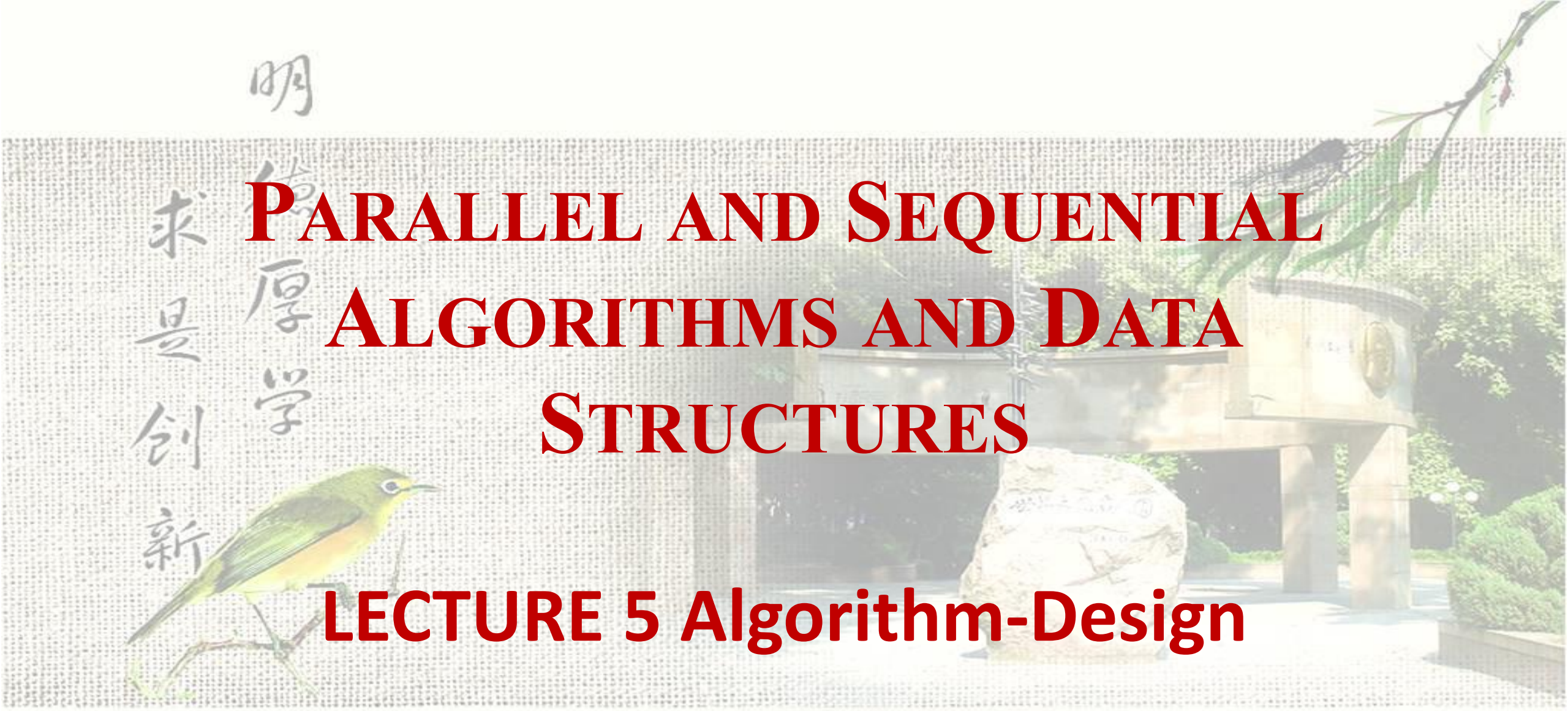


PARALLEL AND SEQUENTIAL ALGORITHMS AND DATA STRUCTURES

LECTURE 5 Algorithm-Design

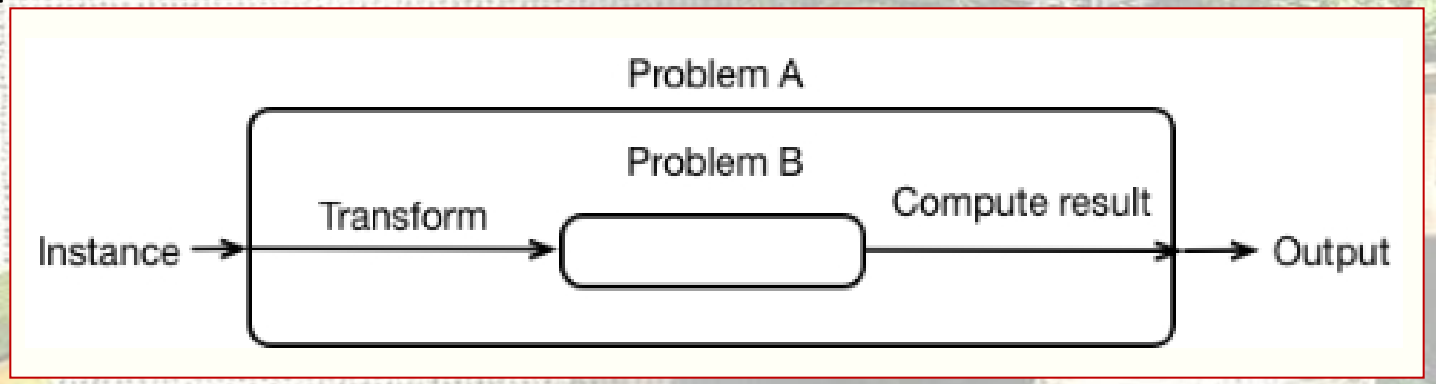
華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

SYNOPSIS

- Basic Techniques
- Divide and Conquer
- Contraction
- Maximum Contiguous Subsequence Sum

Basic Techniques

- two basic algorithm-design techniques
  - Reduction



  - brute force

Reduction

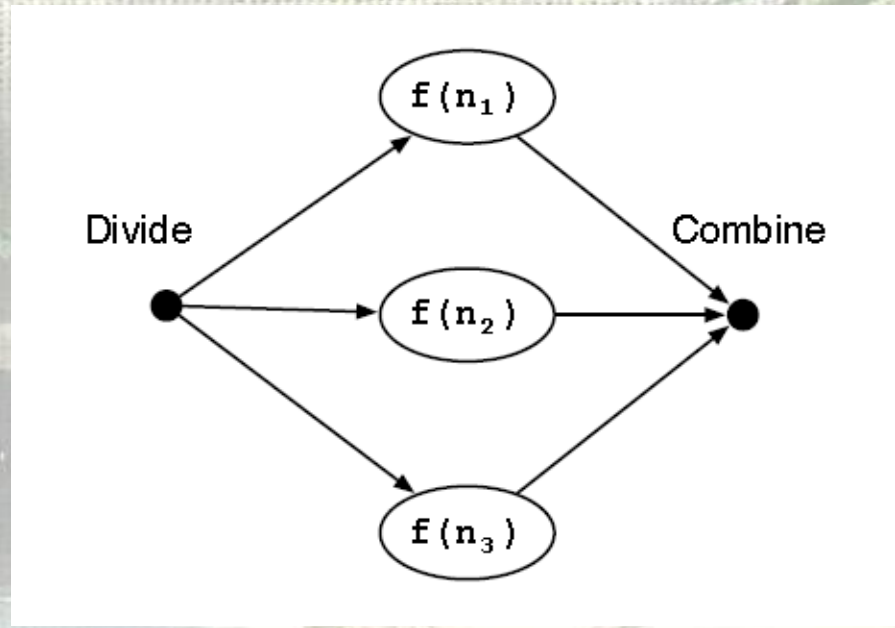
- **Example**
  - **How to find the minimum of a set of values?**
    - ✓**consider a function maxVal that finds the maximum of a set of numbers**
    - ✓**Consider a function of (comparison) sorting**
- **Efficiency of Reduction**
  - a reduction as being ***efficient*** if the total cost of the input and output transformations are asymptotically the same as that of the problem being reduced to
  - **In particular, if we know (or conjecture) that problem A is hard (e.g. requires exponential work), and we can efficiently reduce (e.g. using polynomial work) it to problem B, then we know that B must also be hard**

Brute Force

- The brute-force technique involves enumerating all possible solutions, a.k.a., ***candidate solutions*** to a problem, and checking whether each candidate solution is valid
- **Given that the brute-force algorithms are often inefficient, why do we care about them?**
  - are usually easy to design and are a good starting point toward a more efficient algorithm
  - can help in understanding the structure of the problem, and lead to a more efficient algorithm
  - In practice, a brute-force algorithm can help in testing the correctness of a more efficient algorithm by offering a solution that is easy to implement correctly

Algorithm-Design Techniques

- **Inductive techniques**
  - **Divide and conquer**
  - **Contraction**
  - **Dynamic programming**
  - **Greedy**
  - **Randomization**



SYNOPSIS

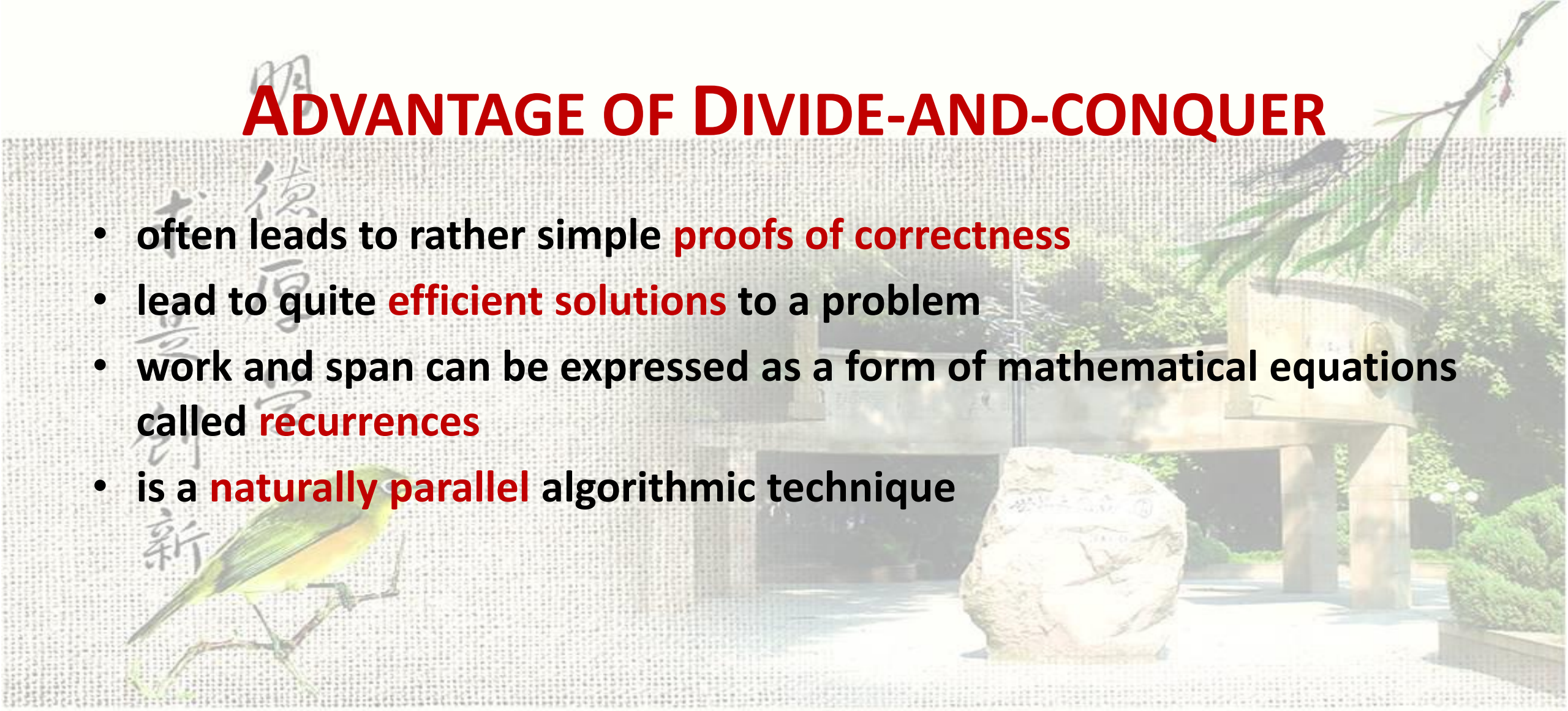
- Basic Techniques
- Divide and Conquer
- Contraction
- Maximum Contiguous Subsequence Sum

DIVIDE-AND-CONQUER ALGORITHM

- **Base Case**
  - When the instance I of the problem P is sufficiently small, return the answer P(I) directly, or resort to a different, usually simpler, algorithm that is well suited for small instances
- **Inductive Step:**
  1. **Divide** I into some number of smaller instances of the same problem P.
  2. **Recurse** on each of the smaller instances to obtain their answers.
  3. **Combine** the answers to produce an answer for the original instance I.

ADVANTAGE OF DIVIDE-AND-CONQUER

- often leads to rather simple proofs of correctness
- lead to quite efficient solutions to a problem
- work and span can be expressed as a form of mathematical equations called recurrences
- is a naturally parallel algorithmic technique



Analysis of Divide-and-Conquer Algorithms

- **Consider an algorithm that divides a problem instance of size n into k>1 independent subinstances of sizes $n_1, n_2, \ldots n_k$, recursively solves the instances, and combine the solutions to construct the solution to the original instance**

$$W(n) = W_{\text{divide}}(n) + \sum_{i=1}^{k} W(n_i) + W_{\text{combine}}(n) + 1.$$

$$S(n) = S_{\text{divide}}(n) + \max_{i=1}^{k} S(n_i) + S_{\text{combine}}(n) + 1.$$

why?

Example

- **Maximal Element:** find the maximal element in a sequence
  1. If the sequence has only one element, we return that element,
  2. otherwise, we divide the sequence into two equal halves and recursively and in parallel compute the maximal element in each half
  3. then return the maximal of the results from the two recursive calls

How?

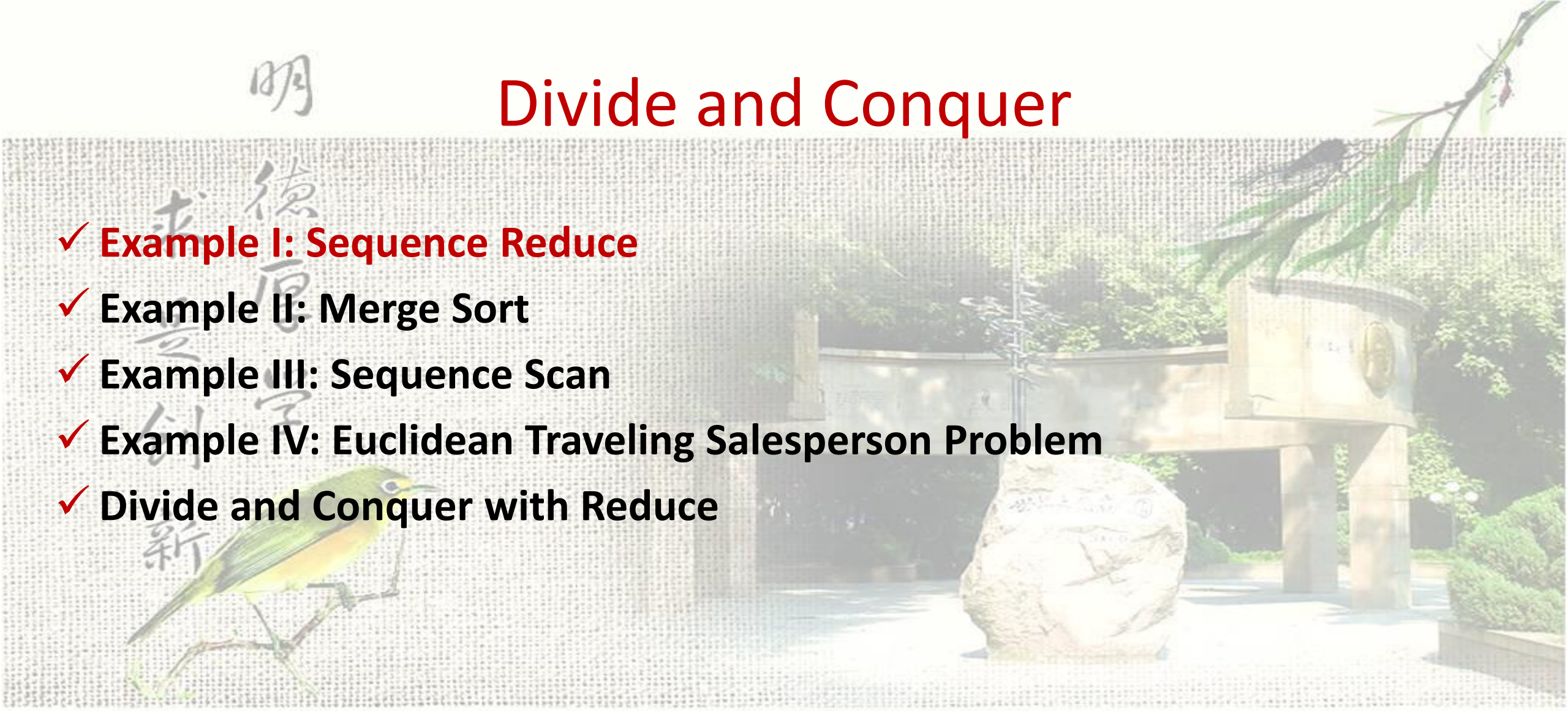
$$W(n) = \begin{cases} \Theta(1) & \text{if } n \le 1 \\ 2W(n/2) + \Theta(1) & \text{otherwise} \end{cases}$$

$$S(n) = \begin{cases} \Theta(1) & \text{if } n \le 1 \\ S(n/2) + \Theta(1) & \text{otherwise.} \end{cases}$$

$W(n) = \Theta(n)$ and
$S(n) = \Theta(\lg n)$.

