BÀI 9. SEARCHING, MERGING & SORTING

NỘI DUNG BÀI HỌC

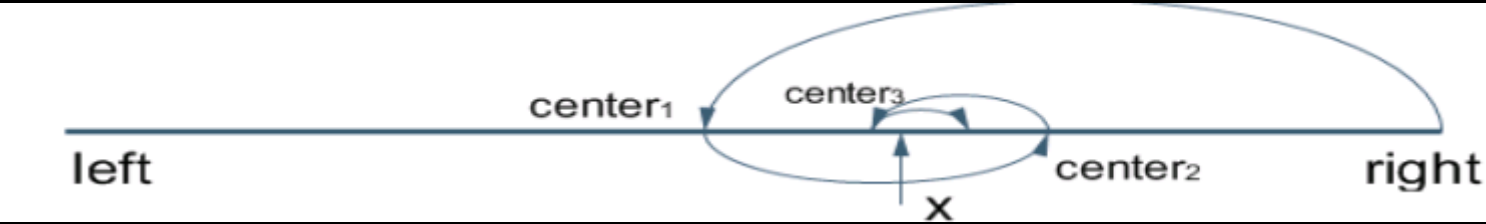
- Tìm kiếm song song với mảng đã sắp xếp.
- Trộn hai mảng đã sắp xếp
- Kỹ thuật trộn và sắp xếp theo Bitonic
- Các thuật toán sắp xếp đơn giản
- Song song hóa Quick Sort

1. SEARCHING

TÌM KIẾM TRÊN DÃY ĐÃ SẮP XẾP

- Phát biểu bài toán:
  - Input: A[1..n] | $A_1 < A_2 < \ldots < A_n$; X bất kỳ
  - Output: k | $A_k <= X <= A_{k+1}$
- Thuật giải tuần tự
  - Tìm kiếm nhị phân ($T(n) = O(\log_2 n)$)
- Thuật giải song song với PRAM:
  - Với n BXL
  - Với p BXL

THUẬT GIẢI TUẦN TỰ




```
input   : A[1..n] | A[1] < A[2] < ... < A[n];  x bat ky
output : k | A[k]  ≤ x ≤ A[k+1]
begin
      left = 1; right = n;
      while (left < right) then
            if x < A[left] then k = left -1;
            if x > A[right] then k = right;
            center = (left + right)/2;
            if x = A[center] then k = center;
            if  x < A[center] then right = center -1;
            else left = center + 1;
      end while
      k = left;
end
```

THUẬT GIẢI SONG SONG VỚI n BXL

```
input   : A[1..n] | A[1] < A[2] < ... < A[n];
        : x bat ky, PRAM O(n) BXL
output : k | A[k]  ≤ x ≤ A[k+1]
begin
      A[0] = - inf;  A[n+1] = inf;
      for i = 0 to n do in parallel
            if A[i] ≤ x < A[i+1] then  k = i;
      end parallel
end.
```

THUẬT GIẢI SONG SONG VỚI p BXL

```
input   : A[1..n] | A[1] < A[2] < ... < A[n];
        : x bat ky, PRAM O(p) BXL
output : k | A[k]  ≤ x ≤ A[k+1]
ParallelSearch(A,left, right, x, p)
begin
      if (right - left ≤ p) then Call SimplePSearch
      else
      begin
            q = n/p;
            for i = 1 to p do in parallel
                  if A[(i-1)q+1] ≤ x ≤ A[iq] then
                        left   = (i-1)q+1;
                        right = iq;
                  end if
            end parallel
            ParallelSearch(A,left, right, x,p);
end.
```

2. MERGING

TRỘN 2 MẢNG ĐÃ SẮP XẾP

- Phát biểu bài toán:
  - Input: Cho 2 mảng đã sắp xếp: A[1..n], B[1..n]
  - Output: trộn A,B thành mảng C được sắp xếp
- Thuật giải tuần tự: O(n)
- Thuật giải song song
  - Kỹ thuật Partitioning
  - Phương pháp Ranking

