COMP 633 - Parallel Computing

Lecture 3
Aug 26, 31 + Sep 2 2021

PRAM (2)
PRAM algorithm design techniques

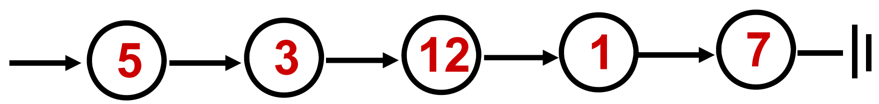
- Reading for next class (Sep 7): PRAM handout secn 5
- Written assignment 1 is posted, due Thu Sep 16

Topics

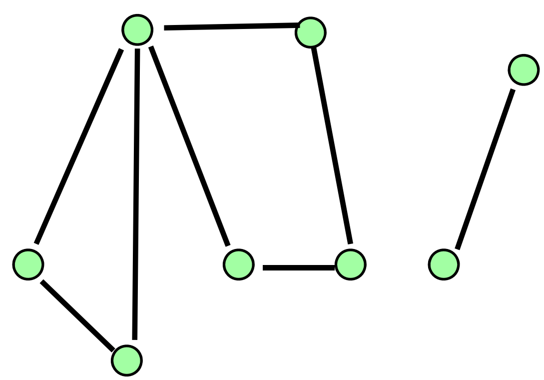
- PRAM Algorithm design techniques
  - pointer jumping
  - algorithm cascading
  - parallel divide and conquer

Design Technique: Pointer Jumping

- Fast parallel processing of linked data structures
  - linked lists
    - Membership, reduction and prefix sum of linked lists



  - graphs (adjacency lists, edge lists)
    - connected components
    - minimum spanning trees



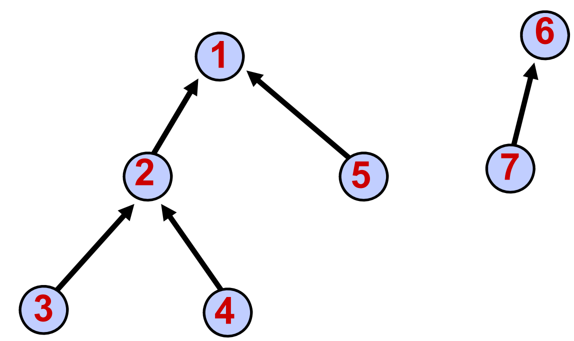
Example: Finding the roots of a forest

- Input

  G = (V,E) a forest of directed trees

- Output

  s[1:n] where for each vertex *j*,
  s[j] is the root of the tree containing *j*



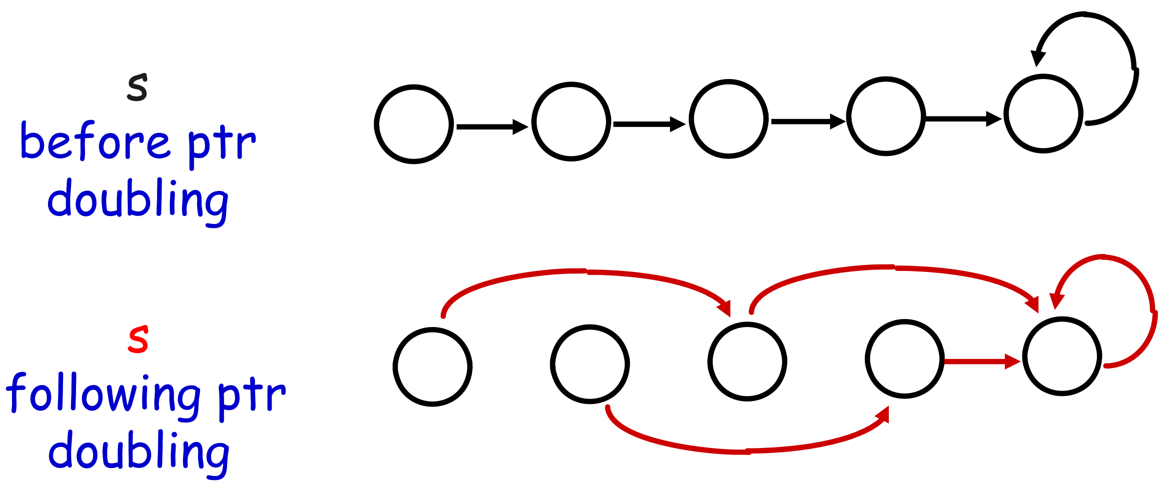
- Representation of G
  - in a directed tree
    - the root has no parent
    - every other vertex has a unique parent
  - V = {1, ..., *n*}
  - E is defined by *s*: $V \rightarrow V$
    - *s*(*u*) = *v* if *v* is parent of *u* in *G*
    - *s*(*r*) = *r* if *r* is a root in *G*
    - *s* is represented using an array *s*[1:*n*]

	1	2	3	4	5	6	7
s	1	1	2	2	1	6	6

Following a list in parallel: Pointer jumping

- Let (n, s[1..n]) be the representation of directed forest G
- Pointer jumping operation
  - every vertex directs its edge to its grandparent in parallel
  - also called *pointer doubling*

```
forall i in 1:n do
    s[i] := s[s[i]]
enddo
```



Analysis of pointer jumping

- pointer jumping halves distance to the root in s
  - let d be the distance in *s* from vertex *u* to the root
  - after pointer jumping distance in *s* from *u* to root is $\lceil d/2 \rceil$

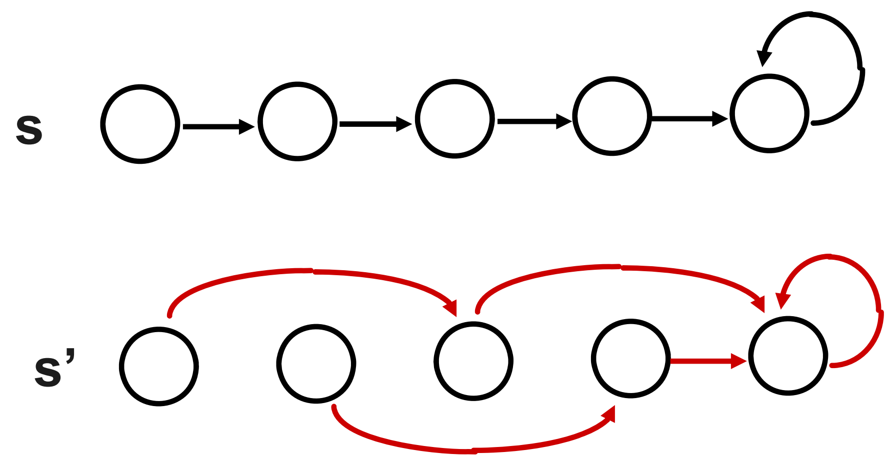
```
forall i in 1:n do
     s[i] := s[s[i]]
enddo
```