Divide and Conquer

- ✓ **Example I: Sequence Reduce**
- ✓ **Example II: Merge Sort**
- ✓ **Example III: Sequence Scan**
- ✓ **Example IV: Euclidean Traveling Salesperson Problem**
- ✓ **Divide and Conquer with Reduce**



SEQUENCE REDUCE

- how we may implement reduce using divide and conquer?

```
reduceDC f id a =
  if isEmpty a then
    id
  else if isSingleton a then
    a[0]
  else
    let
      (l, r) = splitMid a
      (a, b) = (reduceDC f id l || reduceDC f id r)
    in
      f(a, b)
end
```

- Base Case ?
- Inductive step: divide recurse combine ?

SEQUENCE REDUCE

- ***A*** **is arraySequence?**
- **Can you write the recursions for *work* and *span*?**
  - **W(n) = 2W(n/2) + O(1), S(n) = S(n/2) + O(1)**
- **W=? S=?**

How to prove?

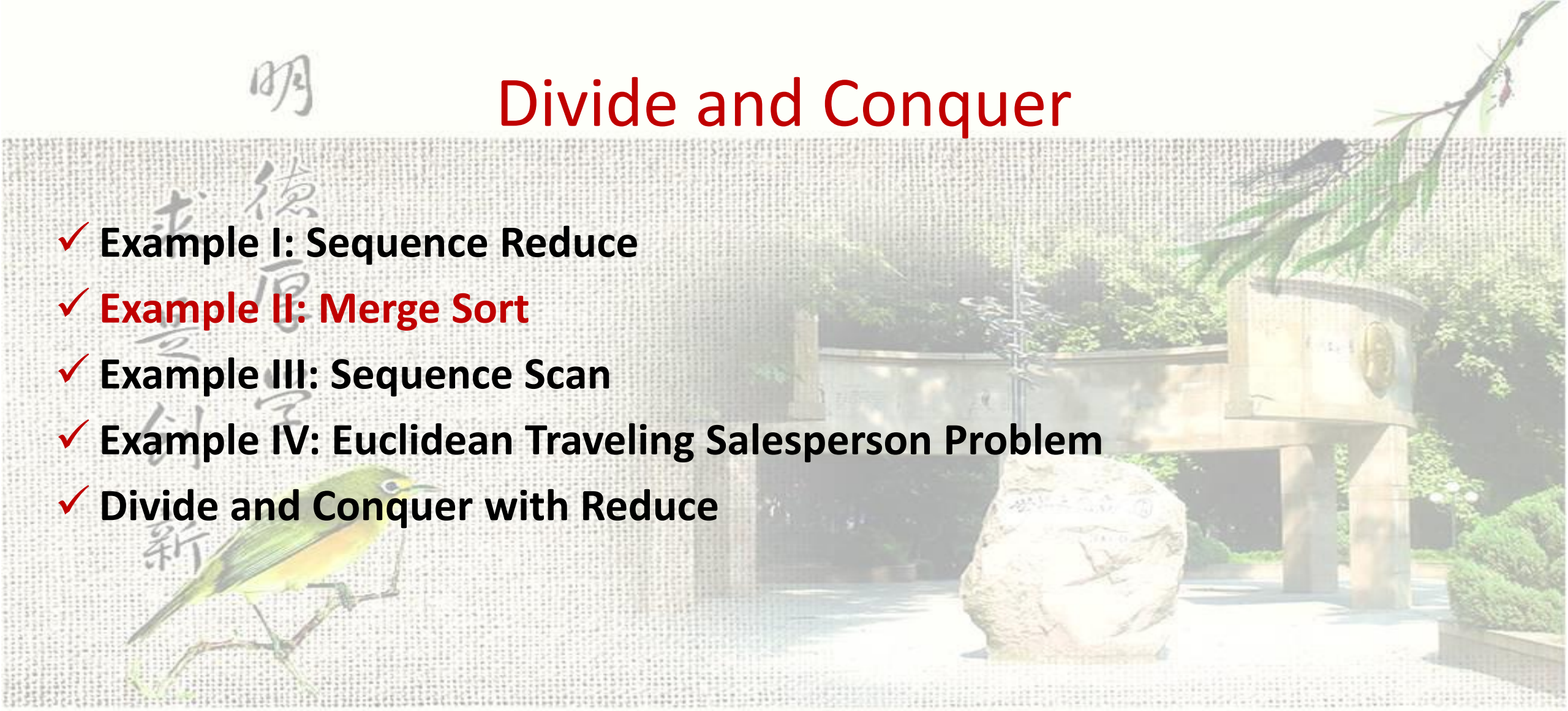
can?

can?

```
reduceDC f id a =
  if isEmpty a then
    id
  else if isSingleton a then
    a[0]
  else
    let
      (l, r) = splitMid a
      (a, b) = (reduceDC f id l || reduceDC f id r)
    in
      f(a, b)
end
```

Divide and Conquer

- ✓ Example I: Sequence Reduce
- ✓ Example II: Merge Sort
- ✓ Example III: Sequence Scan
- ✓ Example IV: Euclidean Traveling Salesperson Problem
- ✓ Divide and Conquer with Reduce



MERGE SORT

- **Sorting Problem**
  - Given a sequence S of elements from a universe U, with a total ordering given by <, return the same elements in a sequence R in sorted order, i.e. Ri≤ Ri+1; 0 < i<=|S|- 1
- **Mergesort and Quicksort**
  - Both use Θ(n log n) work
  - Mergesort, has a trivial divide step and interesting combine step
  - Quicksort, has an interesting divide step but trivial combine step

MERGE SORT

- **Merge sort**
  - splitMid: split a sequence in approximately half

```
mergeSort a =
  if |a| ≤ 1 then
    a
  else
    let
      (l, r) = splitMid a
      (l', r') = (mergeSort l || mergeSort r)
    in
      merge(l', r')
    end
```

Just simply split?

How to merge?

  - Base Case ?
  - Inductive step: divide recurse combine ?

MERGE SORT

- **Analysis**
  - **Can you prove the correctness?**
  - **W & S?**

```
mergeSort a =
  if |a| ≤ 1 then
    a
  else
    let
      (l, r) = splitMid a
      (l', r') = (mergeSort l || mergeSort r)
    in
      merge(l', r')
    end
```

$$W(n) = W_{\text{divide}}(n) + \sum_{i=1}^{k} W(n_i) + W_{\text{combine}}(n) + 1$$

=?

$$S(n) = S_{\text{divide}}(n) + \max_{i=1}^{k} S(n_i) + S_{\text{combine}}(n) + 1$$

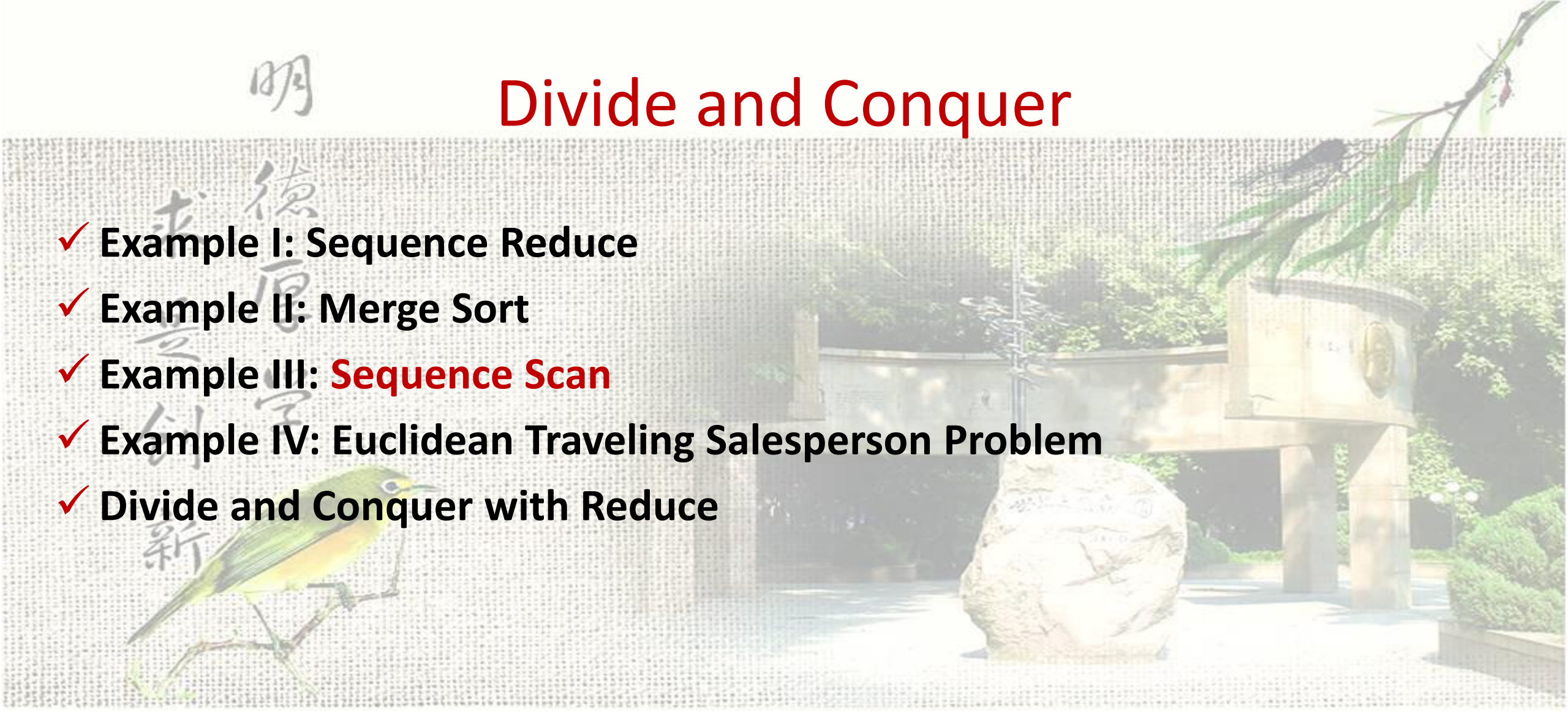
=?

MERGE SORT

- **Mergesort**
  - **W & S?**
    - ✓ **Recurrences?**
    - ✓ **W(n) = 2W(n/2) +O(n)** — **how?**
    - ✓ **S(n) = max(S(n/2), S(n/2))+O(log n) = S(n/2) + O(log n)** — **how?**
    - ✓ **W=?**
    - ✓ **Θ (n log n)** — **Why?**
    - ✓ **S=?**
    - ✓ **$O(\log^2 n)$**

Divide and Conquer

- ✓ Example I: Sequence Reduce
- ✓ Example II: Merge Sort
- ✓ Example III: Sequence Scan
- ✓ Example IV: Euclidean Traveling Salesperson Problem
- ✓ Divide and Conquer with Reduce



SEQUENCE SCAN

- How can we divide?
- How can we put the results together?
  - <2, 1, 3, 2, 2, 5, 4, 1>
  - ((Sl, tl) = (<0, 2, 3, 6>, 8) and (Sr, tr) = (< 0, 2, 7, 11>, 12)
  - Base Case ?
  - Inductive step: divide recurse combine ?

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

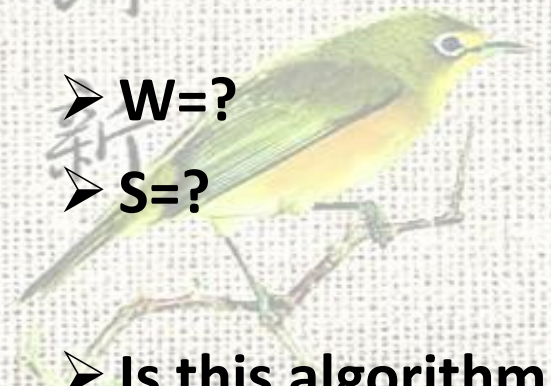
SEQUENCE SCAN

- **D & C for sequence Scan**

```
scanDC f id a =
  if |a| = 0 then
    (⟨ ⟩, id)
  else if |a| = 1 then
    (⟨ id ⟩, a[0])
  else
    let
      (b, c) = splitMid a
      ((l, b'), (r, c')) = (scanDC f id b || scanDC f id c)
      r' = ⟨ f(b', x) : x ∈ r ⟩
    in
      (append (l, r'), f(b', c'))
    end
```

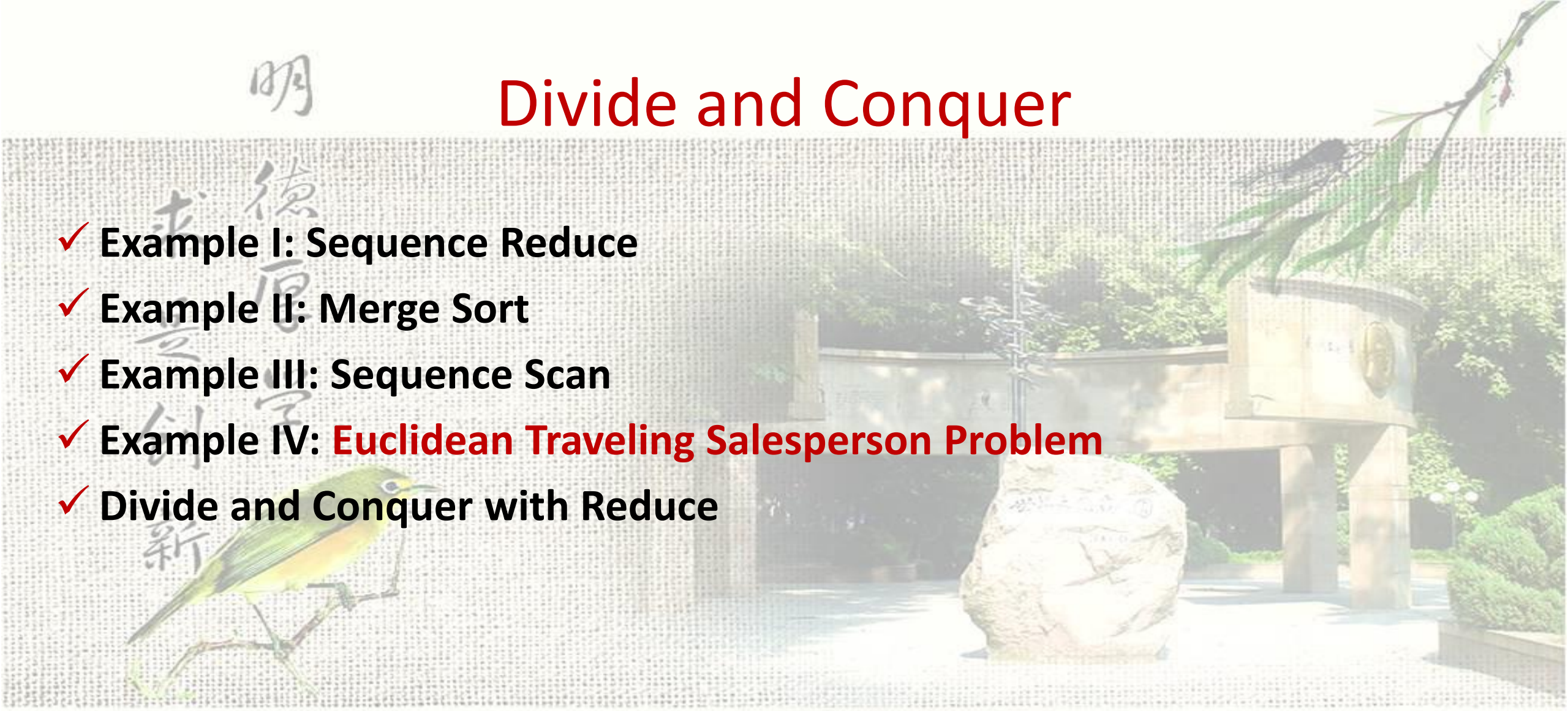
SEQUENCE SCAN

- the *work* and *span* for the algorithm
  - W(n) = 2W(n/2) + O(n) — how?
  - S(n) = S(n/2) + O(1) — how?
  - W=?
  - S=?
  - Is this algorithm the best?



