maximum sum of straddling sequence



$$b_l = b[0] = a[m];$$
$$for(i = m-1; i >= f; i--)$$
$$\quad b[m-i] = b[m-i-1] + a[i];$$
$$\quad if\ (b[m-i] > b_l)\ then$$
$$\qquad \boxed{b_l} = b[m-i]$$

$$b_r = b[0] = a[m+1];$$
$$for(i = m+2; i <= l; i++)$$
$$\quad b[i-m-1] = b[i-m-2] + a[i];$$
$$\quad if\ (b[i-m-1] > b_r)\ then$$
$$\qquad \boxed{b_r} = b[i-m-1]$$

Retained largest partial sums

- Then the maximum sum is the largest of three terms: two from left and right, one *bl+br*

- Complexity

$$T(n) = 2T(\frac{n}{2}) + n = O(n \log n)$$

- Bruteforce method

for (f=1; f<=n; f++) ← All possible first pos

for (l=f; l<=n; l++) ← All possible last pos

for (k=f; k<=l; k++) ← Sum from first to last

Add up all the a[k]

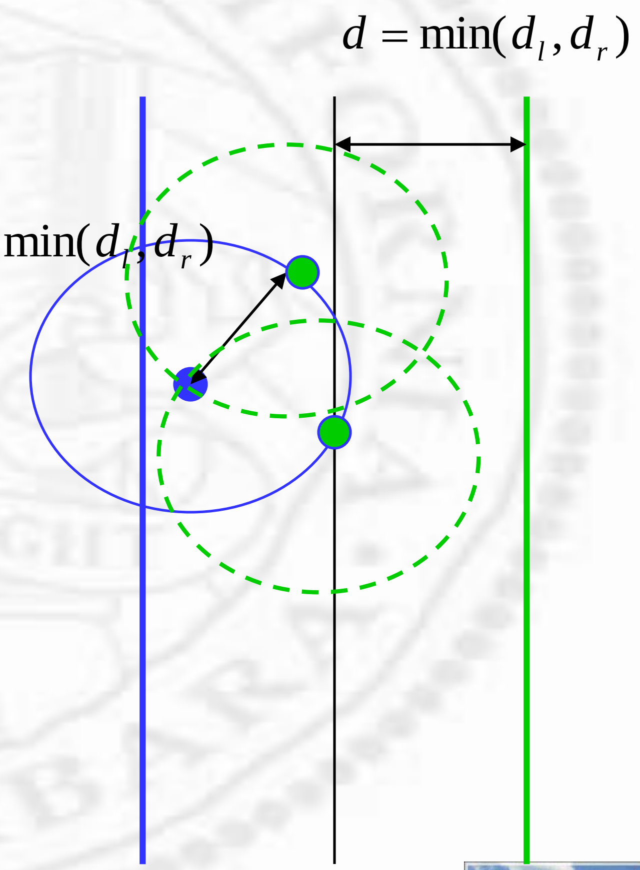
will be O(n^3)

Closest Pair of Points

- Given a set of points on a plane, find the two points which are closest to each other
- Brute force method is O(n^2)
- Can divide-and-conquer do better?
- Obvious solution:
  - partition data sets into two halves (recursively)
  - closest pair of points are in
    - the left half or right half
    - one each in each half

- The closest points in the left and right halves can be found recursively
- But how to find points across boundary?
  - Obvious solution: check each *n/2* points in the left against each *n/2* points in the right
  - The solution will be O(n^2), no better than brute force method
- Again, the problem is that two problems are not entirely independent and combining subsolutions can be tricky.

- Goal: if we want an O(nlogn) solution, then the combination step must be of O(n)
- What is the linear solution in combination?
- A clever trick



- Q: How many points do you have to check?
- A: No blue (green) point can lie inside the circle of radius *d* around another blue (green) point

Check box d(in x) and 2d(in y)