- S(n) = O(1)

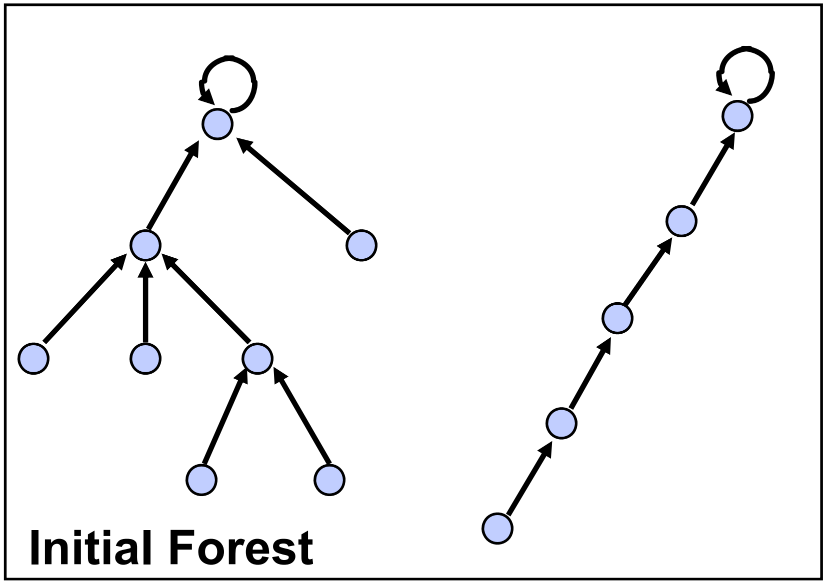
- W(n) = O(n)

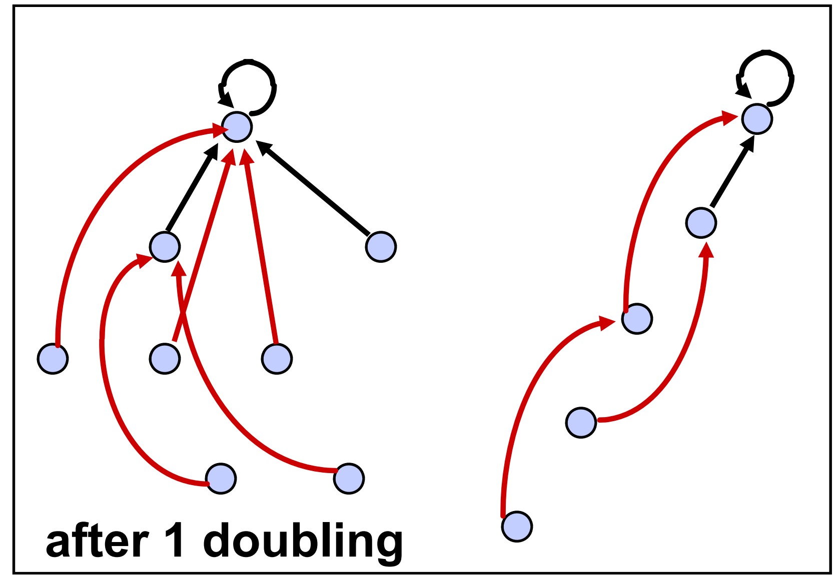
- PRAM model

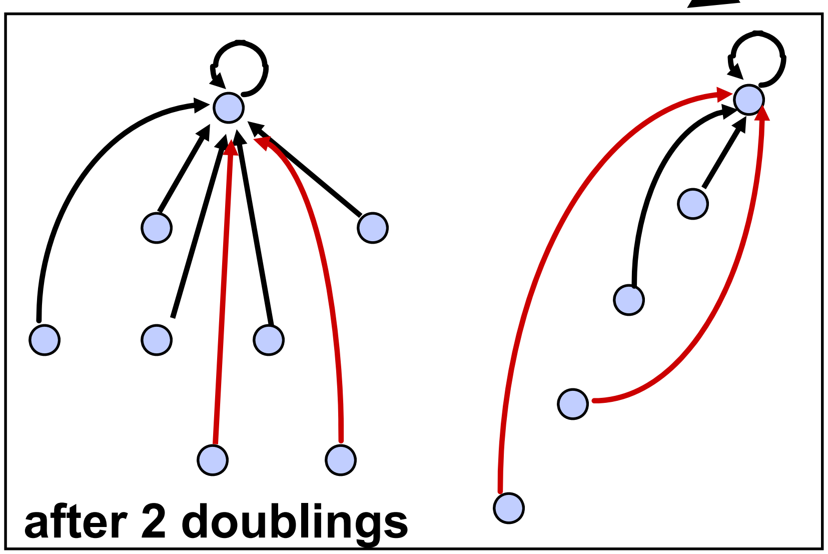
CREW



Pointer jumping in a forest







All vertices point to the root of their tree

Finding roots of a forest

- pointer jumping reaches a fixed point when forest has max height ≤ 1
  - vertex i is distance 1 or less from root when s[i] = s[s[i]]

- forest height $\leq 1 \Rightarrow$ s[i] = root of tree containing i

```
forall i in 1:n do
        while  s[i] != s[s[i]] do
             s[i] := s[s[i]]
        end do
enddo
```

Problem: find distance to root in directed forest

- Construct an algorithm for the following problem
  - Let (n, s[1..n]) be directed forest G
  - For each vertex $1 \leq i \leq n$, set d[i] to be the distance from i to the root of its tree