Divide and Conquer

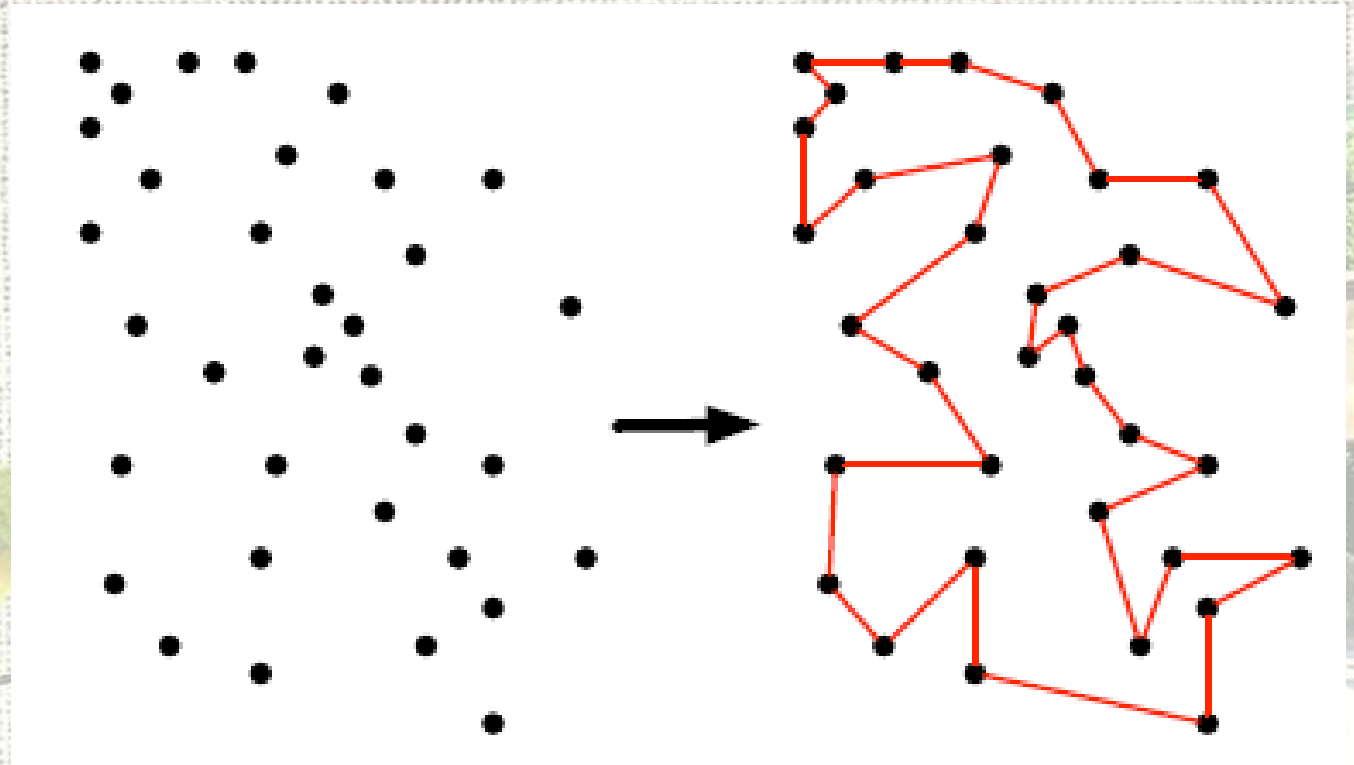
- ✓ Example I: Sequence Reduce
- ✓ Example II: Merge Sort
- ✓ Example III: Sequence Scan
- ✓ Example IV: Euclidean Traveling Salesperson Problem
- ✓ Divide and Conquer with Reduce



THE EUCLIDIAN TSP

- Given a set of points in a n-dimensional Euclidian space
  - What is a Euclidian space?
- Find the shortest Hamiltonian cycle
  - What is a Hamiltonian cycle?
- We get a planar Euclidian Traveling Salesperson Problem when the points are in 2-dimensional space

THE PLANAR TSP

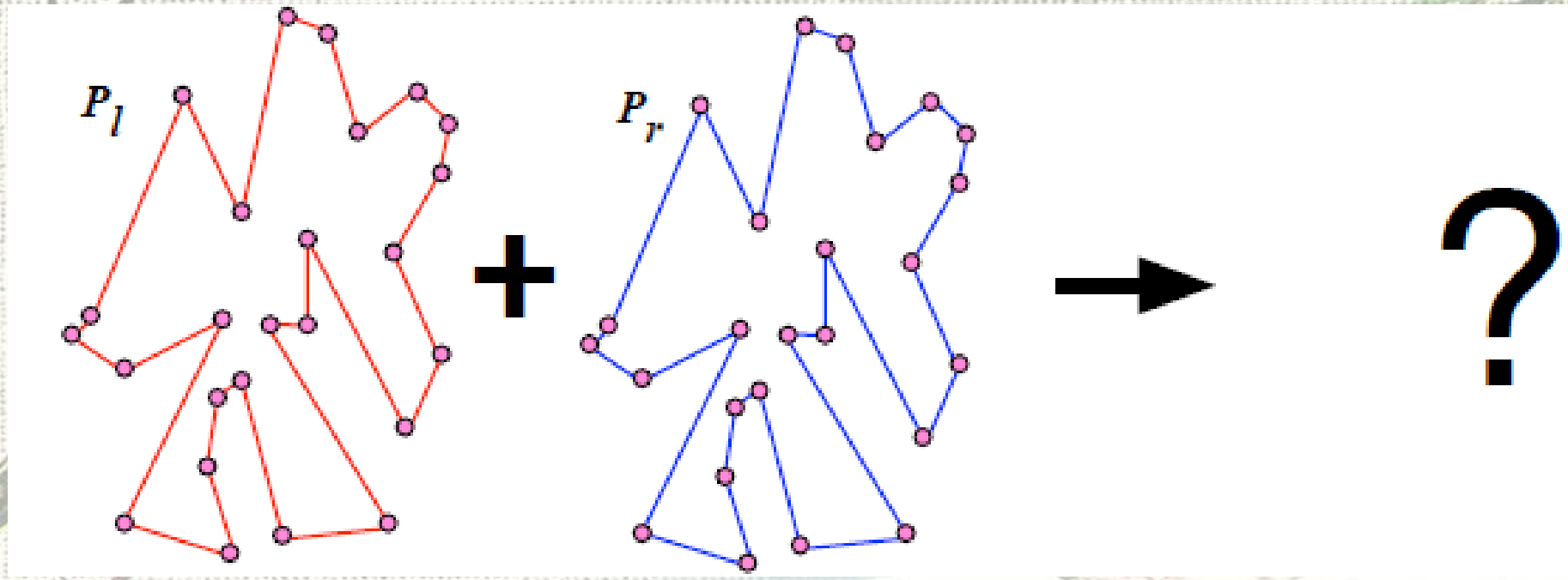


A Divide-and-Conquer Heuristic

- What is a heuristic?
- Approximation algorithm
  - ❿ Resulting tour length is guaranteed to be close to the actual minimum tour length
  - ❿ If you spend enough work (but polynomial)
- The Divide-and-Conquer does work both before and after the recursive calls

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

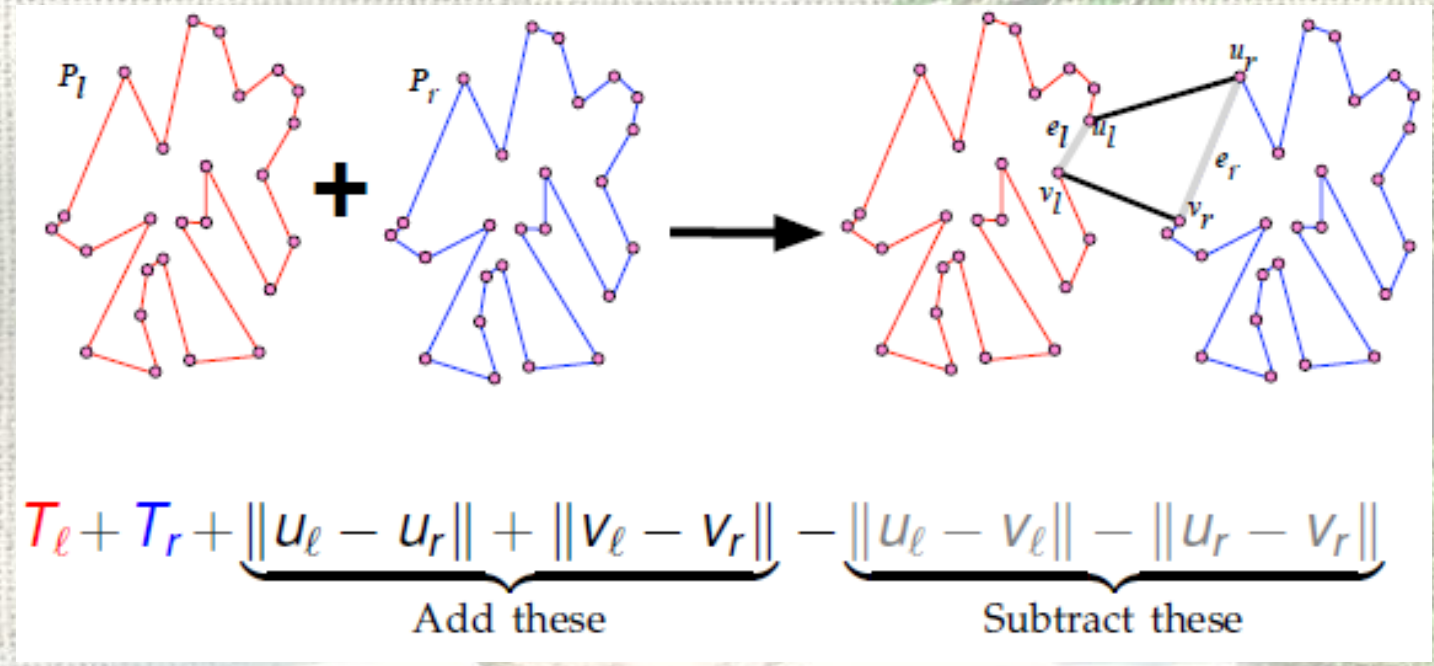
A Divide-and-Conquer Heuristic



- Assume P_l and P_r have tour lengths T_l and T_r
- Tour length for the combination?

A Divide-and-Conquer Heuristic

- Try all pairs of edges e_l from P_l and e_r from P_r
  - How many pairs are there?
- For each pair of edges, find the smallest increase
- Then combine the small tours into a large tour



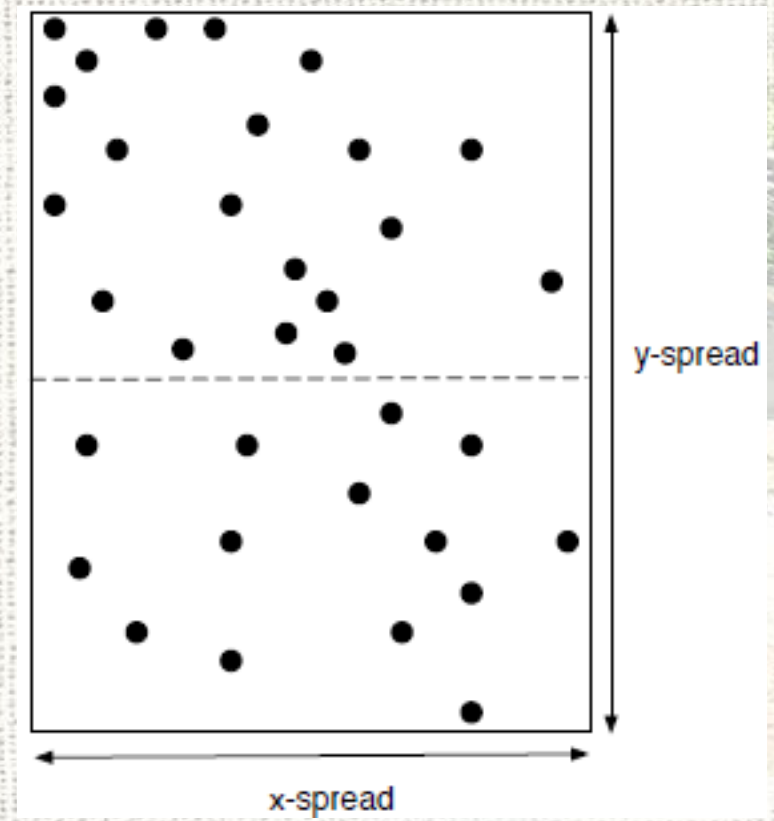
A Divide-and-Conquer Heuristic

$$
\begin{aligned}
&eTSP\ (P) = \\
&\quad \texttt{if } |P| < 2 \texttt{ then} \\
&\qquad \texttt{raise } \mathit{TooSmall} \\
&\quad \texttt{else if } |P| = 2 \texttt{ then} \\
&\qquad \langle\ (P[0], P[1]), (P[1], P[0])\ \rangle \\
&\quad \texttt{else} \\
&\qquad \texttt{let} \\
&\qquad\quad (P_\ell, P_r) = \mathit{split}\ P \texttt{ along the longest dimension} \\
&\qquad\quad (L, R) = (eTSP\ P_\ell)\ ||\ (eTSP\ P_r) \\
&\qquad\quad (c, (e, e')) = \mathit{minVal}_{\mathit{first}}\ \{(\mathit{swapCost}(e, e'), (e, e')) : e \in L, e' \in R\} \\
&\qquad \texttt{in} \\
&\qquad\quad \mathit{swapEdges}\ (\mathit{append}\ (L, R), e, e') \\
&\qquad \texttt{end}
\end{aligned}
$$

how?

how?

SPLITTING THE POINTS



- **Split at the median along the longer spread dimension**

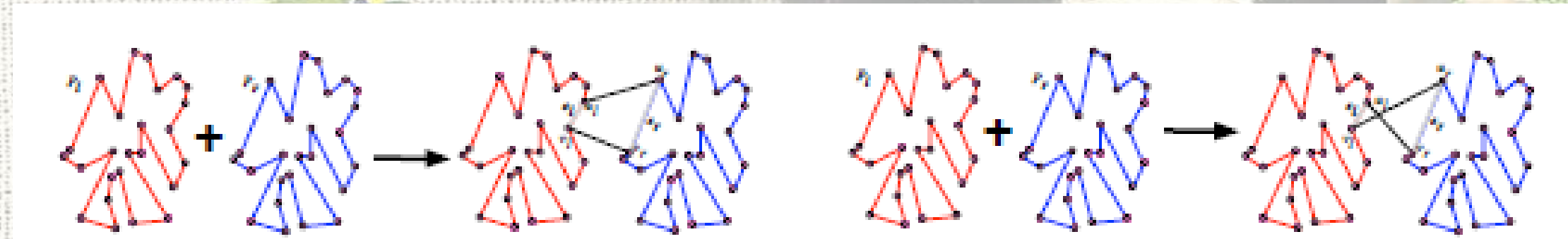
SWAP COST

- Given el = $(u_l, v_l) \in L$ and er = $(u_r, v_r) \in R$

  swapCost((u_l, v_l), (u_r, v_r)) = Cost Added−Cost Removed

  Cost Added = $\min(||u_l - u_r|| + ||v_l - v_r||, ||u_l - v_r|| + ||v_l - u_r||)$

  Cost Removed = $||u_l - v_l|| + ||u_r - v_r||$



SWAPPING EDGES

- swapEdges(append(L,R),e'_l,e'_r)
  - Appends the Tour edge lists from subproblems
  - Then removes and adds appropriate edges

COST ANALYSIS

- $W(n) = 2W(n/2) + O(n^2)$ — how?
- $S(n) = S(n/2) + O(\log n)$ — how?
- S(n)=? , W(n)=?

$$
\begin{aligned}
&eTSP\ (P) = \\
&\quad \texttt{if } |P| < 2 \texttt{ then} \\
&\qquad \texttt{raise } TooSmall \\
&\quad \texttt{else if } |P| = 2 \texttt{ then} \\
&\qquad \langle\, (P[0], P[1]), (P[1], P[0]) \,\rangle \\
&\quad \texttt{else} \\
&\qquad \texttt{let} \\
&\qquad\quad (P_\ell, P_r) = split\ P \texttt{ along the longest dimension} \\
&\qquad\quad (L, R) = (eTSP\ P_\ell) \parallel (eTSP\ P_r) \\
&\qquad\quad (c, (e, e')) = minVal_{first} \left\{ (swapCost(e, e'), (e, e')) : e \in L, e' \in R \right\} \\
&\qquad \texttt{in} \\
&\qquad\quad\ \ swapEdges\ (append\ (L, R), e, e') \\
&\qquad \texttt{end}
\end{aligned}
$$

Cost Analysis

- Solve (directly)

  $W(n) = 2W(n/2) + k \cdot n^{1+\varepsilon}$

  for constant $\varepsilon > 0$.