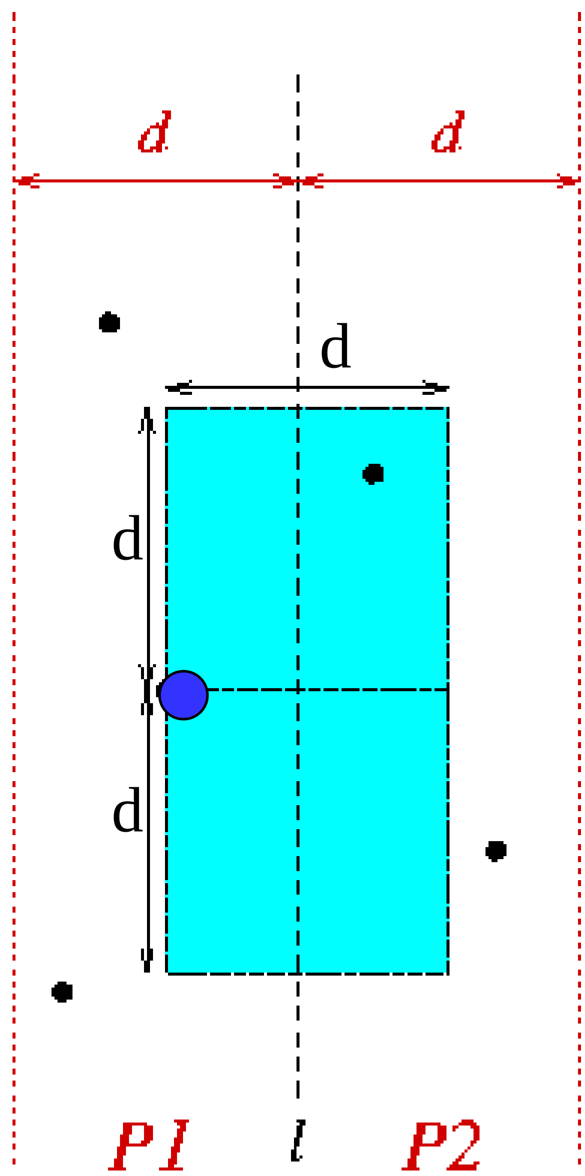