- Invariant: let d[i] be the distance in G from i to s[i]
  - establish initially
  - maintain property with each pointer doubling
  - termination implies result

```
forall i in 1:n do
   d[i] := (s[i]== i)? 0 : 1
end do
for i := 1 to (lg n) do
   forall i in 1:n do
      d[i] := d[i] + d[s[i]]
      s[i] := s[s[i]]
    end do
end do
```

- Complexity

  W(n) = $O(n \lg n)$

  S(n) = $O(\lg n)$

Design Technique: Algorithm Cascading

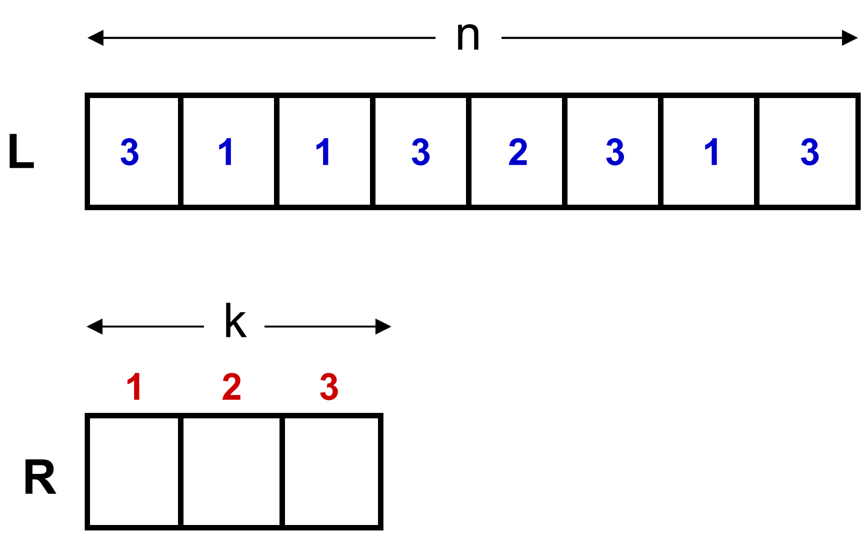
- Technique for improving work efficiency of an algorithm
  - suppose we have
    - work-inefficient but fast parallel algorithm A
    - work-efficient but slow algorithm B (typically sequential)
  - combine ("cascade") A and B to get best of both

"Speeding up by slowing down"

Example: histogram values in a sequence

- Input
  - Sequence L[1..n] with integer values in the range 1..k, where k = lg n

- Output
  - R[1..k] with R[i] = # occurrences of i in L[1..n]



Sequential algorithm

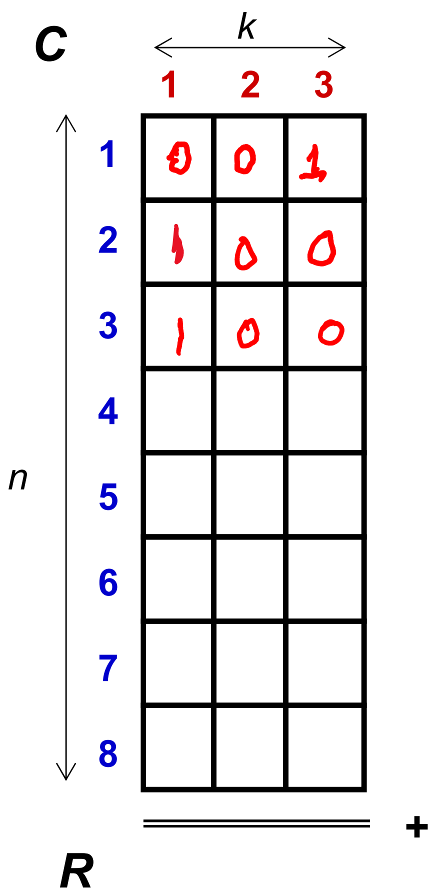
```
R[1:k] := 0
for i := 1 to n do
    R[L[i]] := R[L[i]] + 1
end do
```

$T_s(n) =$ O(n)

Parallel Algorithm: First try

$$C_{i,j} = \begin{cases} 1, & \text{if } L_i = j \\ 0, & \text{otherwise} \end{cases} \qquad R_j = \sum_{i \in 1:n} C_{i,j}$$

L							
3	1	1	3	2	3	1	3




```
integer C[1:n,1:k]
forall i in 1:n, j in 1:k do
        C[i,j] := (L[i]==j) ? 1 : 0
end do
forall j in 1:k do
        R[k] := REDUCE(C[1:n,j], + )
end do
```

PRAM

W(n) = $O(nk) + O(nk)$

S(n) = $O(1) + O(\lg n)$

model CREW

Cascading the histogram algorithm

- partition L into m "chunks" of size (lg n)
  - k = lg n (assume k divides n)
  - m = n / k = n / lg n

- compute mini-histogram sequentially within a chunk

  S_{chunk} = O(lg n)
  W_{chunk} = O(lg n)

- compute all m mini-histograms in parallel

  $S_{all} = S_{chunk}$
  $W_{all} = m \cdot W_{chunk}$ $= \frac{n}{\lg n} \cdot \lg n = O(n)$

- combine histograms by summing

  $S_{combine}$ = O(lg n)
  $W_{combine}$ = O(n)

```
integer C[1:m,1:k]
forall  i in 1:m,  j in 1:k do
    C[i,j]  := 0
end do
forall  i in 1:m  do
    for j  := 1 to k do
        C[i, L[(i-1)k+j] ] += 1
    end do
end do
forall j in 1:k do
    R[k]  := REDUCE(C[1:m,j], +)
end do
```

W(n) = O(n)

S(n) = O(lg n)

PRAM model? EREW

Parallel Divide and Conquer

- To solve problem instance P using parallel divide-and-conquer
  - divide P into subproblems (possibly in parallel)
  - apply D&C recursively to each subproblem in parallel
  - combine subsolutions to produce solution (possibly in parallel)

- Example: sorting
  - mergesort
    - combining
      - subproblems: left/right half of array
      - sort each subproblem
      - merge results
  - quicksort
    - partitioning
      - subproblems: values less than pivot, values greater than or equal to pivot
      - sort each subproblem
      - concatenate results

Parallel Mergesort (parallel divide and conquer)