  - Depth is $\log_2 n$ (Is this technically right?)
  - At level i, we have 2^i nodes each costing $k \cdot (n/2^i)^{1+\varepsilon}$

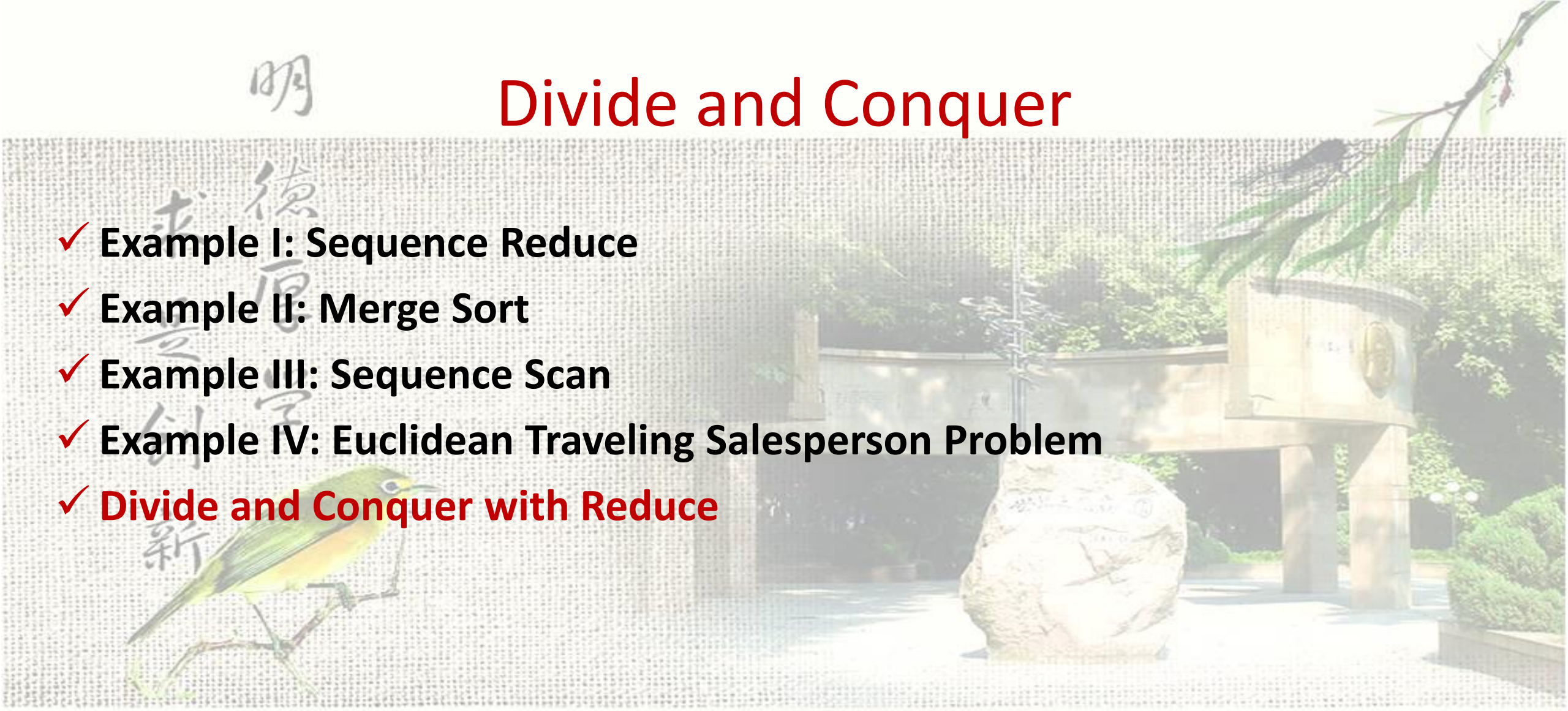
$$W(n) = \sum_{i=0}^{\log n} k \cdot 2^i \cdot (n/2^i)^{1+\varepsilon} = k \cdot n^{1+\varepsilon} \cdot \sum_{i=0}^{\log n} 2^{-i-\varepsilon}$$

$$\leq k \cdot n^{1+\varepsilon} \cdot \sum_{i=0}^{\infty} 2^{-i-\varepsilon}$$

$W(n) \in O(n^{1+\varepsilon})$

Divide and Conquer

- ✓ Example I: Sequence Reduce
- ✓ Example II: Merge Sort
- ✓ Example III: Sequence Scan
- ✓ Example IV: Euclidean Traveling Salesperson Problem
- ✓ Divide and Conquer with Reduce



STRENGTHENING

- strengthen the problem definition
  - ➢ defining a problem that solves more than what you ultimately need, but makes it easier or even possible to use solutions of sub-problems to solve the larger problem
  - ➢ Any instance before?
    - ✓ solve the *Parenthesis Matching problem* by defining a version of the problem that returns the number of unmatched parentheses on the right and left

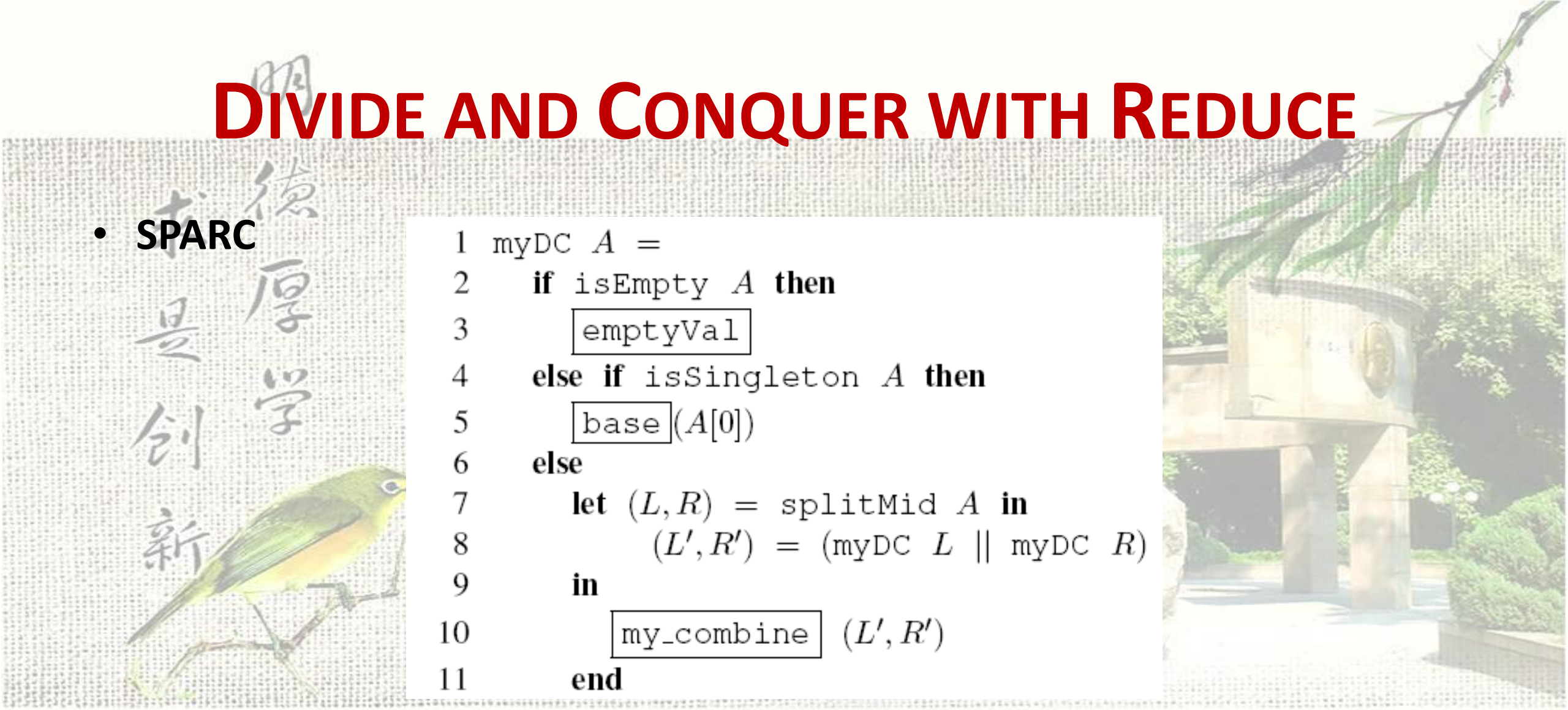
DIVIDE AND CONQUER WITH REDUCE

- **Inductive Step**
  - 1. **Divide** I into some number of smaller instances of the same problem P.
  - 2. **Recurse** on each of the smaller instances to obtain their answers.
  - 3. **Combine** the answers to produce an answer for the original instance I.

DIVIDE AND CONQUER WITH REDUCE

- **SPARC**

```
1  myDC A =
2    if isEmpty A then
3      emptyVal
4    else if isSingleton A then
5      base (A[0])
6    else
7      let (L, R) = splitMid A in
8          (L', R') = (myDC L || myDC R)
9      in
10       my_combine (L', R')
11     end
```



DIVIDE AND CONQUER WITH REDUCE

- **Sequence opertation?**

reduce my_combine emptyVal (map $x \in S$ (base x))

reduce my_combine emptyVal (map base S)

Divide and Conquer

- ✓ Divided-&-conquer steps?
- ✓ How to use d-&-c steps to solve problems?
  - ✓ Sequence reduce, sequence scan, mergesort, TSP……
- ✓ How to get W & S?

SYNOPSIS

- Basic Techniques
- Divide and Conquer
- Contraction
- Maximum Contiguous Subsequence Sum

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Structure of Contraction Algorithm

- **Base Case**
  - If the problem instance is sufficiently small, then compute and return the answer, possibly using another algorithm
- **Inductive Step**
  - 1. Contract
    - "contract", i.e., map the instance of the problem P to a smaller instance of P
  - 2. Solve
    - solve the smaller instance recursively
  - 3. Expand
    - use the solution to solve the original instance

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

ADVANTAGE OF CONTRACTION

- establishing correctness and efficiency properties in a relatively straightforward fashion using principles of induction
- can be efficient
  - if the contraction and expansion steps have low spans
  - if the problem size can be decreased by geometrically
  - the algorithm will likely have a low span, making it a good parallel algorithm

Contraction

- ✓ Implementing Reduce with Contraction
- ✓ Implementing Scan with Contraction

REDUCE WITH CONTRACT

$$reduce\ (f : \alpha * \alpha \to \alpha)\ (id : \alpha)\ (a : \mathbb{S}_\alpha) : \alpha,$$

- *f* is an associative function, *a* is the sequence, and *id* is the identity element of *f*

- **Constrcut**
  - How can we reduce an instance of the "reduce" problem to a smaller instance of the same problem?
- **Solve**
  - solve the smaller instance recursively
  - Can we perform this contraction step in parallel?
- **Expand**
  - What computations should the expansion step perform?
- **What is the work and span of this algorithm?**

REDUCE WITH CONTRACT

- **ex. compute the sum of the input sequence**
  <2, 1, 3, 2, 2, 5, 4, 1>
- **Can you get an answer?**

```
(* Assumption: |a| is a power of 2 *)
reduceContract f id a =
  if |a| = 1 then
    a[0]
  else
    let
      b = ⟨ f(a[2i], a[2i + 1]) : 0 ≤ i < ⌊|a|/2⌋ ⟩
    in
      reduceContract f id b
    end
```

REDUCE WITH CONTRACT

- Can perform in parallel?
- What the expansion step?
- W=?
  - ✓ W(n) = W(n/2) + O(n), W(n)=?
- S=?
  - ✓ S(n) = S(n/2) + O(1), S(n)=?

How to prove?

```
(* Assumption: |a| is a power of 2 *)
reduceContract f id a =
  if |a| = 1 then
    a[0]
  else
    let
      b = ⟨ f(a[2i], a[2i + 1]) : 0 ≤ i < ⌊|a|/2⌋ ⟩
    in
      reduceContract f id b
    end
```

Contraction

- ✓ Implementing Reduce with Contraction
- ✓ Implementing Scan with Contraction

Scan WITH CONTRACT

$$scan\ (f : \alpha * \alpha \rightarrow \alpha)\ (id : \alpha)\ (a : \mathbb{S}_\alpha)\ :\ (\mathbb{S}_\alpha * \alpha)$$

- *f is an associative function, a is the sequence, and id is the identity element of f*
- *E.g.* scan + 0 <2, 1, 3, 2, 2, 5, 4, 1>
- (< 0, 2, 3, 6, 8, 10, 15, 19>, 20)

- **scan looks inherently sequential**
  - **Naive implementation needs *O(n²)* work.**
  - **Slightly clever sequential implementation needs *O(n)* work**
    - ✓**Any problem with this algorithm?**
    - ✓**Then?**

why?

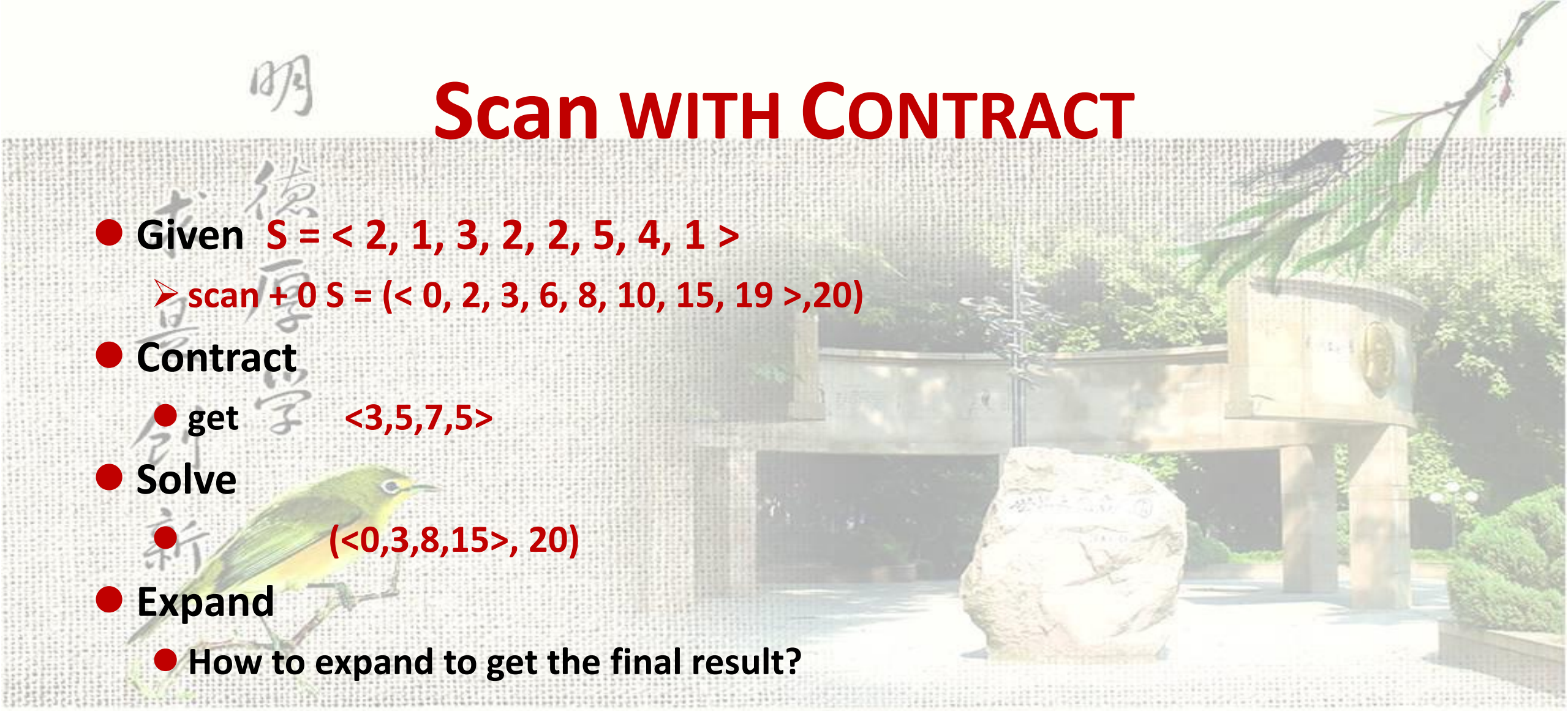
how?