PHƯƠNG PHÁP RANKING

- Khái niệm Rank (hạng 1 phần tử):
  - A[1..n] là dãy đã sắp xếp: A[1] < A[2] < .. < A[n]
  - k = rank (x:A) | A[k] <= x < A[k+1]
- Khái niệm hạng của mảng:
  - Xét A[1..n], B[1..n] là 2 mảng đơn điệu tăng
  - Rank(A:B) = {R_{AB}[i] = rank(A[i] : B)}, i = 1..n

THUẬT GIẢI DỰA TRÊN RANKING

- Ý tưởng:
  - Gọi C là kết quả trộn 2 mảng A, B
  - Hạng 1 phần tử x trên mảng C là số phần tử trên C không lớn hơn x
  - Ta đi xác định mỗi phần tử của A và B trên C trong đó:
    - Rank(A[i]:A) = i;
    - Rank(A[i]:B) xác định bằng BinarySearch với 1 BXL

THUẬT GIẢI DỰA TRÊN RANKING

```
input   : A[1..n], B[1..n] đơn điệu tăng
output  : C[1..2n] = A[1..n] U B[1..n] đơn điệu tăng
begin
      for i = 1 to n do in parallel
            Rab[i]    =    BinarySearch(A[i] : B);
            Rba[i]    =    BinarySearch(B[i] : A);
            Rac[i]    =    Rab[i] + i;
            Rbc[1]    =    Rba[i] + i;
      end parallel
      for i = 1 to n do in parallel
            C[Rac[i]] =    A[i];
            C[Rbc[i]] =    B[i];
      end parallel
end.
```

TRỘN 2 NỬA ĐƠN ĐIỆU

- Phát biểu bài toán:
  - Input: 2 dãy A[1..n], B[1..n] sắp xếp trái chiều
  - Output: trộn A, B được C là 2 dãy đơn điệu
- Thuật giải tuần tự:
  - Sắp xếp kiểu Bitonic
- Thuật giải song song:
  - Mạng Bitonic

KHÁI NIỆM DÃY BITONIC

- Dãy A[1..n] được gọi là Bitonic nếu:
  - Tồn tại (j,k) thuộc 1..n sao cho 2 dãy:
    - {A[j+1], A[j+2],…. A[k]}
    - {A[(k+1) mod n],…. , A[j]}
  - Là 2 dãy đơn điệu trái chiều nhau
- Ví dụ: dãy Bitonic với j=4, k = 14

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	22	27	39	48	45	40	35	34	32	25	24	18	8	5	7	14

ĐỊNH LÝ VỀ DÃY BITONIC

- Nếu A[1..n] là dãy Bitonic thì:
  - L[i] = min{A[i], A[i+n/2]} với mọi i = 1.. n/2
  - R[i] = max {A[i], A[i+ n/2]} với mọi i = 1.. n/2
  - Ta được 2 dãy {L[i]} và {R[i]}, i = 1.. n/2 là các dãy Bitonic và L[i] <= R[j] mọi i, j = 1..n/2

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	22	27	39	48	45	40	35	34	32	25	24	18	8	5	7	14
i	1	2	3	4	5	6	7	8								
L	22	25	24	18	8	5	7	14								
i									1	2	3	4	5	6	7	8
R									32	27	39	48	45	40	35	34

SẮP XẾP DÃY BITONIC

- Phát biểu bài toán:
  - Input: A[1..n] là dãy bitonic
  - Output: sắp xếp dãy A[1..n]

SẮP XẾP DÃY BITONIC

```
input    : dãy A[1..n] bitonic
output   : sắp xếp dãy A

PBitonic(A,n)
begin
     for i = 1 to n/2 do in parallel
          L[i]      = min { A[i], A[i+n/2] }
          R[i]      = max{ A[i], A[i+n/2] }
     end parallel

     PBitonic(L,n/2);
     PBitonic(R,n/2);
end.
```

TRỘN 2 MẢNG ĐÃ SẮP XẾP

- Ý tưởng:
  - Nếu A[1..n], B[1..n] là 2 mảng đã sắp xếp cùng chiều thì:
    - C[1..n] = A[1..n]
    - C[n+1..2n] = B[n..1]
  - Nếu A[1..n], B[1..n] là 2 mảng đơn điệu trái chiều thì:
    - C[1..n] = A[1..n]
    - C[n+1,..2n] = B[1..n]
  - Kết quả được C[1..2n] là dãy Bitonic. Sắp xếp dãy C[1..2n] theo thuật toán trước