- Assume parallel EREW `merge(A,B)` for |A| = |B| = O(n) with

  $W_{merge}(n) = O(n)$

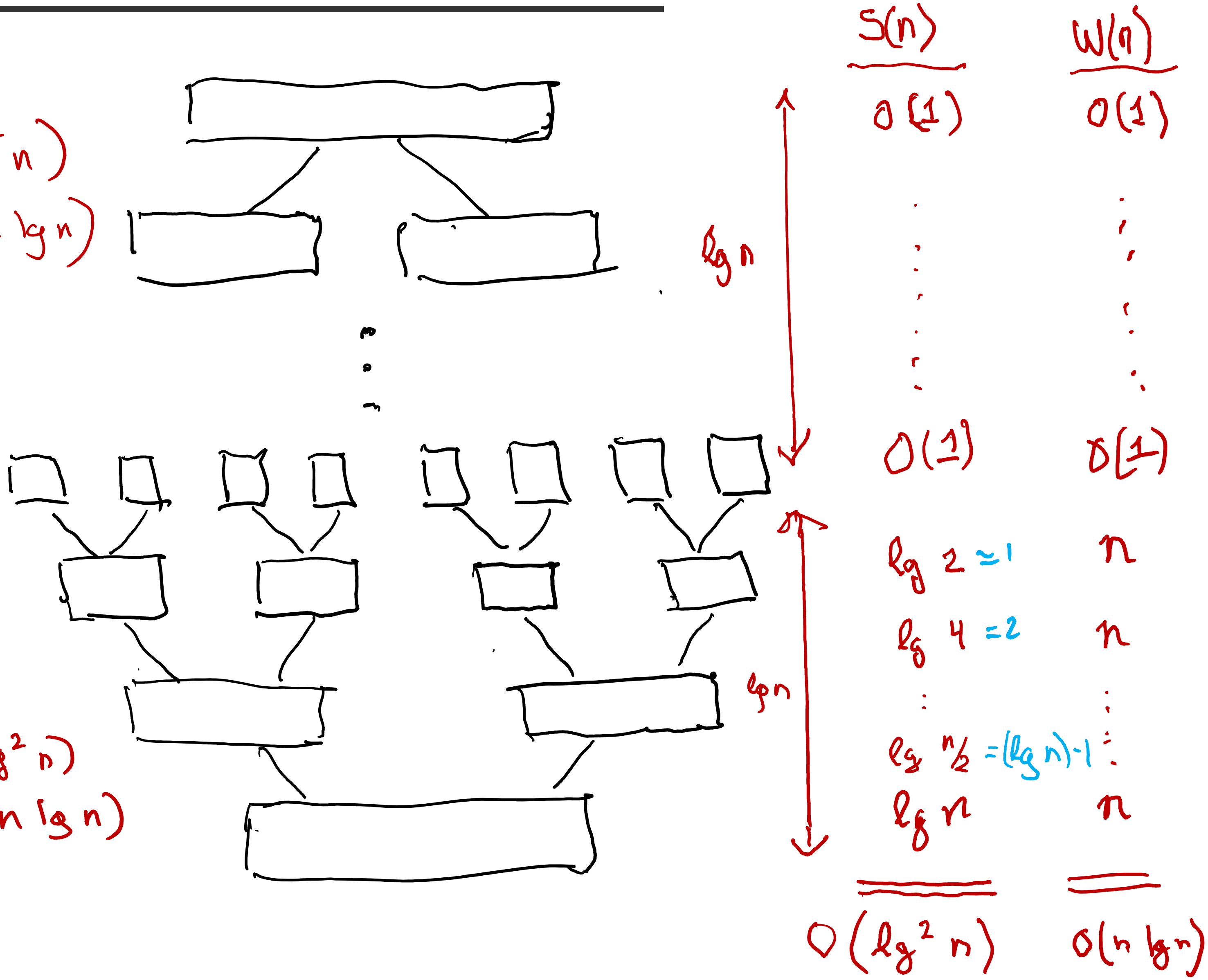
  $S_{merge}(n) = O(\lg n)$

```
mergesort(V[1:n]) =
if  n ≤ 1  then  S[1:n] := V[1:n]
else
    m := n/2
    {
        R[1:m]   = mergesort V[1:m]
      ||
        R[m+1:n] = mergesort V[m+1:n]
    }
    S[1:n] := merge( R[1:m], R[m+1:n] )
endif
return S[1:n]
```

Mergesort complexity (figure)

$S(n) = O(\lg^2 n)$

$W(n) = O(n \lg n)$



total

$S(n) = O(\lg^2 n)$

$W(n) = O(n \lg n)$

Parallel Mergesort (forall)

- Assume parallel EREW `merge(A,B)` for |A| = |B| = O(n) with

  $W_{merge}(n) = O(n)$

  $S_{merge}(n) = O(\lg n)$ ← exists, but hard

```
mergesort(V[1:n]) =
if  n ≤ 1  then  S[1:n] := V[1:n]
else
    m := n/2
    forall i in 0:1 do
       R[i*m+1 : (i+1)*m] = mergesort V[i*m+1 : (i+1)*m]
    end do
    S[1:n] := merge( R[1:m], R[m+1:2*m] )
endif
return S[1:n]
```