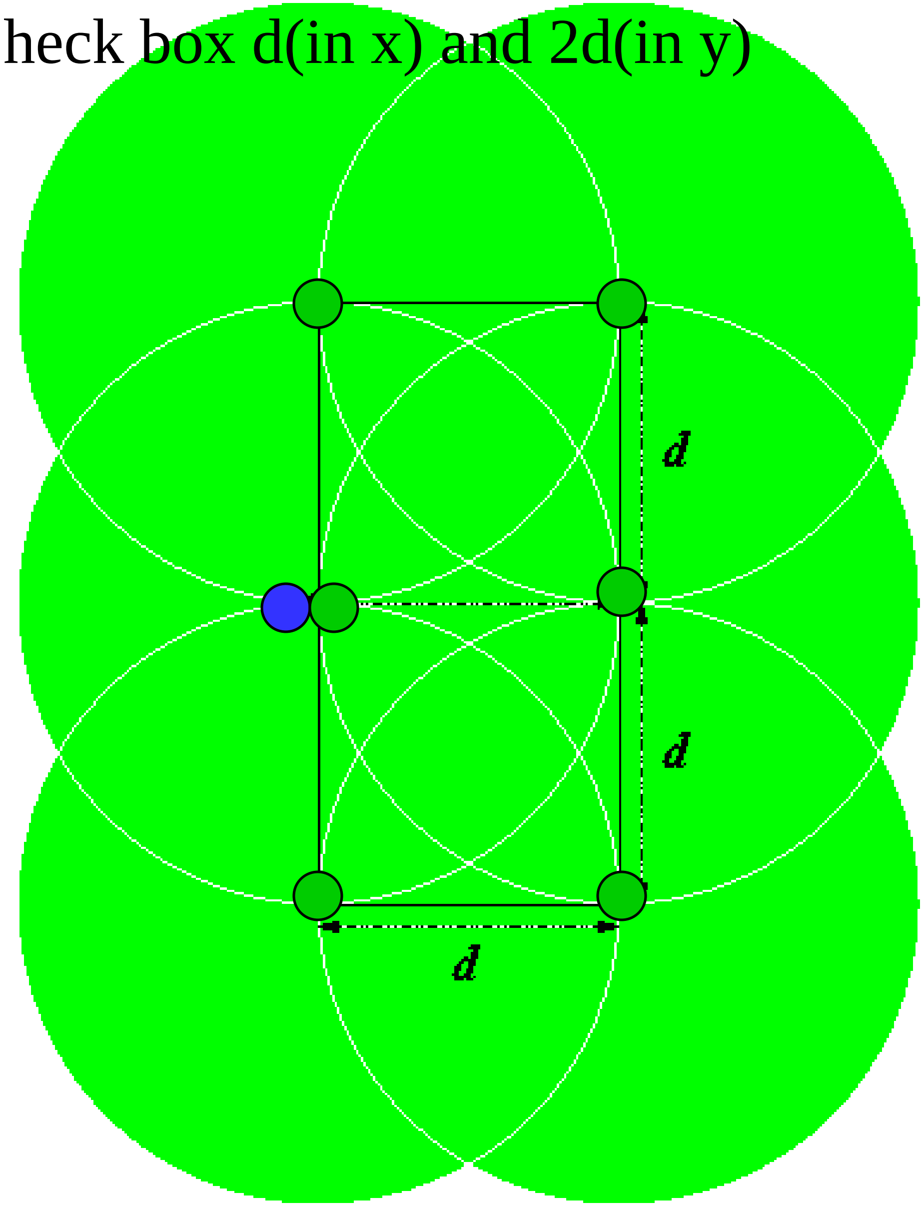
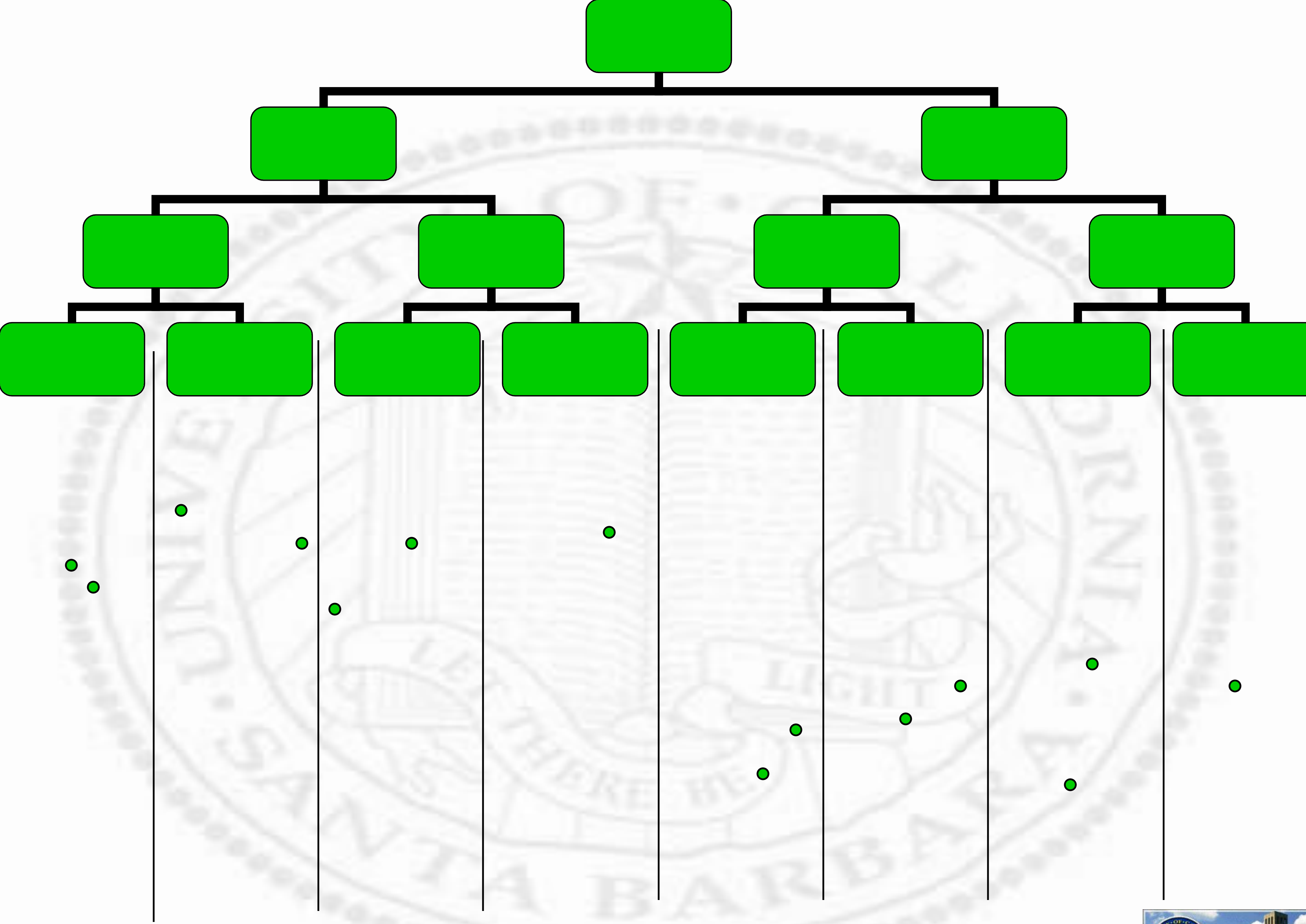