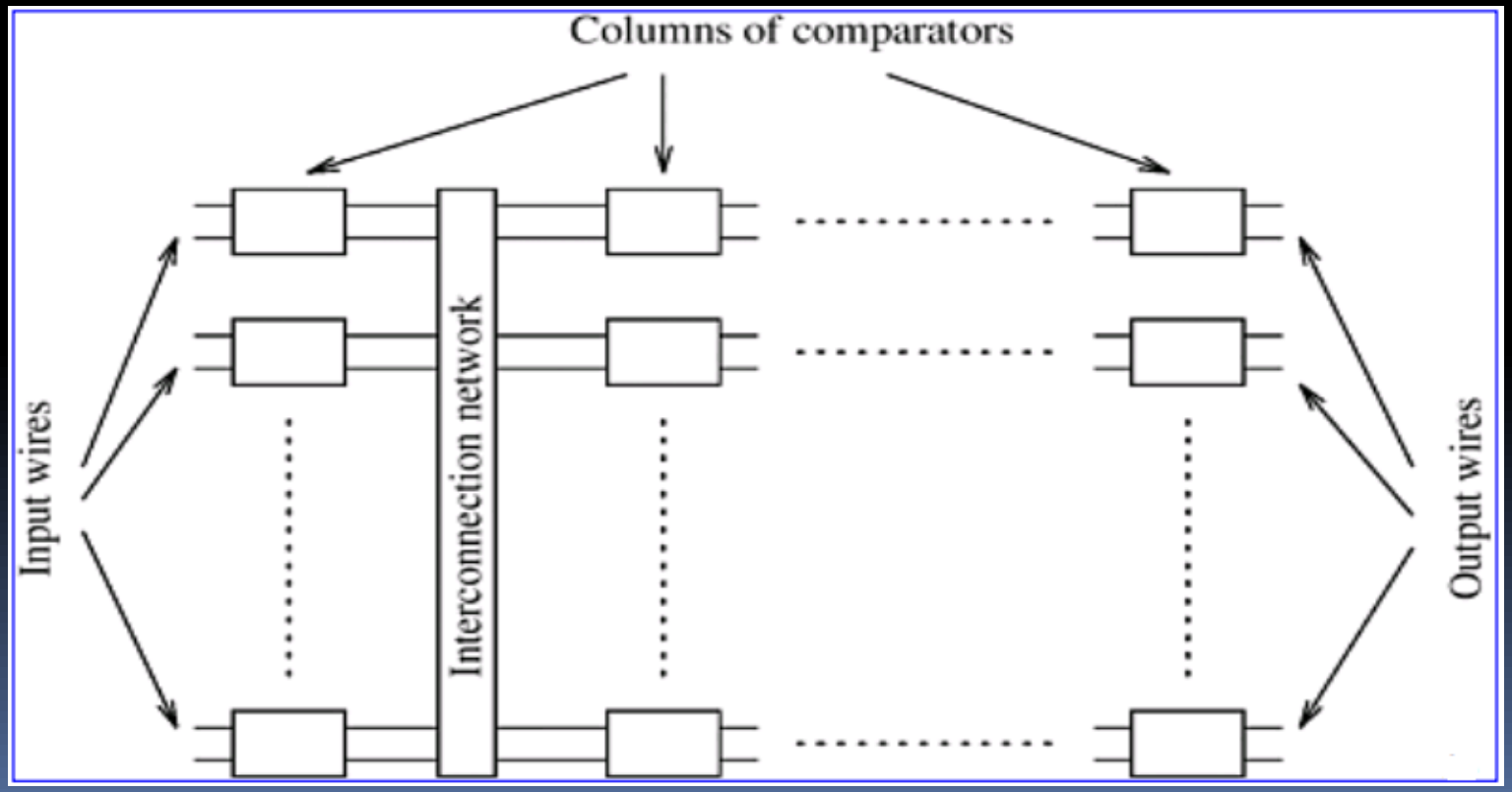
TRỘN 2 MẢNG ĐÃ SẮP XẾP

```
Input:    a[0] < a[1] < .. < a[n-1]
          b[0] < b[1] < .. < b[n-1]
Output: dãy trộn a[0] < a[1] < .. < a[2n-1].
begin
     A(n:2n-1) = B(n-1:0)
     for i = 0 to n-1 do in parallel
          if( a[i] > a[i+n]) then
               Swap(a[i],a[i+n]);
          end if;
     end parallel;

     thực hiện đệ quy với 2 nửa A(0:n-1) và A(n:2n-1) một
     cách riêng rẽ và song song.
end.
```