$S_{mergesort}(n) =$ $O(\lg^2 n)$

$W_{mergesort}(n) =$ $O(n \lg n)$

Parallel Quicksort

- Assume parallel EREW `partition(A,p)` for |A| = O(n) with

  $W_{partition}(n) = O(n)$

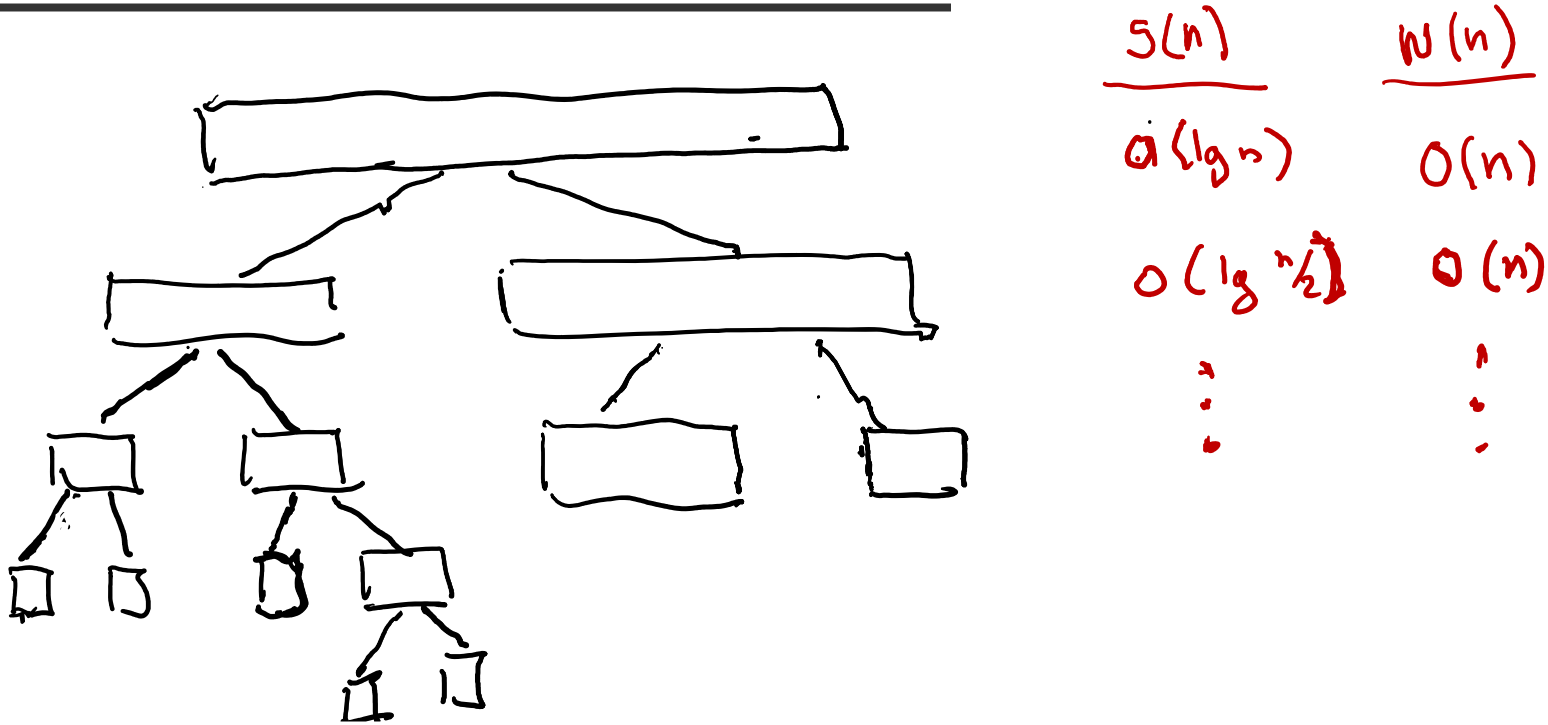
  $S_{partition}(n) = O(\lg n)$

```
quicksort(V[1:n]) =
if  n ≤ 1  then  S[1:n] := V[1:n]
else
    p := V[ random(1:n) ]
    R[1:n], m := partition (V[1:n], p)
    h[0:2] := [0, m, n]
    forall i in 0:1 do
       S[h(i)+1 : h(i+1)] = quicksort R[h(i)+1 : h(i+1)]
    end do
end if
return S[1:n]
```

$S_{quicksort}(n) = S(\frac{n}{2}) + O(\lg n) = O(\lg^2 n)$

$W_{quicksort}(n) = 2W(n/2) + O(n) = O(n \lg n)$

Quicksort complexity (figure)



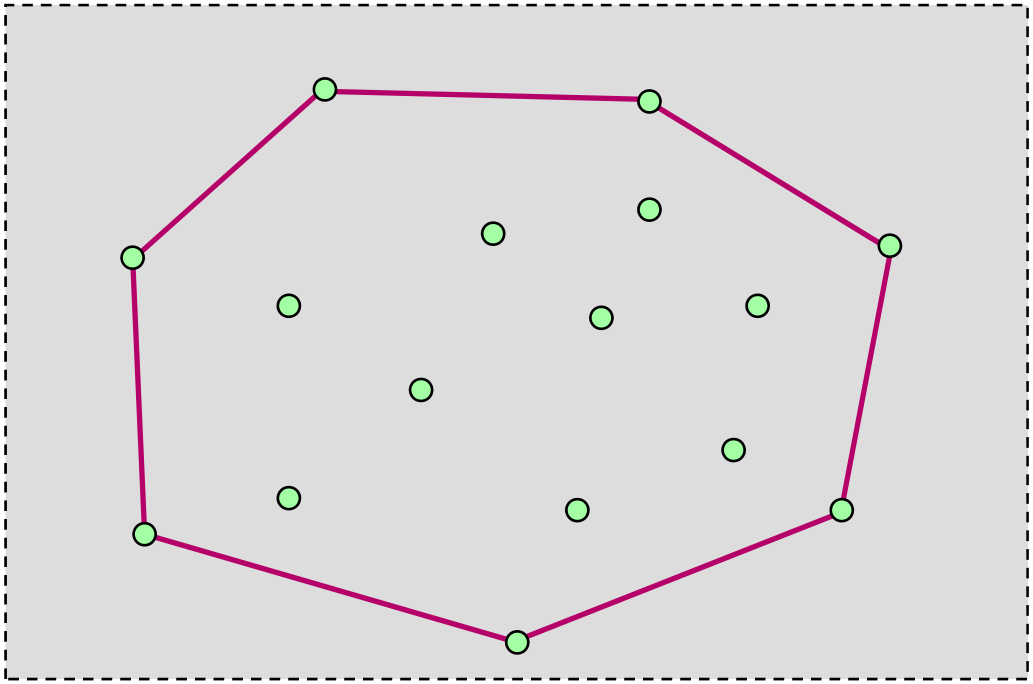
Best case: W(n) = 2W(n/2)+O(n) ➔ W(n)=O($n \lg n$)
S(n) = S(n/2) + O(lg n) ➔ S(n)=O($\lg^2 n$)

General case: unpredictable number and size of subproblems

Worst case: $\mathrm{W(n)} = O(n^2), S(n) = O(n \lg n)$

Planar Convex Hull Problem

- Input
  - $S = \{(x_i, y_i)\}$ set of n points in the plane
  - assume x_i distinct, y_i distinct, and no three points co-linear
- Output
  - tour of smallest convex polygon containing all points of S
- Complexity
  - $T_S^*(n) = \Theta(n \lg n)$