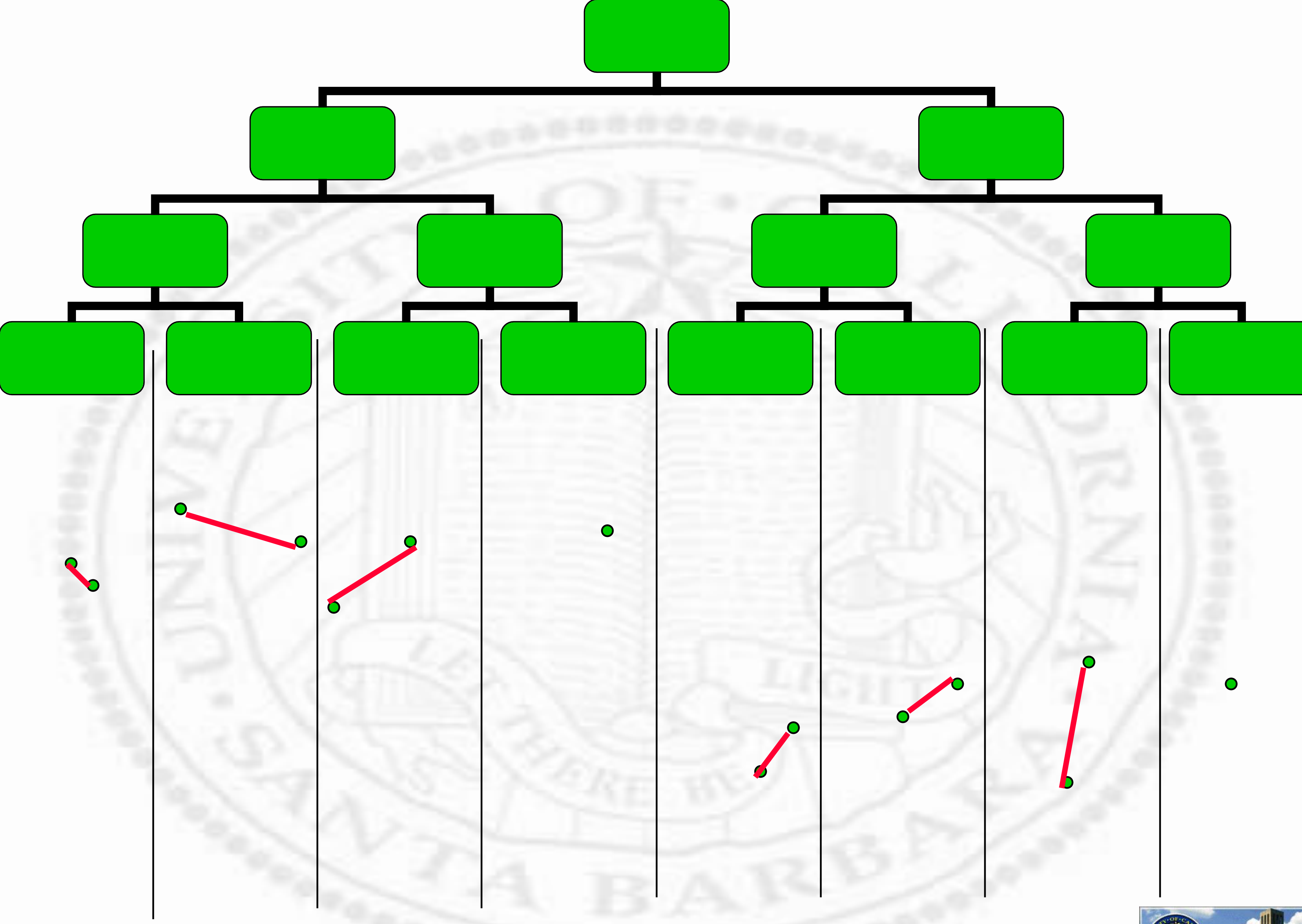
