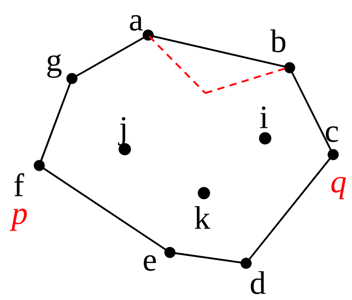
*definition of a convex set?
*how to check a convex-hull edge?
*how to check a convex-hull vertex?

Problem SM-5: Convex hull
Input: a set $S = \{v_1, v_2, \ldots, v_n\}$ of n points
Output: The convex hull $CH(S)$ of S (clockwise)
Model: CREW PRAM of n processors



sorted by x-coordinates

S = {f, g, j, a, e, k, d, i, b, c}

$CH(S)$ = {f, g, a, b, c, d, e}

$UH(S)$ = {f, g, a, b, c}

$LH(S)$ = {c, d, e, f}

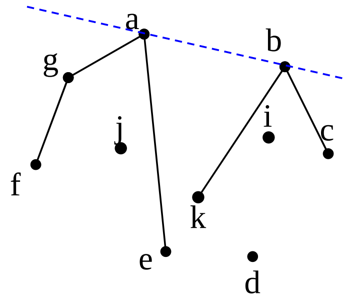
(1) Sort the points in S by their x coordinates. After the sorting, we have $x(v_1) < x(v_2) < \ldots < x(v_n)$.

Remark: Sorting n numbers can be done in $O(\log n)$ time on the EREW PRAM of n processors. (R. Cole, "Parallel merge sort," *SIAM Journal on Computing*, vol. 17, no. 4, pp. 770-785, 1988.)

(2) Let p and q be the points of the smallest and the largest x coordinates, respectively. Clearly, p and q belong to $CH(S)$ and partition $CH(S)$ into an upper hull $UH(S)$ and a lower hull $LH(S)$. In the following, we concentrate on determining $UH(S)$.

For simplicity, assume that no two points have the same x or y coordinates and $n=2^k$.

(3) **Algorithm** $UH(S)$



S_1 = {f, g, j, a, e}
S_2 = {k, d, i, b, c}

$UH(S_1)$ = {f, g, a, e}
$UH(S_2)$ = {k, b, c}
upper common tangent: (a, b)

$UH(S)$ = {f, g, a, b, c}

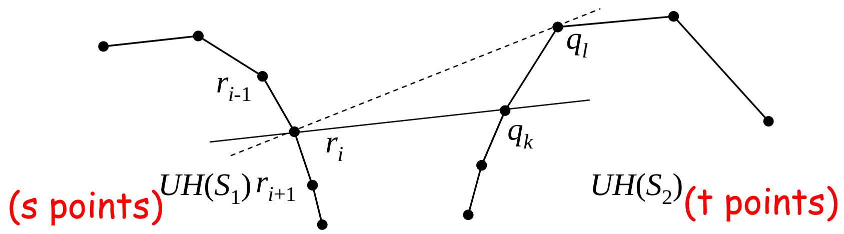
step 1: If $|S| \leq 4$, then use a brute-force method to determine $UH(S)$, and **exit**.

step 2: Let $S_1 = \{v_1, v_2, \ldots, v_{n/2}\}$ and $S_2 = \{v_{n/2+1}, v_{n/2+2}, \ldots, v_n\}$. Recursively, compute $UH(S_1)$ and $UH(S_2)$ in parallel.

step 3: Find the *upper common tangent* between $UH(S_1)$ and $UH(S_2)$, and deduce $UH(S)$.

(4) Let $UH(S_1) = \{r_1, r_2, \ldots, r_s\}$ and $UH(S_2) = \{q_1, q_2, \ldots, q_t\}$.

Remark: Given a point r_i and a point q_k, we can determined in $O(1)$ sequential time whether $l < k$, $l = k$, or $l > k$, where r_iq_l be the tangent from a point r_i to $UH(S_2)$.



In this example, we can conclude that $l > k$.