Two Parallel Algorithms for Planar Convex Hull

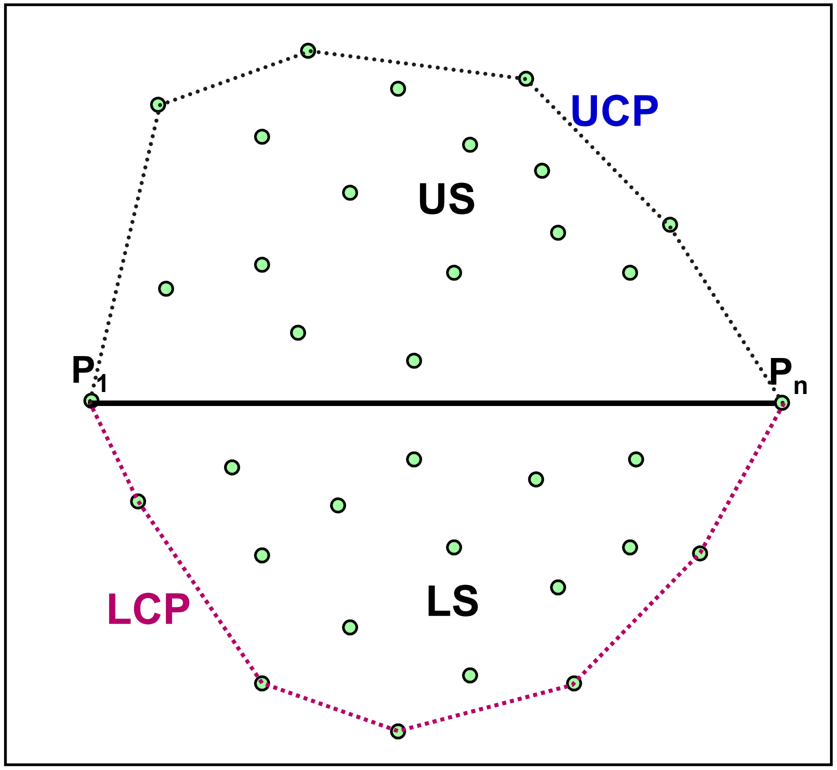
- two divide and conquer algorithms
  - combining approach
  - partitioning approach

- combining algorithm (like mergesort)
  - assume input points presented in order of increasing x coordinate
    - can be obtained using $O(n \lg n)$ work, $O(\lg^2 n)$ step sorting algorithm
  - optimal worst case performance

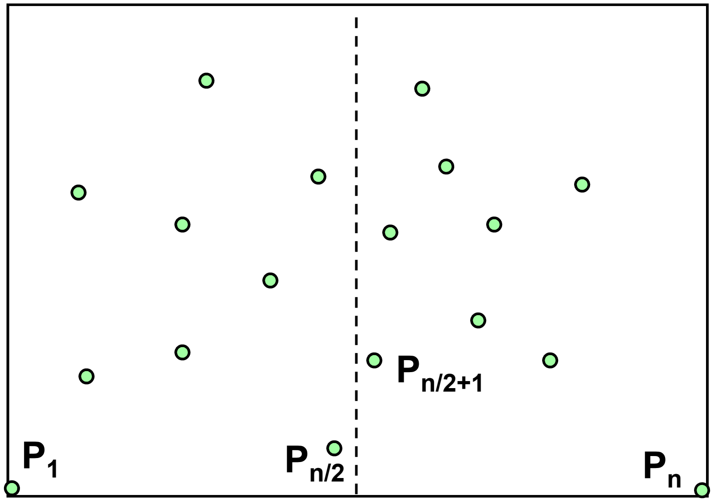
- partitioning algorithm (like quicksort)
  - no assumptions about order of input points
  - suboptimal worst case performance
  - very good expected case performance

D&C algorithm via combining

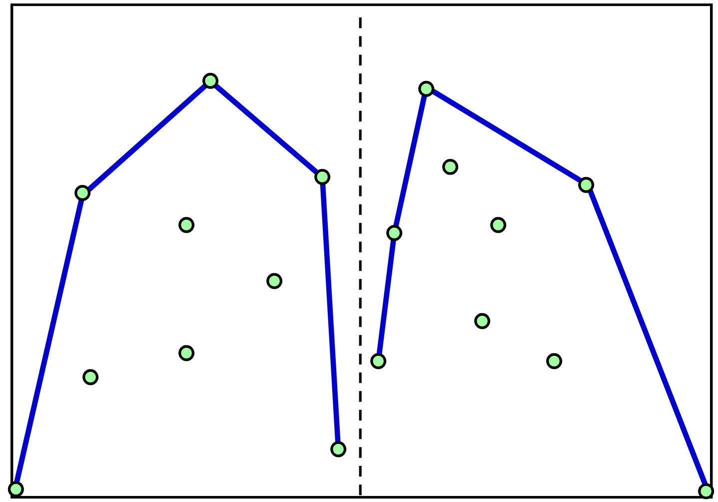
1. Divide S into US, LS by line $P_1 – P_n$
2. Compute Upper Convex Path and Lower Convex Path using D&C algorithm
3. Combine UCP, LCP to construct convex hull