Scan WITH CONTRACT

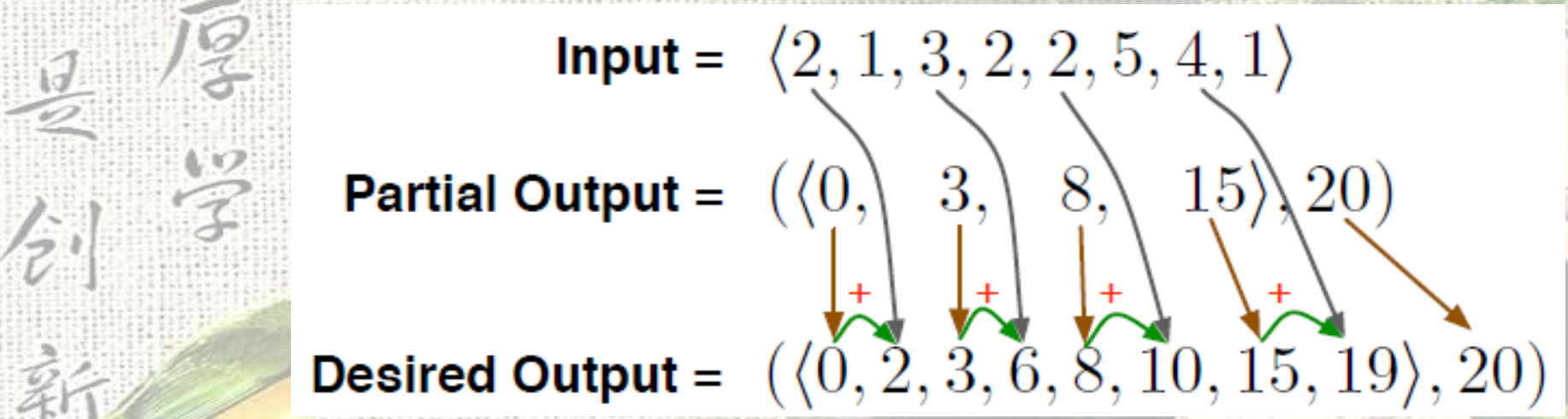
- Contract
  - Construct a much smaller instance of the problem
- Solve
  - Solve the smaller instance recursively
  - Perform the step (can in parallel?)
- Expand
- What is the work and span of this algorithm?

Scan WITH CONTRACT

- Given S = < 2, 1, 3, 2, 2, 5, 4, 1 >
  - scan + 0 S = (< 0, 2, 3, 6, 8, 10, 15, 19 >,20)
- Contract
  - get <3,5,7,5>
- Solve
  - (<0,3,8,15>, 20)
- Expand
  - How to expand to get the final result?



Scan WITH CONTRACT



Scan WITH CONTRACT

- W=?
  - W(n) = W(n/2) + O(n), W=?
- S=?
  - S(n) = S(n/2) + O(1), S=?

(* Assumption: $|a|$ is a power of two. *)

$scan\ f\ id\ a =$

if $|a| = 0$ then

$(\langle\ \rangle, id)$

else if $|a| = 1$ then

$(\langle\ id\ \rangle, a[0])$

else

let

$a' = \langle\ f(a[2i], a[2i+1]) : 0 \le i < n/2\ \rangle$

$(r, t) = scan\ f\ id\ a'$

in

$(\langle\ p_i : 0 \le i < n\ \rangle, t)$, where $p_i = \begin{cases} r[i/2] & \text{even}(i) \\ f(r[i/2], a[i-1]) & \text{otherwise} \end{cases}$

end

Contraction

- ✓ Contraction steps?
- ✓ How to use contraction steps to implement Reduce & Scan?

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

SYNOPSIS

- Basic Techniques
- Divide and Conquer
- Contraction
- Maximum Contiguous Subsequence Sum

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Maximum Contiguous Subsequence Sum

- ✓ **The Problem**
- ✓ **Brute Force**
- ✓ **Applying Reduction**
- ✓ **Divide And Conquer**

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEFINITION

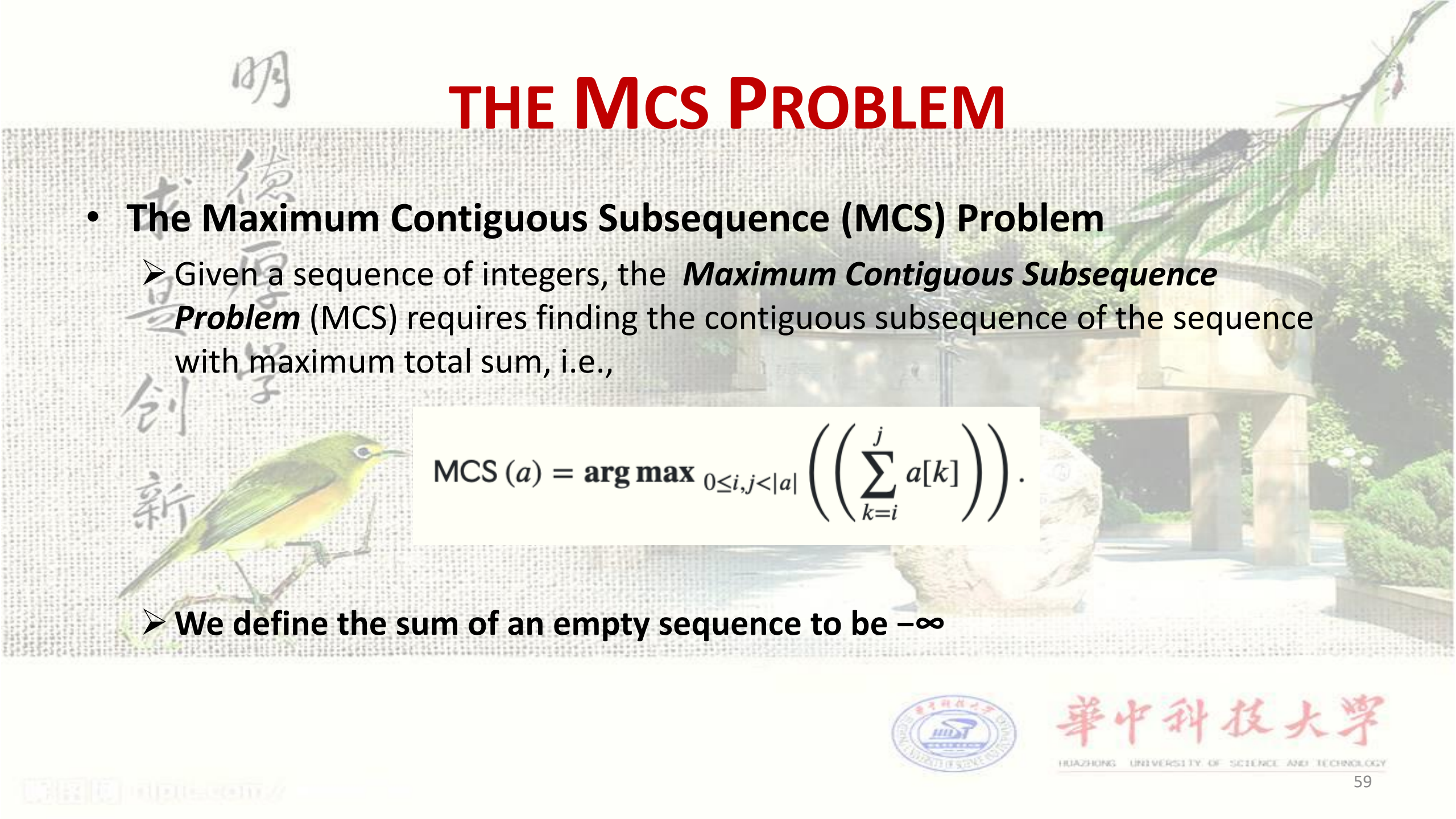
- **Subsequence**
  - A subsequence b of a sequence a is a sequence that can be derived from a by deleting zero or more elements of a without changing the order of remaining elements.
- **Contiguous subsequence**
  - For any sequence a of n elements, the subsequence $b = a[i \ldots\ldots j], 0 \le i \le j < n$, which consists of the elements at positions i,i+1,……,j is a contiguous subsequence of a
  - $S = \langle 0, -1, 2, -1, 4, -1, 0 \rangle$

THE MCS PROBLEM

- **The Maximum Contiguous Subsequence (MCS) Problem**
  - Given a sequence of integers, the ***Maximum Contiguous Subsequence Problem*** (MCS) requires finding the contiguous subsequence of the sequence with maximum total sum, i.e.,

$$\text{MCS}(a) = \arg\max_{0 \le i,j < |a|} \left(\left(\sum_{k=i}^{j} a[k] \right) \right).$$

  - **We define the sum of an empty sequence to be $-\infty$**



THE MCSS PROBLEM

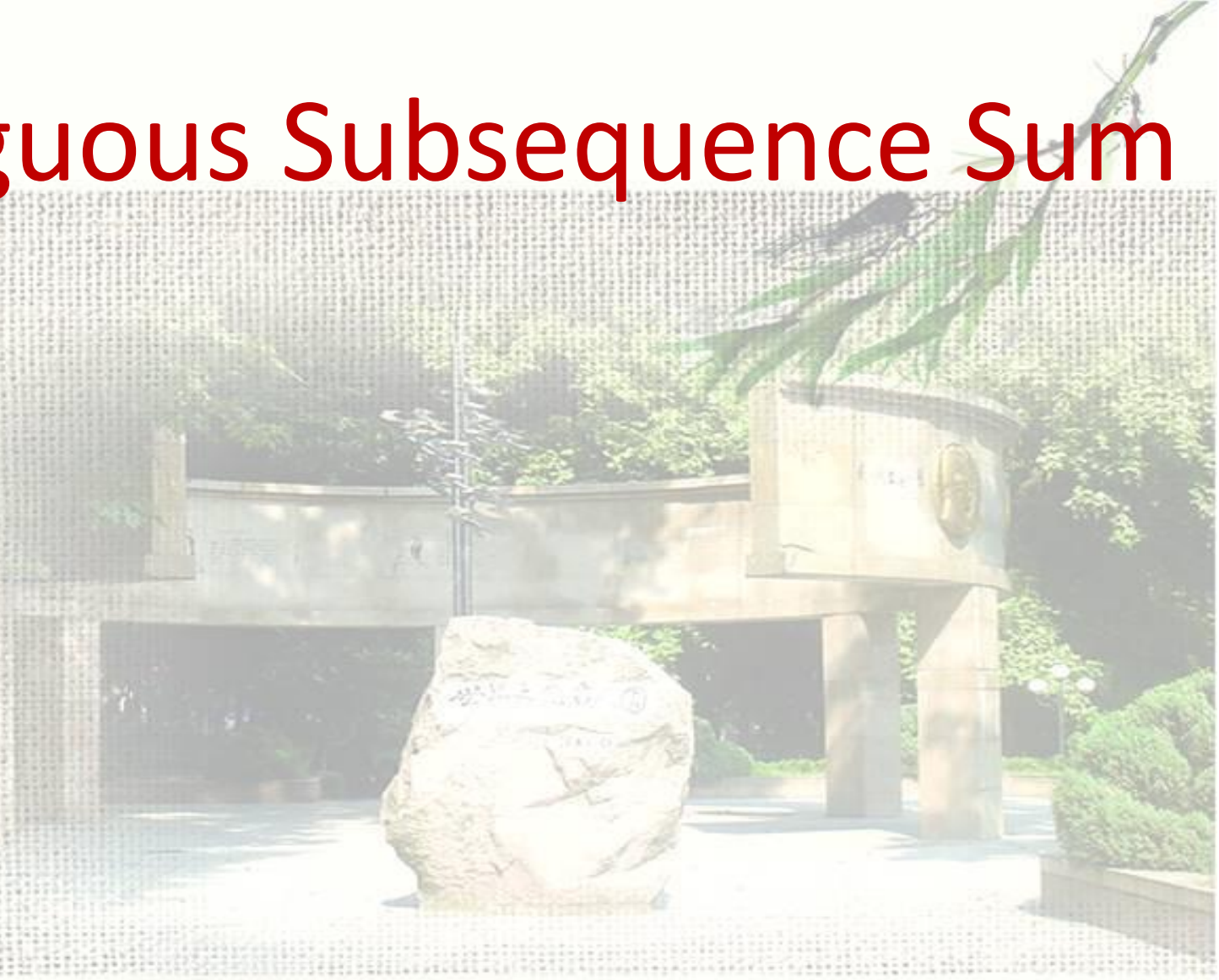
- **The Maximum Contiguous-Subsequence-Sum (MCSS) Problem**
  - **Given a sequence of numbers, the maximum contiguous-subsequence-sum problem is to find**

$$\text{MCSS}(a) = \max\left\{\sum_{k=i}^{j} a[k] \ : \ 0 \le i, j < |a|\right\}.$$

  - **$S = \langle 0, -1, 2, -1, 4, -1, 0\rangle$, MCSS (S) = ?**
  - **How many sumpossible subsequences are there?**

Maximum Contiguous Subsequence Sum

- ✓ The Problem
- ✓ Brute Force
- ✓ Applying Reduction
- ✓ Divide And Conquer



華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

NATURAL CHOICE

- **Reducing MCSS problem to MCS (maximum-contiguous-subsequence) problem**
  - **MCS: finding the subsequence itself**
- **how we can solve the MCSS problem by reducing it to the maximum-contiguous-subsequence MCS problem?**
  - **subsequences can be represented by a pair of integers (i, j), $0 \le i \le j < n$, we can generate all such integer pairs**
  - **compute the sum for each sequence**
  - **pick the largest**

How many?

NATURAL CHOICE

- **MCS (maximum-contiguous-subsequence) problem**

$$
\begin{array}{l}
MCSBF\ a = \\
\quad \texttt{let} \\
\quad\quad maxSum\ ((i,j,s),(k,\ell,t)) = \texttt{if}\ s > t\ \texttt{then}\ (i,j,s)\ \texttt{else}\ (k,\ell,t) \\
\quad\quad b = \langle\ (i,j,reduce\ +\ 0\ a[i \cdots j]) : 0 \le i \le j < n\ \rangle \\
\quad\quad (i,j,s) = reduce\ maxSum\ (-1,-1,-\infty)\ b \\
\quad \texttt{in} \\
\quad\quad (i,j) \\
\quad \texttt{end}
\end{array}
$$

BRUTE FORCE ALGORITHM

- Compute the sum of all $O(n^2)$ possible sub-sequences (in parallel)
  - Use plus reduce
- Compute maximum over all $O(n^2)$ sums
  - Use max reduce

```
MCSSBF a =
  let
    (i, j) = MCSBF a
    sum = reduce '+' 0 a[i ··· j]
  in
    sum
  end.
```

BRUTE FORCE ALGORITHM

- The brute force algorithm has some redundancy
  - to find the solution, it computes the result for the MCS problem and then computes the sum of the result sequence, which is already computed by the MCS algorithm.

```
MCSSBF a =
  let
    (i, j) = MCSBF a
    sum = reduce '+' 0 a[i ··· j]
  in
    sum
  end.
```

$$
\begin{aligned}
&MCSSBF\ a = \\
&\quad \text{let} \\
&\qquad b = \langle\, reduce\ +\ 0\ a[i \cdots j] : 0 \le i \le j < n \,\rangle \\
&\quad \text{in} \\
&\qquad reduce\ max\ -\infty\ b \\
&\quad \text{end}
\end{aligned}
$$

ALGORITHM ANALYSIS

- Sum of subsequence (i, j) needs
  - O(j – i) work (Why?)
  - O(log(j – i)) span (Why?)
- Compute maximum over all $O(n^2)$ sums
  - Total costs for brute force are