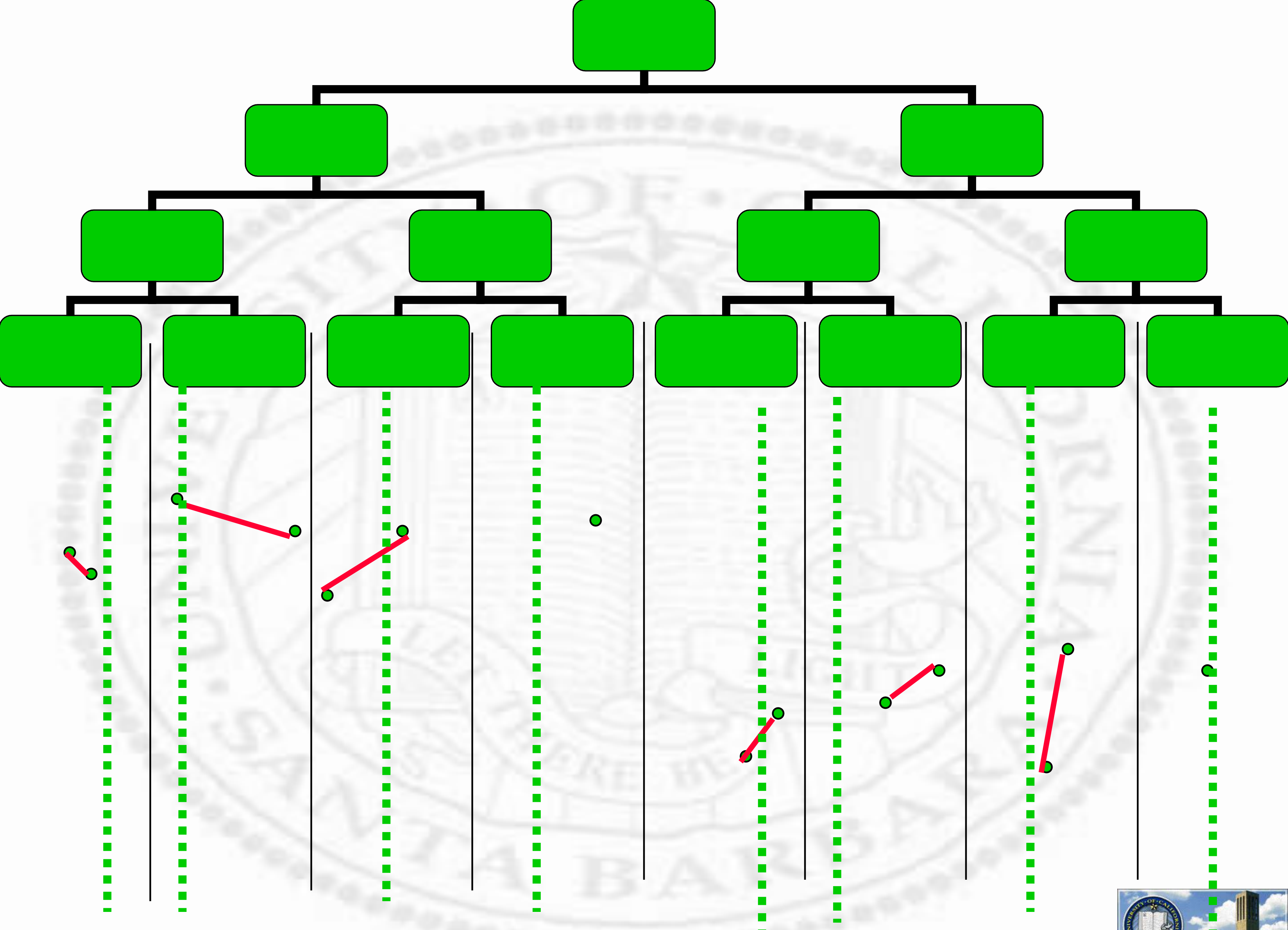