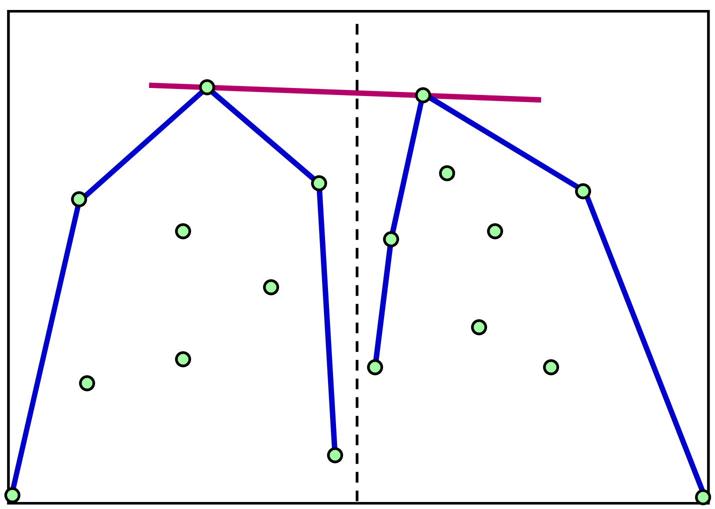
Construction of upper convex path



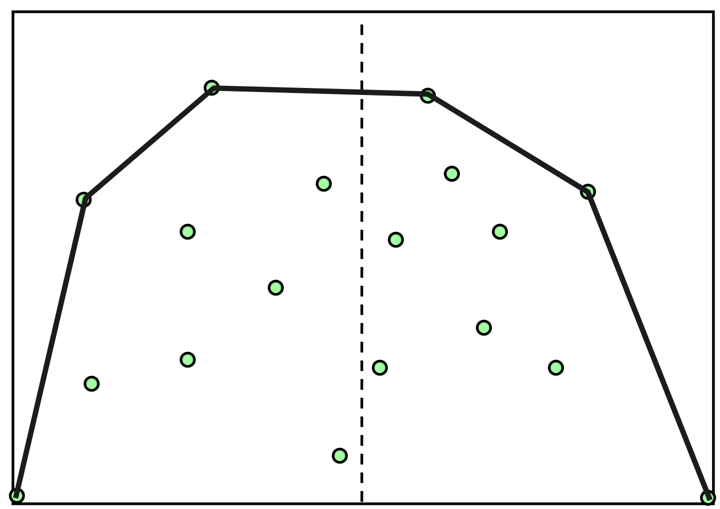
Divide



Recur



Combine (1): find upper common tangent



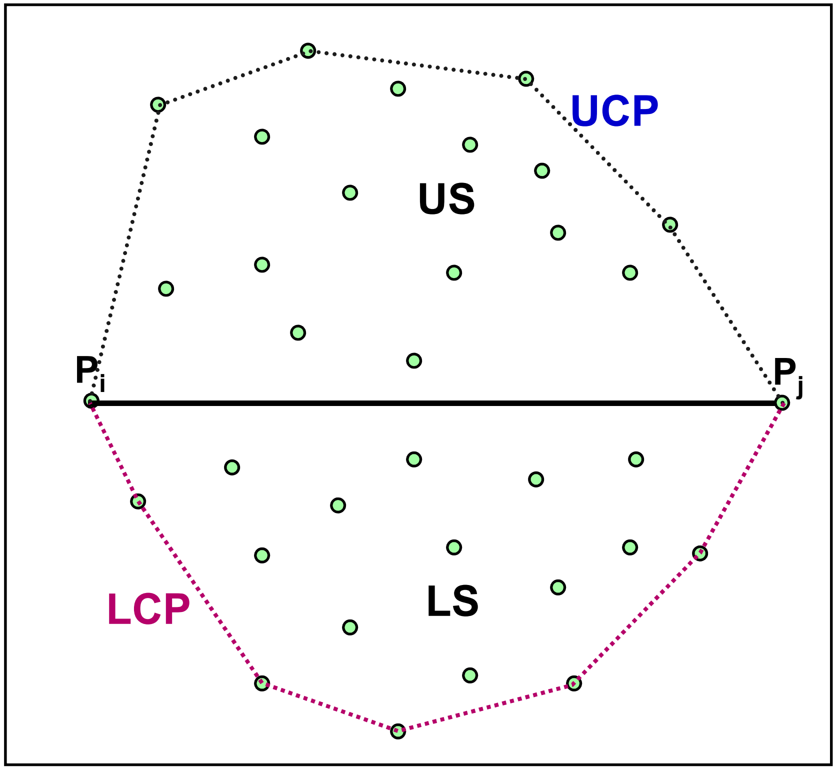
Combine (2): create upper convex path

Analysis (Combining algorithm)

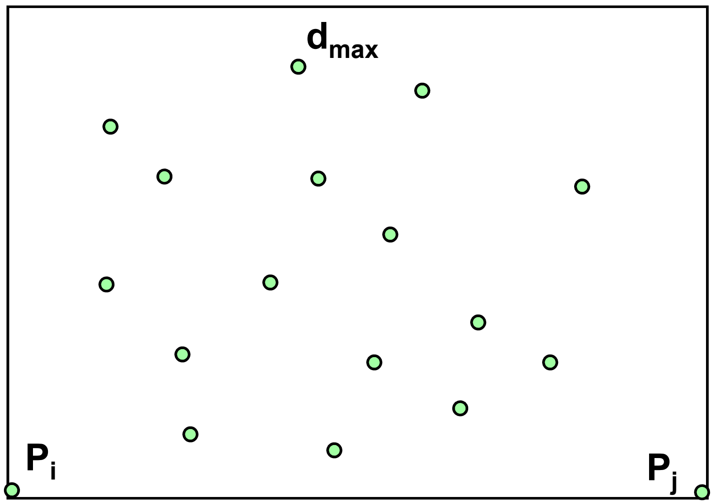
- Upper/Lower Convex path
  - Find common tangent (UCT/LCT)
    - binary search of convex paths to find tangent points [Overmars & van Leeuwen]
    - Sequential: $S(n) = W(n) = O(\lg n)$
  - Connect paths
    - CREW: $S(n) = O(1)$, $W(n) = O(n)$
    - EREW: $S(n) = O(\lg n)$, $W(n) = O(n)$
- Convex Hull
    - $S(n) = S(n/2) + O(\lg n)$
      - $S(n) = O(\lg^2 n)$
    - $W(n) = 2\,W(n/2) + O(n)$
      - $W(n) = O(n \lg n)$
  - Work-efficient, since $T_S(n) = \Theta(n \lg n)$