MẠNG SẮP XẾP

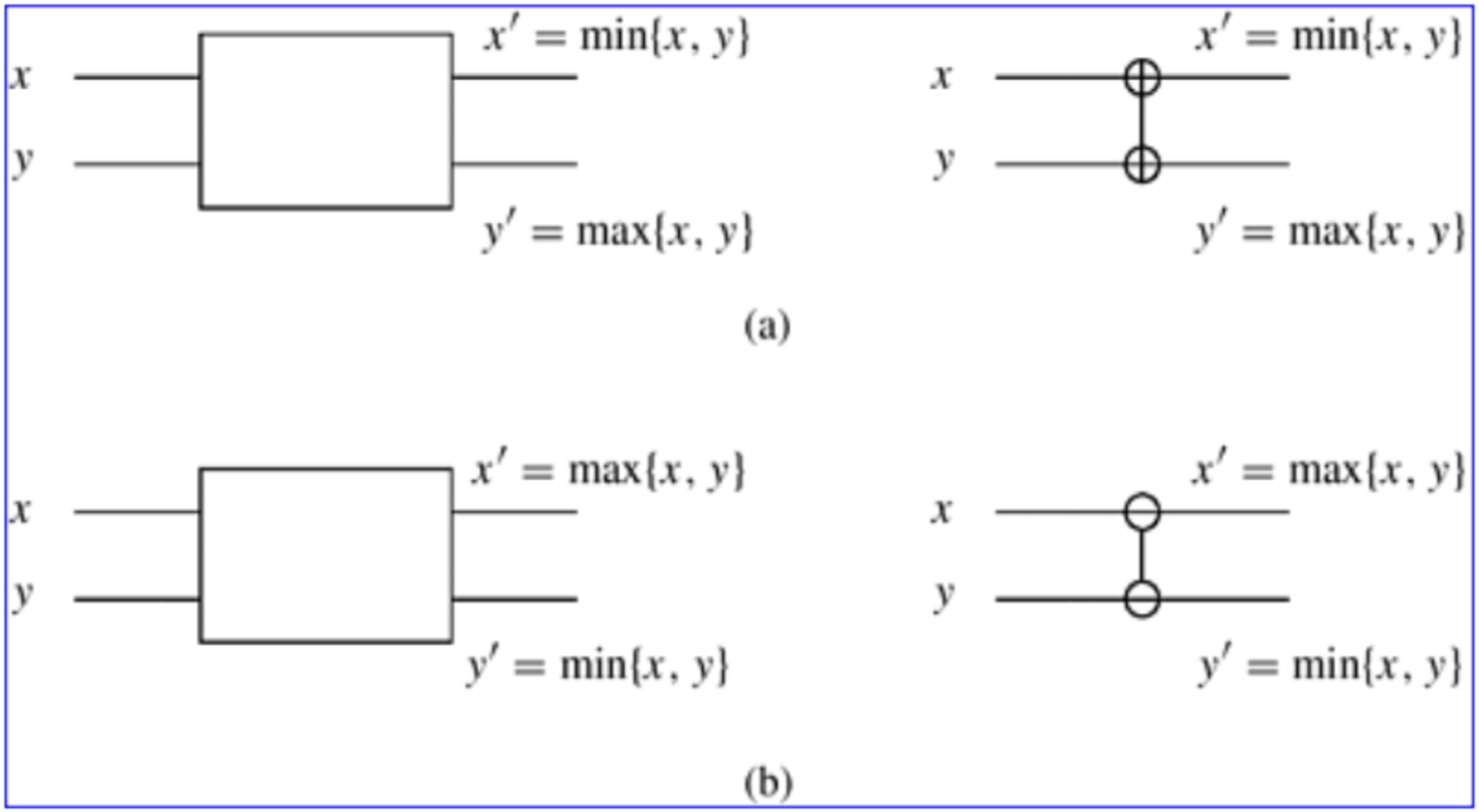
- Mạng sắp xếp: Là mạng các nút so sánh được thiết kế theo 1 kiến trúc nào đó.