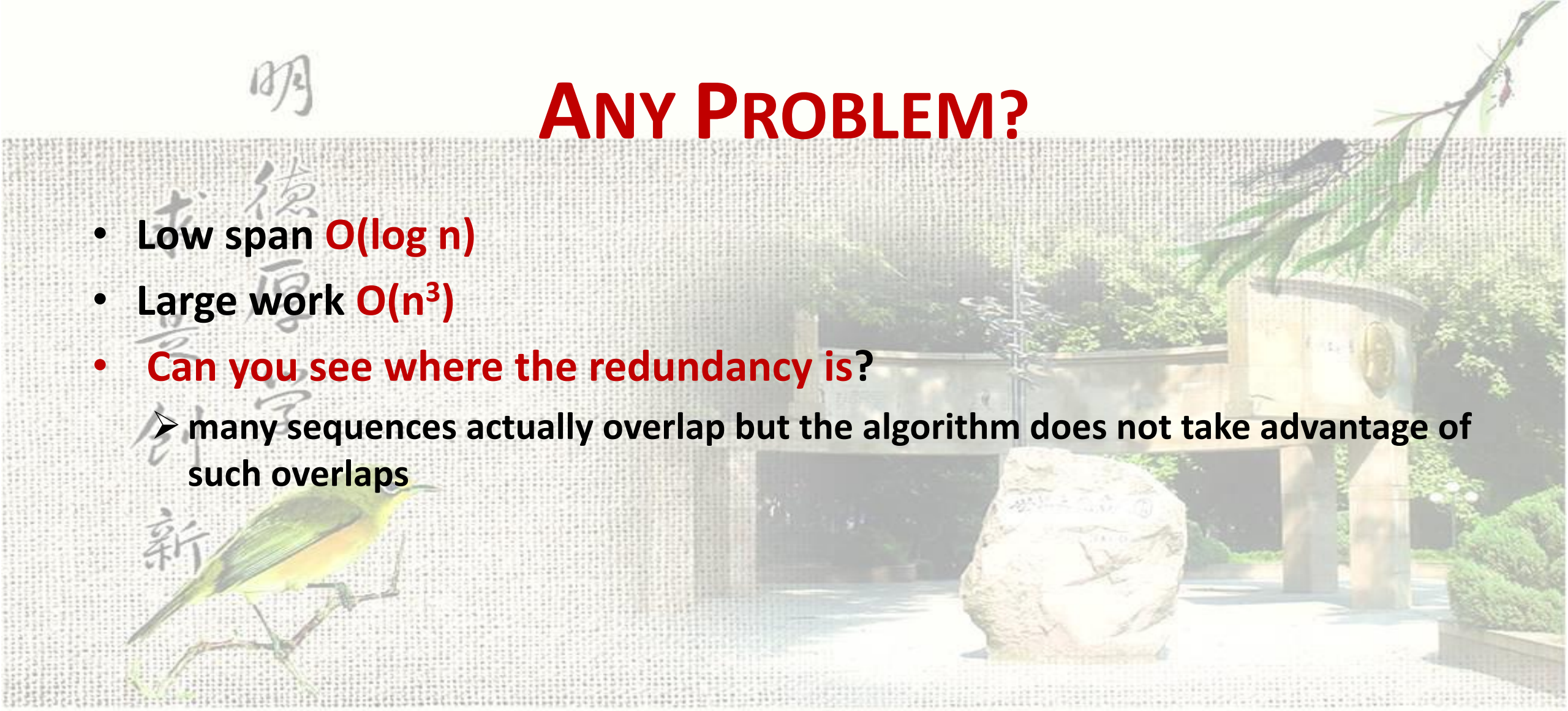
$$W(n) = 1 + \sum_{1 \le i \le j \le n} W_{reduce}(j-i) \le 1 + n^2 \cdot W_{reduce}(n)$$

$$= 1 + n^2 \cdot O(n) \in O(n^3)$$

$$S(n) = 1 + \max_{1 \le i \le j \le n} S_{reduce}(j-i) \le 1 + S_{reduce}(n) \in O(\log n)$$

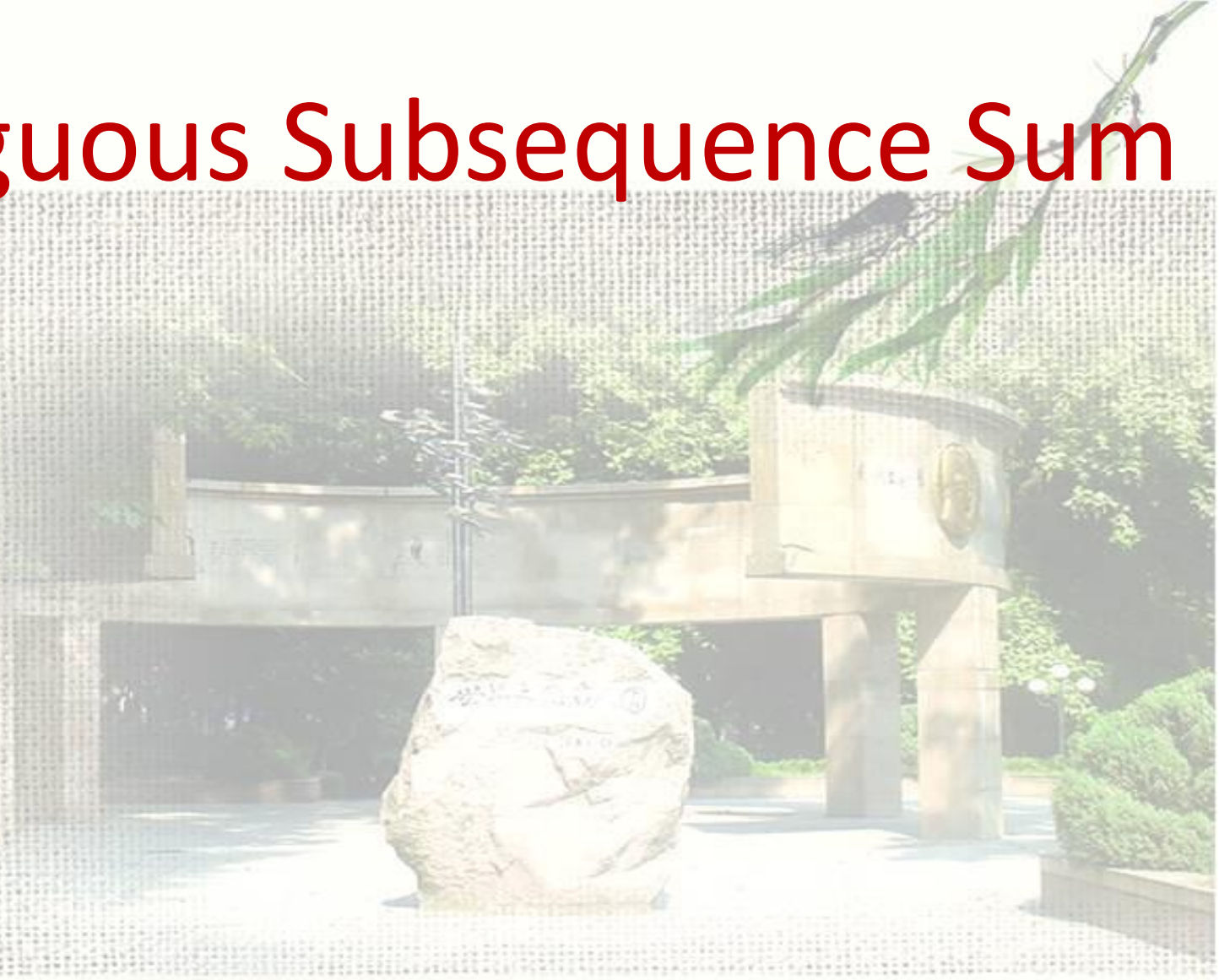
ANY PROBLEM?

- Low span $O(\log n)$
- Large work $O(n^3)$
- Can you see where the redundancy is?
  - many sequences actually overlap but the algorithm does not take advantage of such overlaps



Maximum Contiguous Subsequence Sum

- ✓ The Problem
- ✓ Brute Force
- ✓ Applying Reduction
- ✓ Divide And Conquer



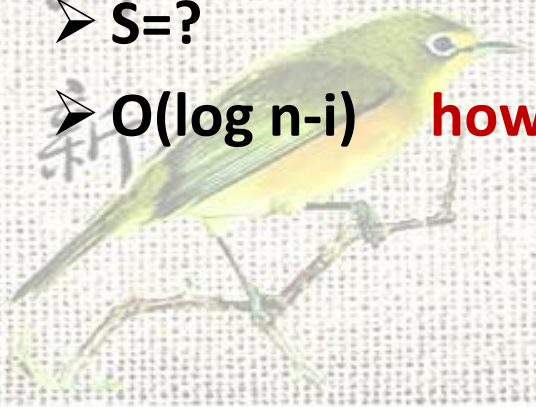
MCSSS AND MCSSE PROBLEM

- **MCSSS**
  - **The Maximum Contiguous Subsequence Sum with Start**
  - **The MCSSS problem requires finding the maximum contiguous subsequence of a sequence that starts at a given position**
- **MCSSE**
  - **the Maximum-Contiguous-Sum-with-Ending problem**
  - **the MCSSE problem requires finding the maximum contiguous subsequence ending at a specified end position.**

MCSSS PROBLEM

- **An Optimal Algorithm for MCSSS**
  - W=?
  - O(n-i) how?
  - S=?
  - O(log n-i) how?

```
MCSSSOpt a i =
  let
    b = scanI' +' 0 a [i ··· (|a| − 1)]
  in
    reduce max −∞ b
  end
```



MCSSE PROBLEM

- **An Optimal Algorithm for MCSSE**
  - W=?
  - O(j) how?
  - S=?
  - O(log j) how?

```
MCSSEOpt a j =
  let
    (b, v) = scan '+' 0 a[0 ··· j]
    w = reduce min ∞ b
  in
    v − w
  end
```

REDUCE MCSS TO MCSSS

- **Reduce MCSS to MCSSS**
  - **try all possible start positions, solve the MCSSS problem for each**
  - **pick the maximum of all the solutions**

$$MCSSReducedForce\ a = reduce\ max\ -\infty\ \langle\ (MCSSSOpt\ a\ i) : 0 \leq i < n \rangle .$$

  - **W=?**
  - **O(n^2)** **how?**
  - **S=?**
  - **O(logn)** **how?**

PROBLEM

- The two algorithms obtained by reduction to MCSSS and reduction to MCSSE both reduce some of the redundant work, but not all, because they don't reduce redundancies when solving for subsequences ending (or starting) at different positions

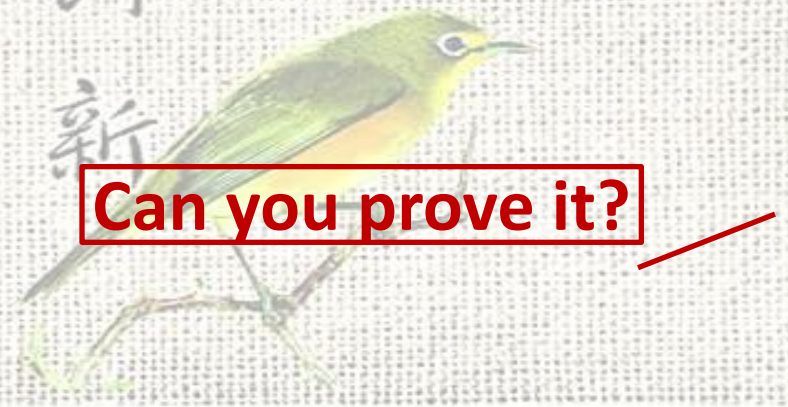
MCSSE EXTENSION

- If we are given the maximum contiguous sequence, Mi ending at position i.
  - We can compute the maximum contiguous sequence ending at position i + 1, Mi+1, from this by noticing that
    Mi+1 = Mi + <A[i]>, or Mi+1 = <A[i]>,
    depending on the sum for each.

Can prove?

MCSS with Iteration

- solve MCSS problem using iteration
  - we can iterate over the sequence and solve the MCSS problem for each ending position and take the maximum over all positions
- MCSS with iteration



Can you prove it?

```
MCSSIterative a =
  let
    f (sum, x) =
      if sum + x ≥ x then
        sum + x
      else
        x
    b = iteratePrefixes f −∞ a
  in
    reduce max −∞ b
  end
```

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

REDUCE MCS2 TO MCS2E

- W(iteratePrefixes)=?
- S((iteratePrefixes)=?
- the algorithm has no parallelism
- IterativeAlgoMCSS: W=? S=?

Why?

```
MCSSIterative a =
  let
    f (sum, x) =
      if sum + x ≥ x then
        sum + x
      else
        x
    b = iteratePrefixes f −∞ a
  in
    reduce max −∞ b
  end
```

KEY OBSERVATION

- any contiguous subsequence of the original sequence can be expressed in terms of the difference between two prefixes

$$\mathcal{R}^{+0}\ A[i,\ldots,j] = \mathcal{R}^{+0}\ A[0,\ldots,j] - \mathcal{R}^{+0}\ A[0,\ldots,i-1]$$

- **MCSSE problem**
  - How to use MCSSE solve MCSS?
  - the maximum sequence ending at position j: which has the minimum of all prefixes up to j

How to use scan to get this?

SCAN-BASED MCSS

```
MCSSOpt a =
  let
    (b, v) = scan '+' 0 a
    c = append b ⟨v⟩
    (d, _) = scan min ∞ c
    e = ⟨c[i] − d[i] : 0 < i < |a|⟩
  in
    reduce max −∞ e
  end
```

- We can compute the sum for all prefixes in one scan
- We can compute the minimum prefix sum preceeding all positions in one scan.
- E.g. a = <1,-2,3,-1,2,-3>
- a=<1,4,-1,5,-4,0,2>
- a=<-1,4,2,-5,4,0,-2>

Can you get the result?

SCAN-BASED MCSS

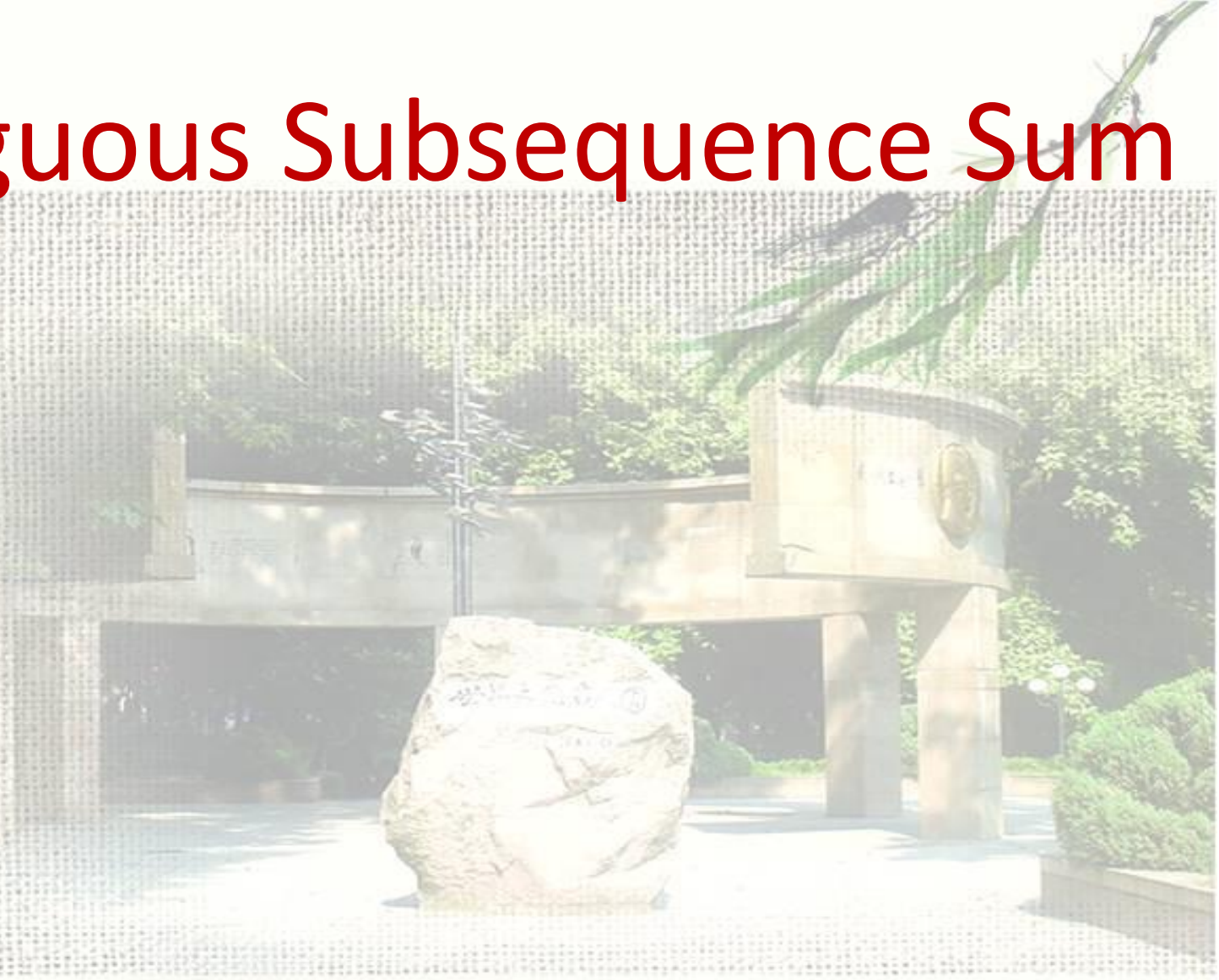
$$
\begin{aligned}
&MCSSOpt\ a = \\
&\quad \text{let} \\
&\qquad (b, v) = scan\ '+'\ 0\ a \\
&\qquad c = append\ b\ \langle v \rangle \\
&\qquad (d, \_) = scan\ min\ \infty\ c \\
&\qquad e = \langle c[i] - d[i] : 0 < i < |a| \rangle \\
&\quad \text{in} \\
&\qquad reduce\ max\ -\infty\ e \\
&\quad \text{end}
\end{aligned}
$$

- W=? S=?