D&C algorithm via partitioning

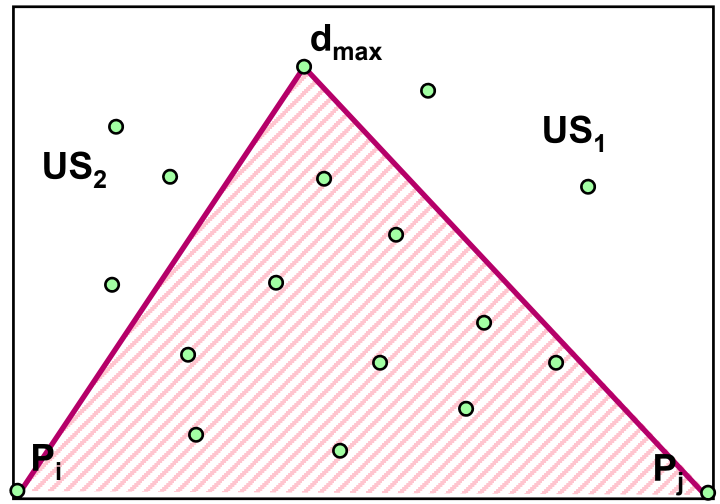
1. Divide S into US, LS by line P_i-P_j where P_i, P_j have extremal x coordinates
2. Compute Upper Convex Path and Lower Convex Path using D&C algorithm
3. Combine UCP, LCP to construct convex hull



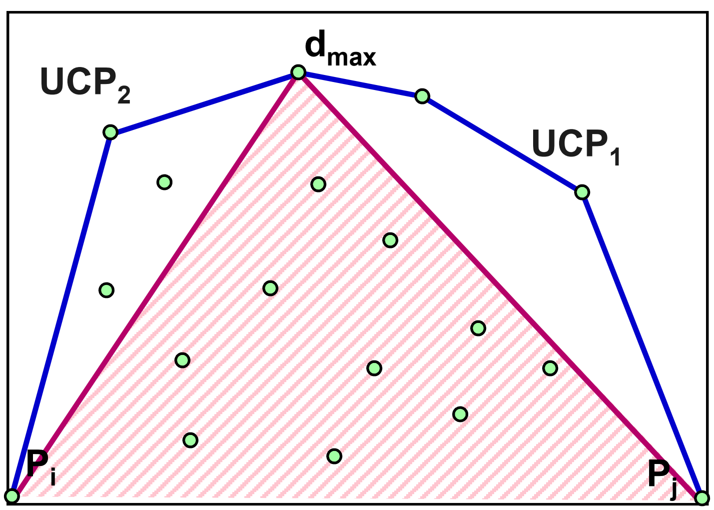
Construction of upper convex path



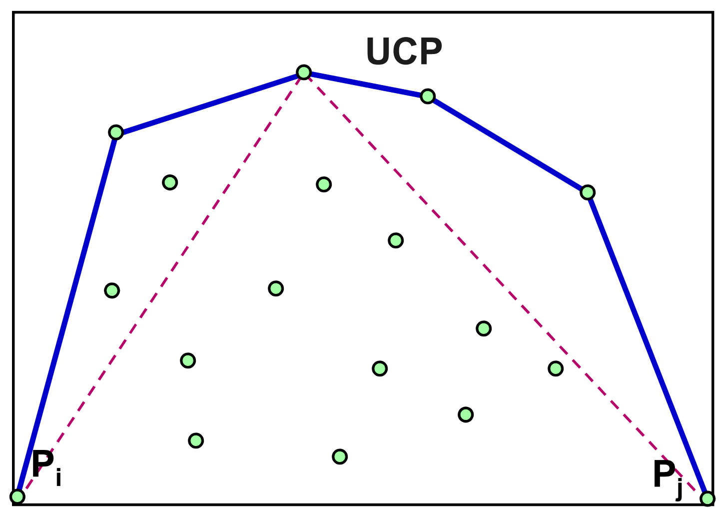
Locate point at max distance from P_i - P_j



Discard interior points and partition remaining points



Recur: find upper convex paths



Combine upper convex paths

Analysis (Partitioning algorithm)

- Upper/Lower Convex path for n points above baseline
  - Find point at maximum distance from baseline
    - $S(n) = O(\lg n)$, $W(n) = O(n)$
  - Partition
    - $S(n) = O(\lg n)$, $W(n) = O(n)$
  - Combine
    - $S(n) = O(\lg n)$, $W(n) = O(n)$

- Convex Hull
  - Find extremal points for initial baseline
    - $S(n) = O(\lg n)$, $W(n) = O(n)$
  - Construct UCP, LCP
    - $S(n) = \max(S(n_1), S(n_2)) + O(\lg n)$
    - $W(n) = W(n_1) + W(n_2) + O(n)$
      - $n_1 + n_2 \leq n$
  - Combine paths
    - $S(n)=O(1)$, $W(n) = O(n)$

Analysis of parallel partitioning algorithm

- Analysis
  - Expected partition, no points eliminated
    - $S(n) = S(n/2) + O(\lg n)$
      - $S(n) = O(\lg^2 n)$
    - $W(n) = 2\,W(n/2) + O(n)$
      - $W(n) = O(n \lg n)$
  - Worst-case partition, no points eliminated
    - $S(n) = S(n-1) + O(\lg n)$
      - $S(n) = O(n \lg n)$
    - $W(n) = W(1) + W(n-1) + O(n)$
      - $W(n) = O(n^2)$
  - Expected partition, random points in the unit square
      - $S(n) = O(\lg n\,(\lg\lg n))$
      - $W(n) = O(n \lg\lg n)$

Reminder: Master theorem for recurrence relations

- Recurrence form

$$H(n) = aH\left(\frac{n}{b}\right) + f(n) \qquad \text{where} \quad a \geq 1, b > 1$$

$$H(1) = O(1)$$

- Solution

$$H(n) = \Theta\left(a^k\right) + \Theta\left(\sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right)\right)$$

where $k = \log_b n$

Termination condition

- What about "while" inside "forall"?
  a) replace with fixed number of iterations
  b) detect termination condition

```
forall i in 1:n do
      while  s[i] != s[s[i]] do
         s[i] := s[s[i]]
      end do
enddo
```

let *h* be the max height of a tree in the forest

```
for i := 1 to  lg n do
    forall i in 1:n do
         s[i] := s[s[i]]
    end do
enddo
```

(a)

```
Seq(Bool) M[1:n]
repeat
    forall i in 1:n do
        s[i] := s[s[i]]
        M[i] := (s[i] == s[s[i]])
    end do
    t := REDUCE(M[1:n], and)
until (t)
```

(b)

W(n) =

S(n) =

W(n) =

S(n) =