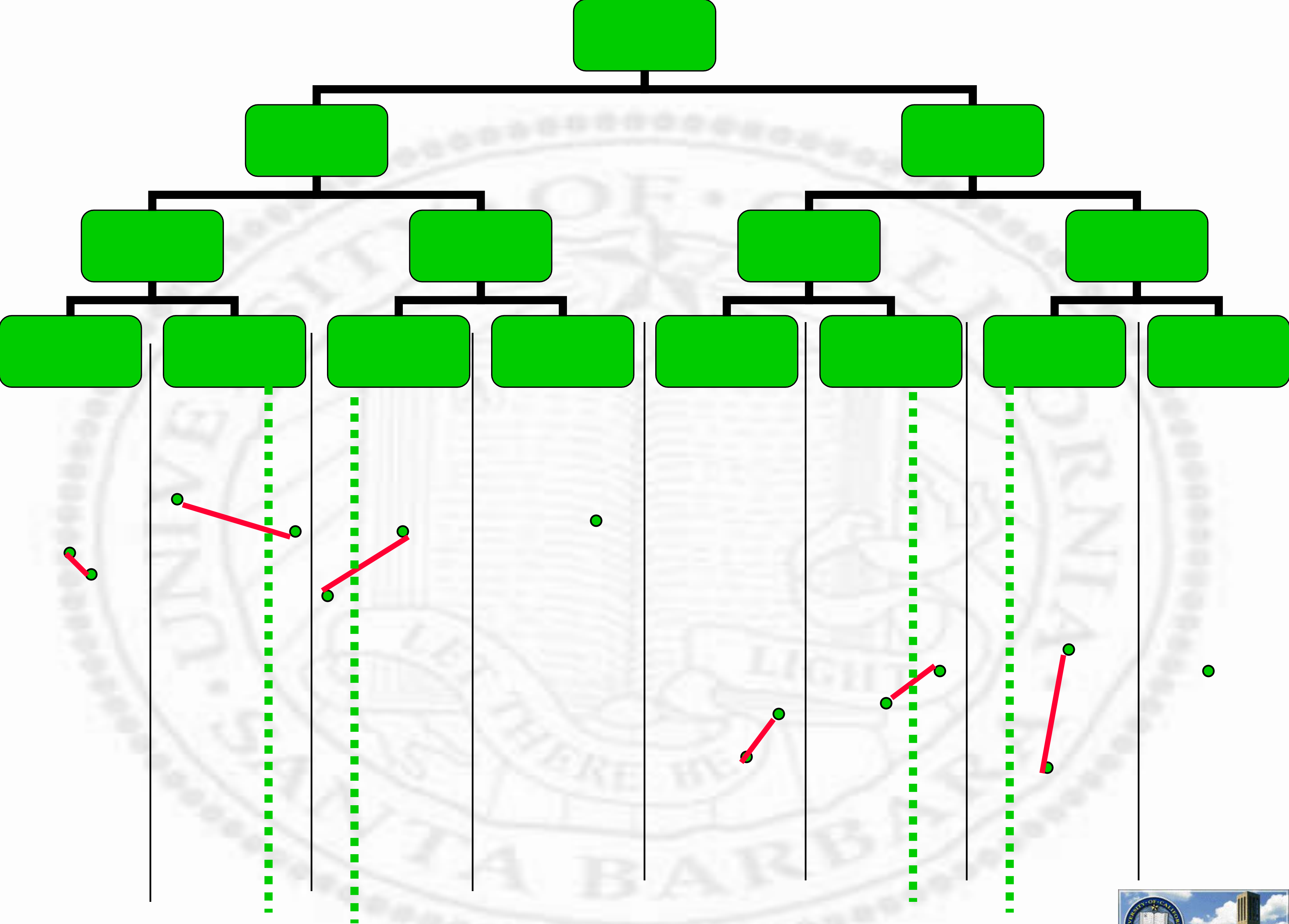
Figure 3.3

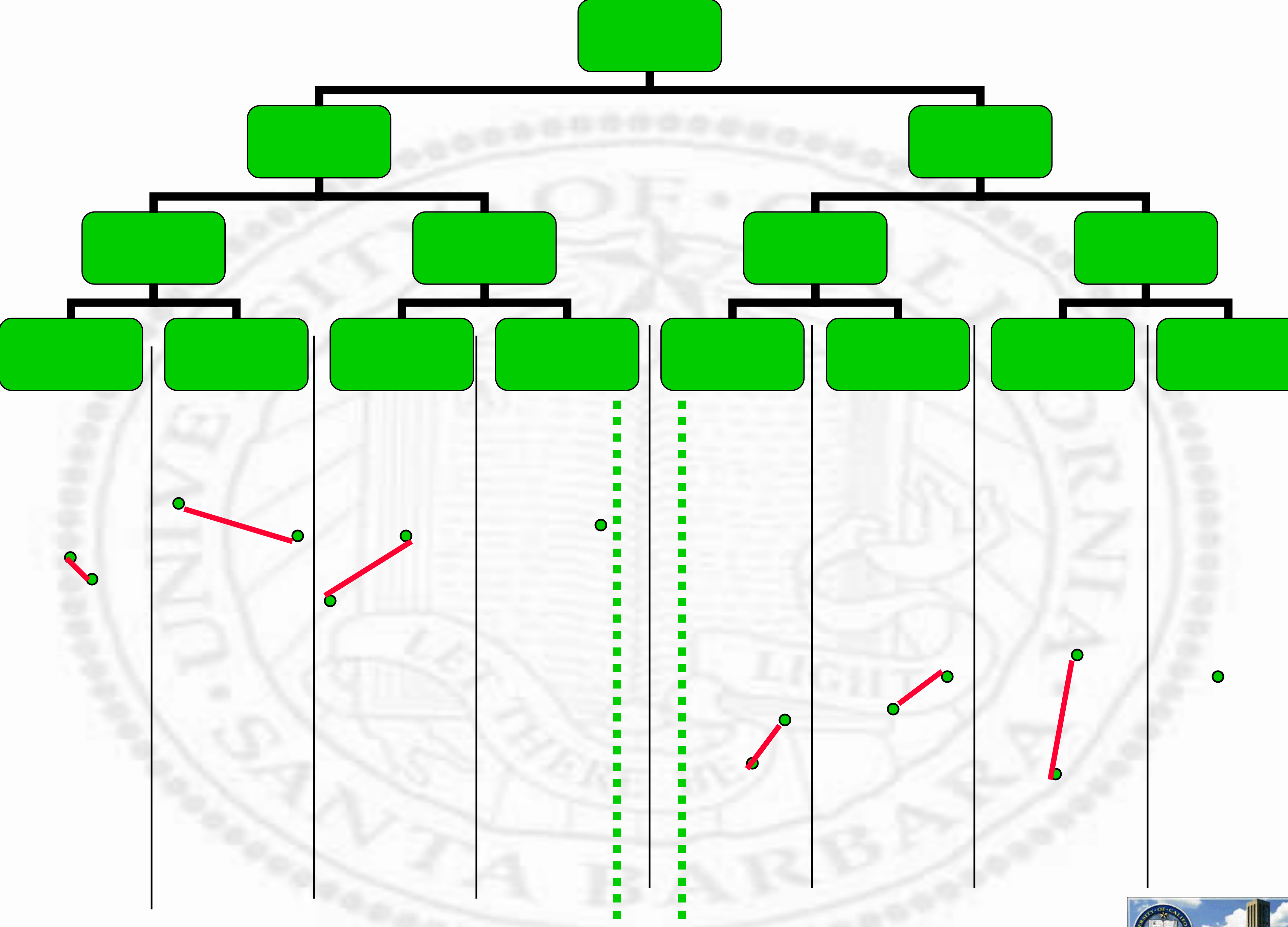












Summary

- How to divide?
  - 1 to 2
    - equal size, e.g. merge sort
    - unequal size, e.g. quick sort
  - 1 to many
    - binary search, Tower of Hanoi (1 to 3)
    - integer multiply, matrix multiply (1 to many)

Summary (cont.)

- When to terminate recursion?
  - depend on the problem at hand
    - simple comparison (binary search)
    - simple move (Hanoi tower)
- How to combine partial results?
  - nothing (binary search)
  - concatenation (quick sort)
  - merge (merge sort)
  - addition and shift (integer multiplication)