CÁC BỘ SO SÁNH - COMPARATOR

- Các nút mạng chỉ thực hiện chức năng so sánh và đổi chỗ
- Phân loại:
  - Bộ so sánh tăng (increasing comparator)
    - y1 = min(x1, x2) và
    - y2 = max(x1, x2)
  - Bộ so sánh giảm (decreasing comparator)
    - y1 = max(x1, x2) và
    - y2 = min (x1, x2)

CÁC BỘ SO SÁNH - COMPARATOR



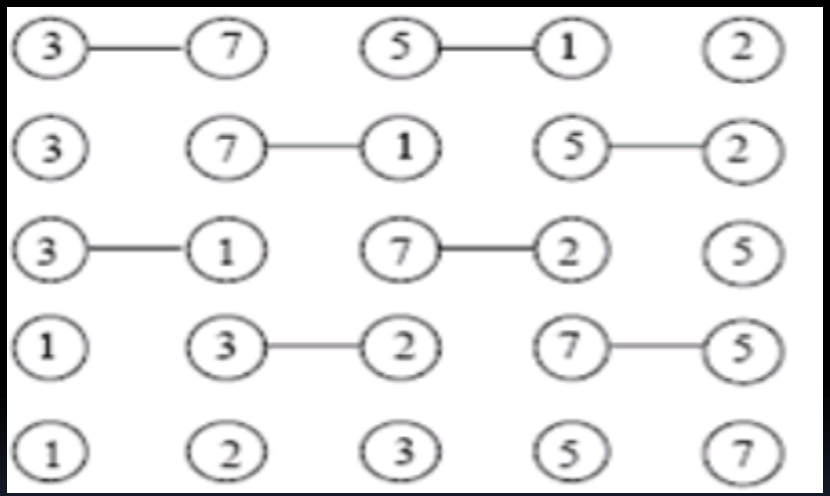
Hình a: increasing comparator; Hình b: decreasing comarator

3. SORTING

SẮP XẾP

- Phát biểu bài toán:
  - Input: Cho dãy A[1..n]
  - Output: sắp xếp dãy A tăng (giảm) dần
- Thuật giải tuần tự
  - Thuật giải Odd-even
  - Thuật giải quick sort
- Thuật giải song song
  - Song song hóa odd-even, quick sort

Ý TƯỞNG ODD - EVEN



SONG SONG HÓA ODD - EVEN

```
input    : A[1..n] bất kỳ.
output   : dãy A đã sắp xếp
begin
      for k = 1 to n do
            if k odd then
                  for i = 1 to n/2 do in parallel
                        swap(A[2k-1], A[2k]);
                  end parallel
            else
                  for i = 1 to n/2 -1 do in parallel
                        swap(A[2k], A[2k+1]);
                  end parallel
            end if
      end for
end
```

THUẬT TOÁN QUICK SORT

- Ý tưởng thuật toán:
  - Chọn phần tử làm key
  - Phân chia dãy thành 2 phần sao cho :
    - Các phần tử nằm ở bên trái có giá trị nhỏ hơn key
    - Các phần tử nằm bên phải có giá trị lớn hơn hoặc bằng key
  - Thực hiện đệ quy từng phần

THUẬT GIẢI TUẦN TỰ QUICK SORT

```
void QuickSort(int* A, int left, int right)
{
      int  x,k,i;
      if(left < right)
      {
            x     =     A[left];
            k     =     left;
            for(i = left+1; i <= right; i++)
            {
                  if(A[i] <= x)
                  {
                        k     =     k+1;
                        Swap(A[k],A[i]);
                  }
            }
            Swap(A[left],A[k]);
            QuickSort(A,left,k-1);
            QuickSort(A,k+1,right);
      }
}
```

SONG SONG HÓA QUICK SORT

- A[1..n] => Qtree thỏa mãn:
  - Các nút thuộc cây con bên trái có giá trị nhỏ hơn hoặc bằng gốc
  - Các nút thuộc cây con bên phải có giá trị lớn hơn giá trị tại gốc
- Duyệt theo thứ tự giữa InOrder.