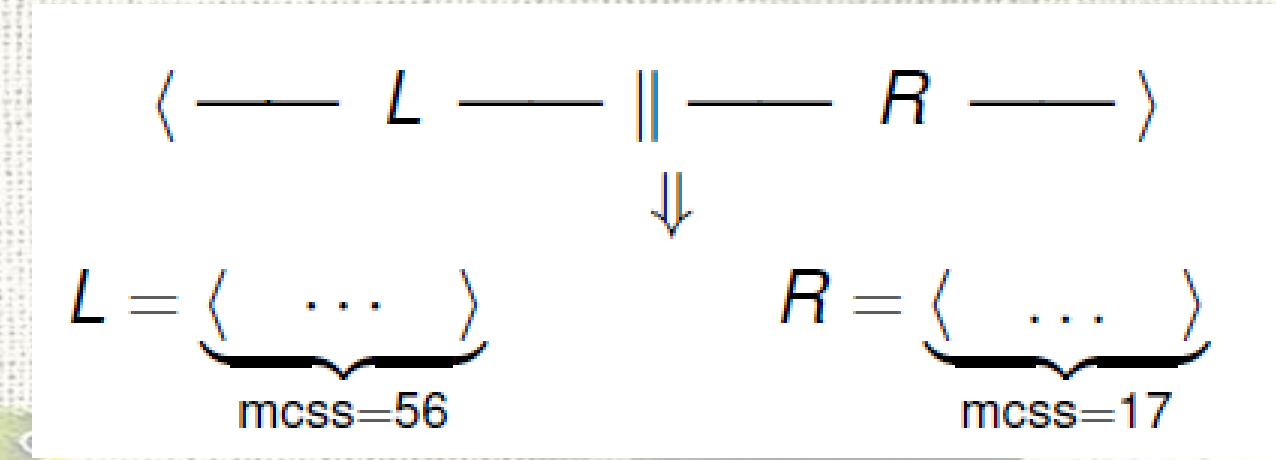
how?

Maximum Contiguous Subsequence Sum

- ✓ The Problem
- ✓ Brute Force
- ✓ Applying Reduction
- ✓ Divide And Conquer



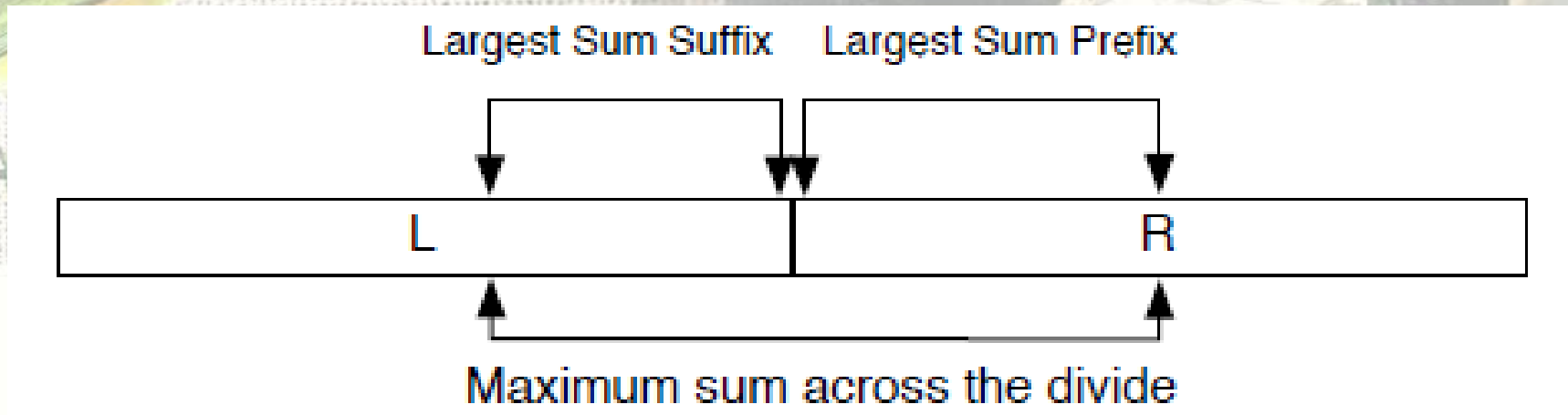
DIVIDE-AND-CONQUER – I



- Let's solve S = <–2,–1, 2, 3, 2,–2>
- Is this right?
- How do we combine subproblem results?

DIVIDE-AND-CONQUER – I

- Recursion handles
  - When mcss(S) subsequence is in the left
  - When mcss(S) subsequence is in the right
- What happens when mcss(S) spans across the divide point?



DIVIDE-AND-CONQUER – I

```
MCSSDC a =
  if |a| = 0 then
    −∞
  else if |a| = 1 then
    a[0]
  else
    let
      (b, c) = splitMid a
      (m_b, m_c) = (MCSSDC b || MCSSDC c)
      m_bc = bestAcross (b, c)
    in
      max{m_b, m_c, m_bc}
    end
```

how?

DIVIDE-AND-CONQUER – I

```
MCSSDC a =
  if |a| = 0 then
    −∞
  else if |a| = 1 then
    a[0]
  else
    let
      (b, c) = splitMid a
      (m_b, m_c) = (MCSSDC b || MCSSDC c)
      m_bc = bestAcross (b, c)
    in
      max{m_b, m_c, m_bc}
    end
```

- W(n) = 2W(n/2) + O(n)
  - → W(n) ∈ O(n log n)

how?

- S(n) = S(n/2) + O(log n)
  - → S(n) ∈ O($\log^2$ n)

how?

DIVIDE-AND-CONQUER – I

- **Cost analysis**

$$
\begin{aligned}
W(n) &\leq 2W(n/2) + k \cdot n \\
&\leq 2(\kappa_1 \cdot \tfrac{n}{2}\log(\tfrac{n}{2}) + \kappa_2) + k \cdot n \\
&= \kappa_1 n(\log n - 1) + 2\kappa_2 + k \cdot n \\
&= \kappa_1 n \log n + \kappa_2 + (k \cdot n + \kappa_2 - \kappa_1 \cdot n) \\
&\leq \kappa_1 n \log n + \kappa_2,
\end{aligned}
$$

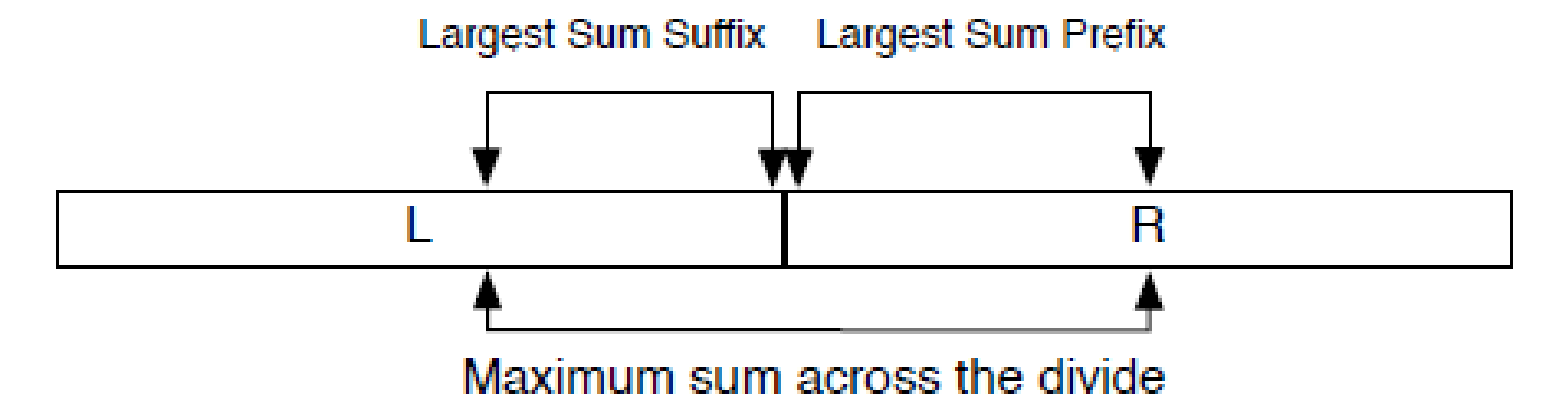
Divide-and-Conquer – II

Important Questions

- Can we do better than O(n log n) work?
- What part of the divide-and-conquer is the bottleneck?
  - Combine takes linear time? (Why?)
- How can we improve?