SONG SONG HÓA QUICK SORT

- So sánh giá trị tương ứng với chỉ số BXL và local_pivot.
- Các nút có giá trị <= giá trị local_pivot nằm bên trái local_pivot
- Các nút có giá trị > giá trị local_pivot nằm bên phải local_pivot

SONG SONG HÓA QUICK SORT

- Mỗi nút bất kỳ A[i]:
  - L[i] là chỉ số nút con trái của A[i]
  - R[i] là chỉ số nút con phải của A[i]
- Cách xác định L[i] và R[i] ngẫu nhiên
  - L[i] chọn từ phần tử <= A[i]
  - R[i] chọn từ các phần tử > A[i]

VÍ DỤ

i	0	1	2	3	4	5	6	7
A	33	21	13	54	82	40	33	72
Left								
Right								
Local_pivot								

- Root = 3

i	0	1	2	3	4	5	6	7
A	33	21	13	**54**	82	40	33	72
Left								
Right								
Local_pivot	3	3	3		3	3	3	3

VÍ DỤ

- L[3]=A[0]
- R[3]=A[4]

i	**0**	1	2	**3**	**4**	5	6	7
A	33	21	13	**54**	82	40	33	72
Left				**0**				
Right				**4**				
Local_pivot	3	**0**	**0**		3	**0**	**0**	**4**

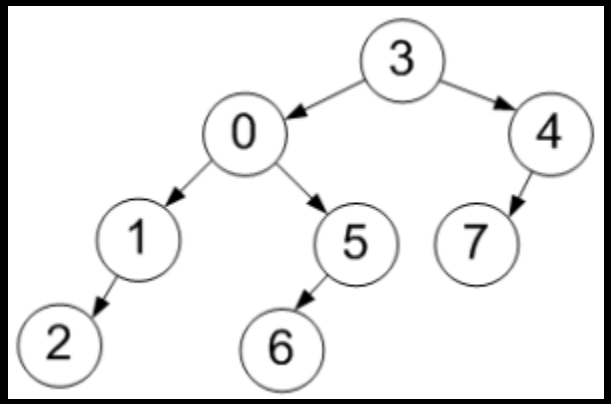
VÍ DỤ

i	**0**	**1**	2	**3**	**4**	**5**	6	**7**
A	33	21	13	54	82	40	33	72
Left	**1**			0	**7**			
Right	**5**			4				
Local_pivot	3	0	**1**		3	0	**5**	4

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
A	33	21	13	**54**	82	40	33	72
Left	1	**2**		0	7	**6**		
Right	5			4				
Local_pivot	3	0	1		3	0	5	4

SONG SONG HÓA QUICK SORT

- Kết quả cây nhị phân như sau



- Nếu ta duyệt cây theo thứ tự giữa InOrder thì ta sẽ có dãy chỉ số như sau: 2 – 1 – 0 – 6 – 5 – 3 – 7 – 4
- Tương ứng là: 13 – 21 – 33 – 33 – 40 – 54 – 72 – 82

THUẬT TOÁN

```
procedure BuildTree_QuickSort(A,n)
begin
        for i = 0 to n-1 do in parallel
                flag[i]             =       true;
                root                =       i;
                local_pivot         =       root;
                flag[root]  =       false;
        end parallel

        while ( OR flag[i], i =0..n-1)
        begin
                for i = 0 to n-1 do in parallel
                        if (flag[i]) then
                                if (A[i] < A[local_pivot]) then
                                        Left[local_pivot]               = i;
                                        flag[Left[local_pivot]]         = false;
                                        if i <> Left[local_pivot] then
                                                local_pivot         = Left[local_pivot];
                                        end if;
                                else
                                        Right[local_pivot]      =       i;
                                        flag[Right[local_pivot]]=       false;
                                        if i <> Right[local_pivot] then
                                                local_pivot = Right[local_pivot];
                                        end if;
                                end if;
                        end if;
                end parallel
        end while;
end
```