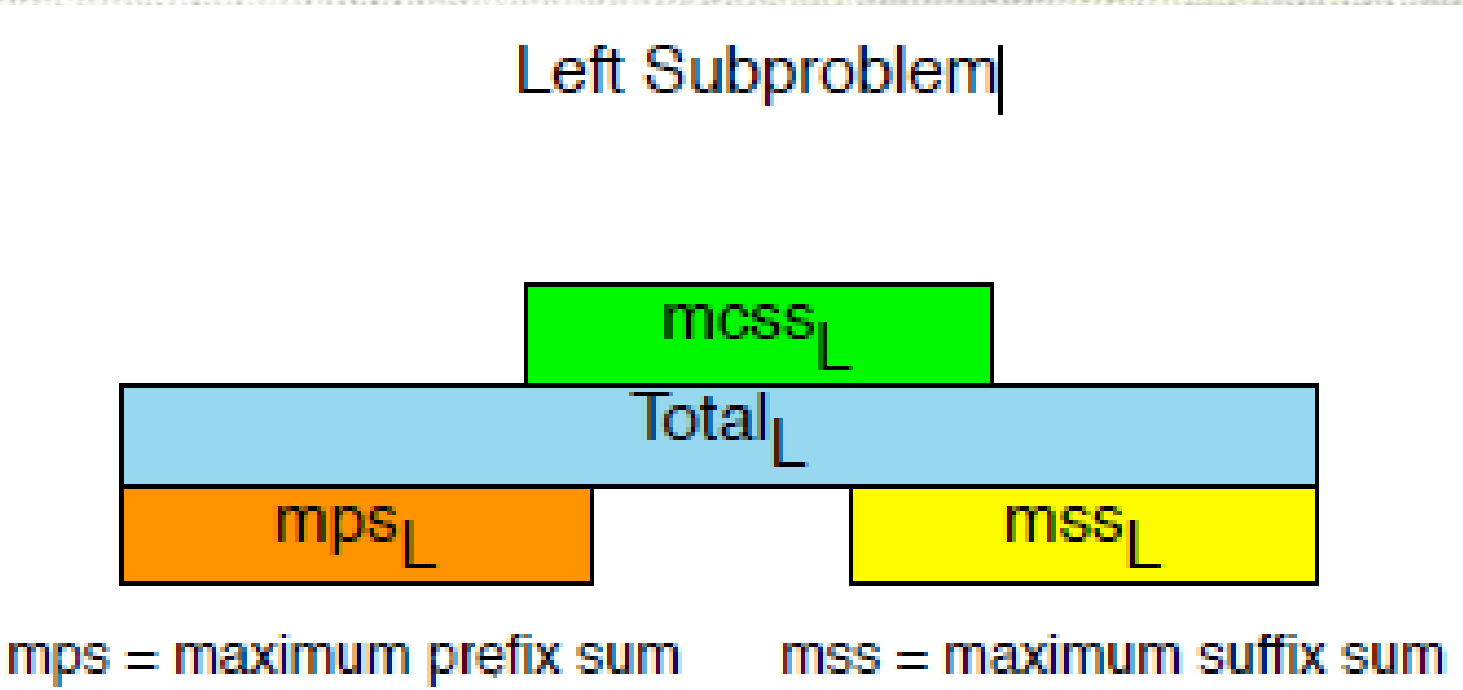
DIVIDE-AND-CONQUER – II

- The answers lie here

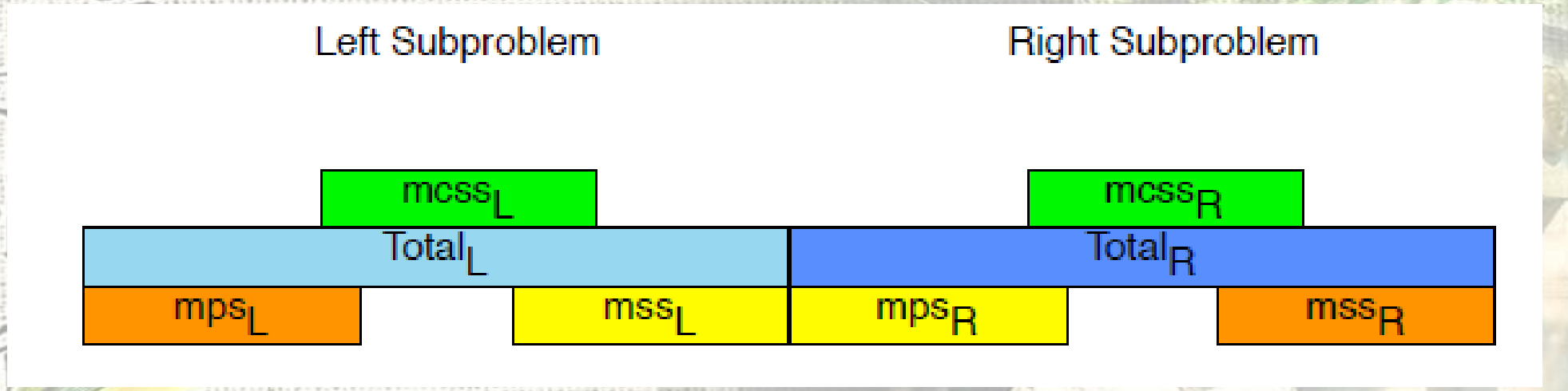


- Strengthen the subproblems
  - Compute additional information

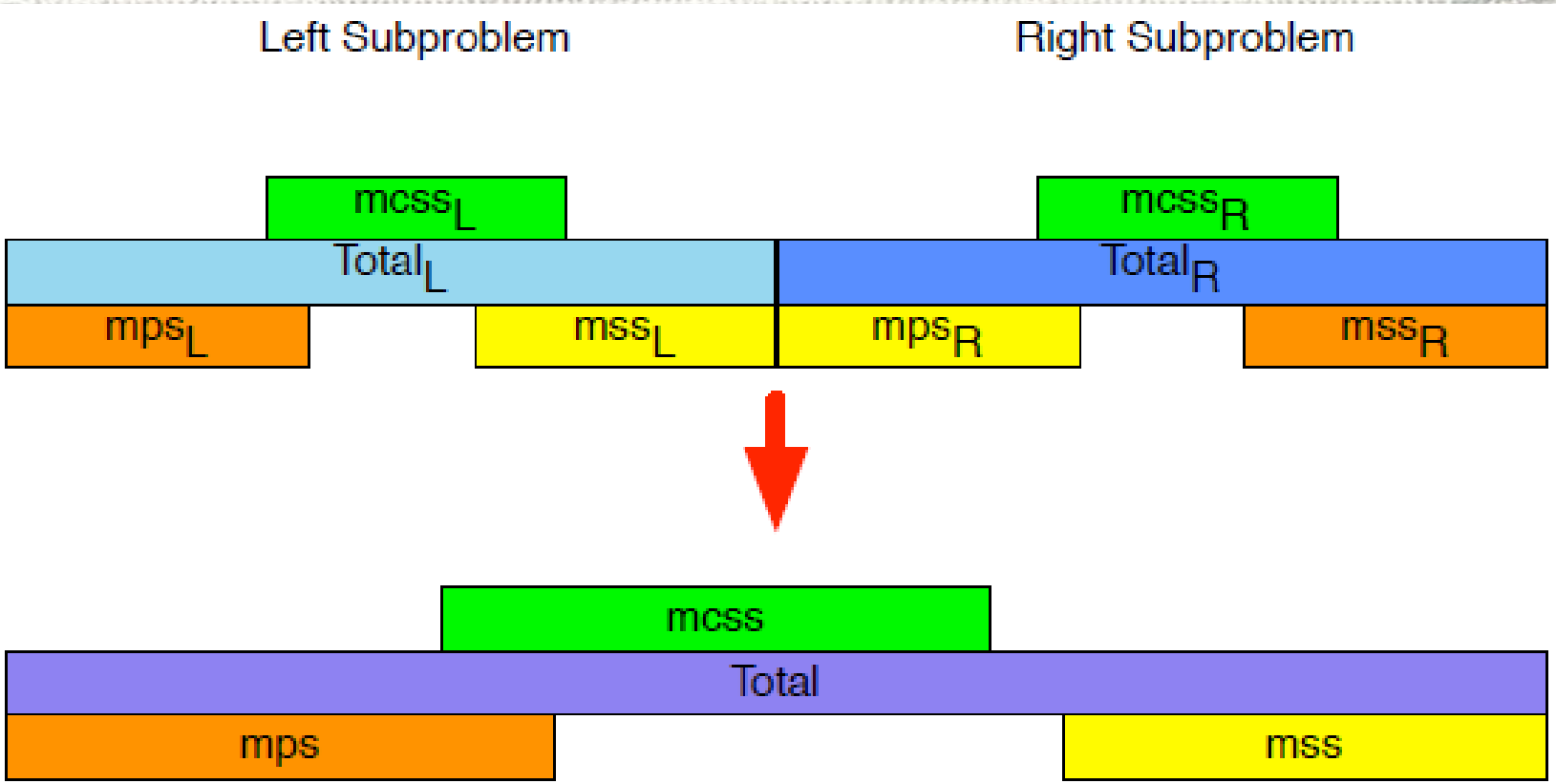
DIVIDE-AND-CONQUER – II



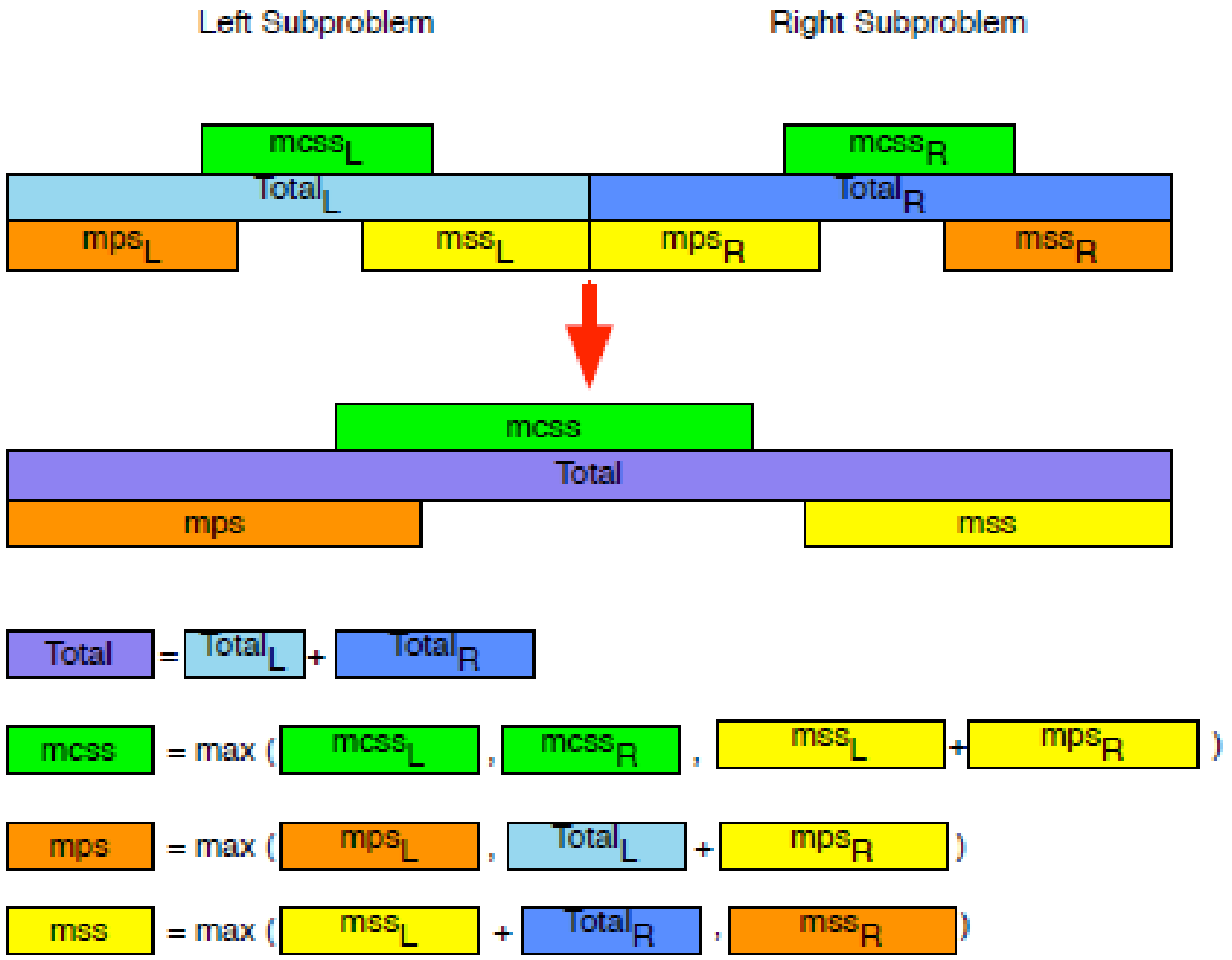
DIVIDE-AND-CONQUER – II



DIVIDE-AND-CONQUER – II



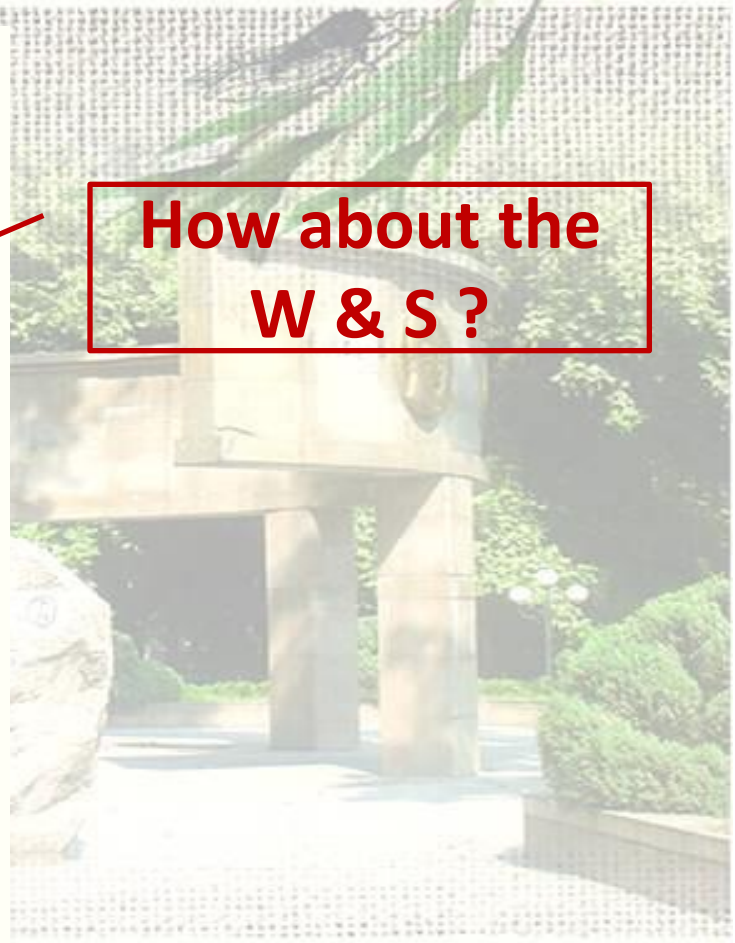
DIVIDE-AND-CONQUER – II



DIVIDE-AND-CONQUER – II

Algorithm (Linear Work Divide-and-Conquer MCSS)

$$
\begin{aligned}
&MCSSDCAux\ a = \\
&\quad \text{if } |a| = 0 \text{ then} \\
&\qquad (-\infty, -\infty, -\infty, 0) \\
&\quad \text{else } if |a| = 1 \text{then} \\
&\qquad (a[0], a[0], a[0], a[0]) \\
&\quad \text{else} \\
&\qquad \text{let} \\
&\qquad\quad (b, c) = splitMid\ a \\
&\qquad\quad ((m_1, p_1, s_1, t_1), (m_2, p_2, s_2, t_2)) = (MCSSDCAux\ b \ ||\ MCSSDCAux\ c) \\
&\qquad \text{in} \\
&\qquad\quad (max\,(s_1 + p_2, m_1, m_2), \\
&\qquad\quad\ max\,(p_1, t_1 + p_2), \\
&\qquad\quad\ max\,(s_1 + t_2, s_2), \\
&\qquad\quad\ t_1 + t_2) \\
&\qquad \text{end} \\
&MCSSDC\ a = \\
&\quad \text{let} \\
&\qquad (m, \_, \_, \_) = MCSSDCAux\ a \\
&\quad \text{in} \\
&\qquad m \\
&\quad \text{end}
\end{aligned}
$$



How about the W & S ?

COST ANALYSIS

Algorithm (Linear Work Divide-and-Conquer MCSS)

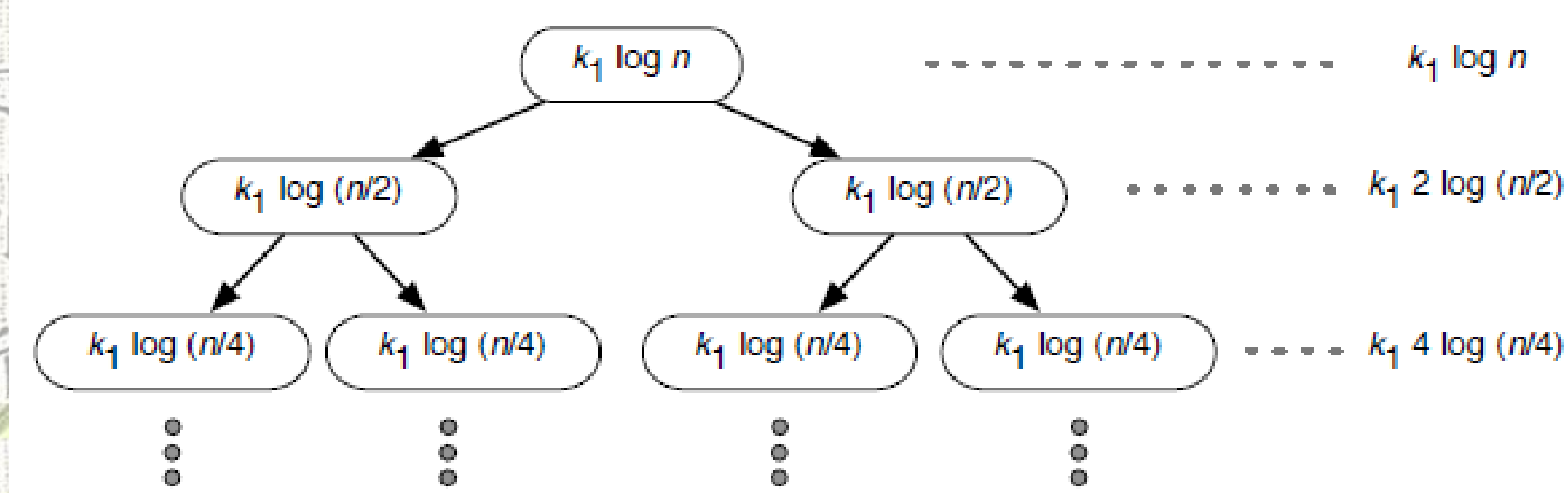
$$
\begin{aligned}
&MCSSDCAux\ a = \\
&\quad \text{if } |a| = 0 \text{ then} \\
&\qquad (-\infty, -\infty, -\infty, 0) \\
&\quad \text{else } if|a| = 1\text{then} \\
&\qquad (a[0], a[0], a[0], a[0]) \\
&\quad \text{else} \\
&\qquad \text{let} \\
&\qquad\quad (b, c) = splitMid\ a \\
&\qquad\quad ((m_1, p_1, s_1, t_1), (m_2, p_2, s_2, t_2)) = (MCSSDCAux\ b \ \|\ MCSSDCAux\ c) \\
&\qquad \text{in} \\
&\qquad\quad (max\,(s_1 + p_2, m_1, m_2), \\
&\qquad\quad\ max\,(p_1, t_1 + p_2), \\
&\qquad\quad\ max\,(s_1 + t_2, s_2), \\
&\qquad\quad\ t_1 + t_2) \\
&\qquad \text{end} \\
&MCSSDC\ a = \\
&\quad \text{let} \\
&\qquad (m, \_, \_, \_) = MCSSDCAux\ a \\
&\quad \text{in} \\
&\qquad m \\
&\quad \text{end}
\end{aligned}
$$

- Since splitMid requires O(log n) work and span in both array and tree sequences
  - W(n) = 2W(n/2) + O(log n)
    S(n) = S(n/2) + O(log n)

Right?

COST ANALYSIS

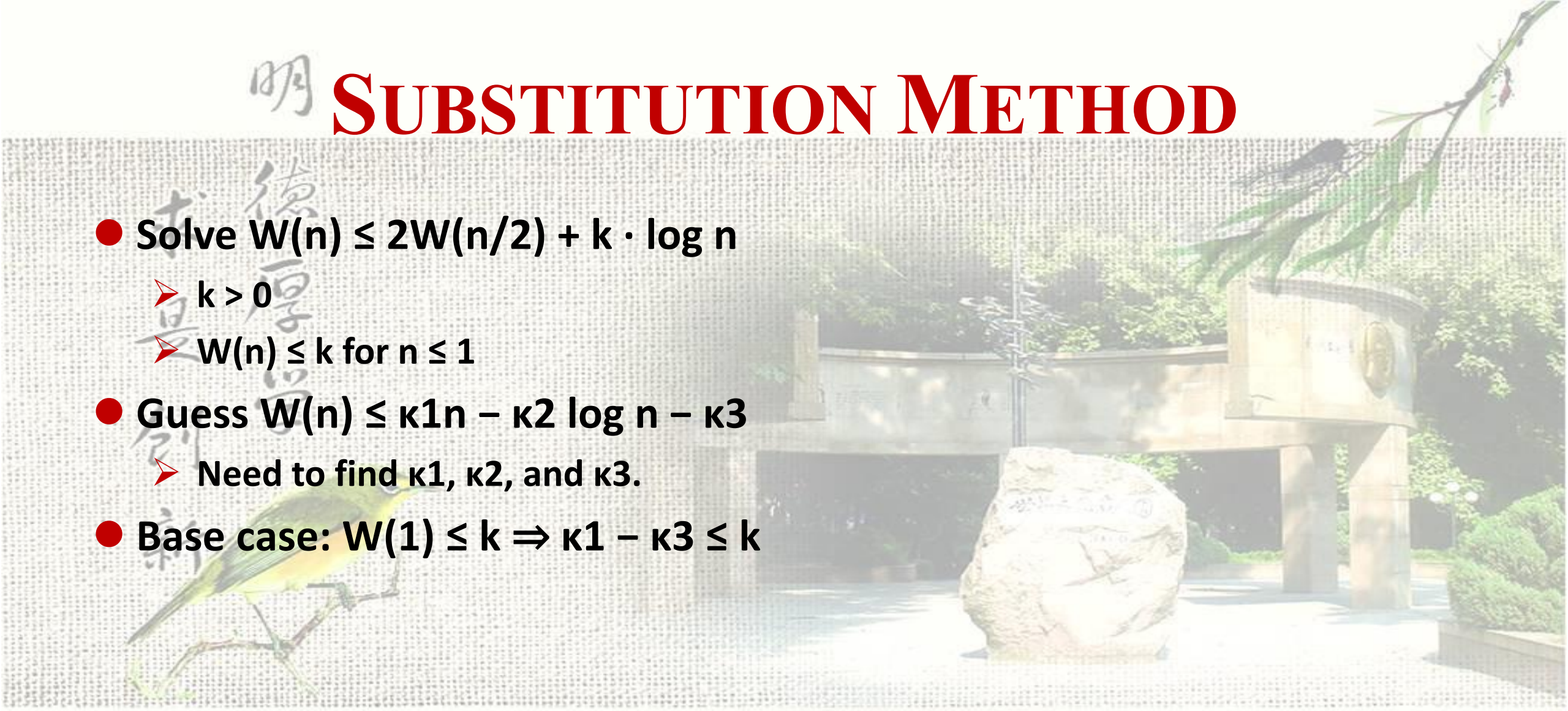
- $W(n) = 2W(n/2) + O(\log n)$



$$W(n) \le \sum_{i=0}^{\log n} k_1 2^i \log(n / 2^i)$$

Substitution Method

- Solve $W(n) \le 2W(n/2) + k \cdot \log n$
  - $k > 0$
  - $W(n) \le k$ for $n \le 1$
- Guess $W(n) \le \kappa 1 n - \kappa 2 \log n - \kappa 3$
  - Need to find $\kappa 1$, $\kappa 2$, and $\kappa 3$.
- Base case: $W(1) \le k \Rightarrow \kappa 1 - \kappa 3 \le k$



SUBSTITUTION METHOD

- **Inductive Step**

$$W(n) \le 2W(n/2) + k \cdot \log n$$
$$\le 2(\kappa_1 n/2 - \kappa_2 \log(n/2) - \kappa_3) + k \cdot \log n$$
$$= \kappa_1 n - 2\kappa_2(\log n - 1) - 2\kappa_3 + k \cdot \log n$$
$$= (\kappa_1 n - \kappa_2 \log n - \kappa_3) + (k \log n - \kappa_2 \log n + 2\kappa_2 - \kappa_3)$$
$$\le \kappa_1 n - \kappa_2 \log n - \kappa_3$$

- **Choose $\kappa_2 = k$ and $2\kappa_2 - \kappa_3 \le 0$ (Why?)**
- **For example, $\kappa_2 = k, \kappa_1 = 3k, \kappa_3 = 2k$ satisfies the constraints.**

Exercise

- The analysis given in the example above is not “tight” in the sense that there is a bound that is asymptotically dominated by O(|a|·|b|). Can you improve on the bound?
- Describe the changes to the algorithms Algorithm MCS: Brute Force and Algorithm MCSS: Brute Force to implement the strengthening described above. How does strengthening impact the work and span costs?
- Prove the MCSSE Extension lemma.