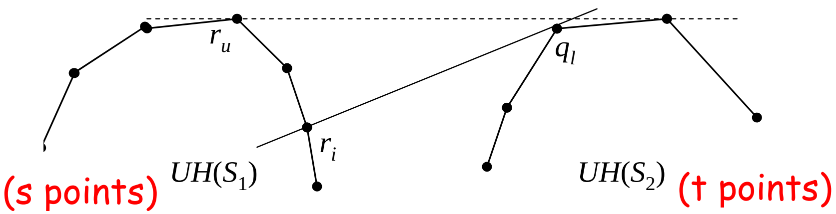
Remark: In $O(1)$ time, the tangent from a point r_i to $UH(S_2)$ can be obtained on the CREW PRAM of $t^{1/2}$ processors.

(parallel *m*-way search)

step 1: From the points $q_{t^{1/2}}, q_{2\times t^{1/2}}, \ldots, q_{t^{1/2}\times t^{1/2}}$, determine the index m that satisfies $(m-1)\times t^{1/2} < l \le m\times t^{1/2}$.

step 2: From the points $q_{m\times t^{1/2}+1}, q_{m\times t^{1/2}+2}, \ldots, q_{m\times t^{1/2}+t^{1/2}}$, determine the index l.

Remark: Let $(r_u q_v)$ be the upper tangent of $UH(S_1)$ and $UH(S_2)$. Given a point r_i and $r_i q_l$ (the tangent from r_i to $UH(S_2)$). We can determine in $O(1)$ sequential time whether $u<i$, $u=i$, or $u>i$.



In this example, we can conclude that $u < i$.

$\sqrt{s}\times\sqrt{t} \le \frac{s+t}{2} < s+t \le n$

Remark: The upper tangent of $UH(S_1)$ and $UH(S_2)$ can be determined in $O(1)$ time on a CREW PRAM of $s+t$ processors.

step 1: For each point $r_{i\times s^{1/2}}$, determine the tangent to $UH(S_2)$. ($s^{1/2}\times t^{1/2}$ processors) Then, determine the index m that satisfies $(m-1)\times s^{1/2} < u \le m\times s^{1/2}$.

step 2: From the points $r_{m\times s^{1/2}+1}, r_{m\times s^{1/2}+2}, \ldots, r_{m\times s^{1/2}+s^{1/2}}$, determine the index u. (Also, index v)

???

$T_s(n) = 2T(n/2) + O(\lg^2 n) + O(n)$

(5) * $T(n)=O(1)+T(n/2)=O(\log n)$

O(1) in parallel

* *Divide and Conquer*: [1] partition the input into the same subproblems of almost equal sizes, [2] solve recursively the subproblems in parallel, and [3] combine the solutions of subproblems.

Problem SM-6: Merging
Input: two sorted sequences $A[1\ldots n]$ and $B[1\ldots n]$
Output: a sorted sequence $C[1\ldots 2n]$
Model: CREW PRAM of $n/\log n$ processors ($n = 2^m$)

Remark: This problem can be solved in $O(n)$ sequential time. SM-12a

Let $R_A[i]$ be the rank of $A[i]$ in B (i.e., the number of values $\le A[i]$ in B.) Define $R_B[i]$ similarly. The merge problem can be viewed as that of determining $R_A[1\ldots n]$ and $R_B[1\ldots n]$. For simplicity, assume that all elements in A and B are distinct. SM-12b

$n = 12$, $m = \lfloor \log n \rfloor = 3$, $p = n/m = 4$

A:	1	5	7	9	11	25	27	30	40	42	47	51
B:	2	4	10	13	14	18	19	20	22	24	29	49

initially

A:	1	5	7	9	11	25	27	30	40	42	47	51
R_A			2			10			11			12
B:	2	**4**	10	13	14	18	19	20	22	**24**	**29**	**49**

$O(\log n)$

stage 1

SM-13a

A_1 A_2 A_3 A_4

A:	1	5	7	9	11	25	27	30	40	42	47	51
R_A	**0**	**2**	2			10	**10**	**11**	11	**11**	**11**	12
B:	2	4	10	13	14	18	19	20	22	24	29	49
R_B	**1**	**1**									**7**	**11**

$O(\log n)$

B_1 B_2 step 1 of stage 2 B_3 B_4

A:	1	5	7	**9**	**11**	25	27	30	40	42	47	51
R_A	0	2	2			10	10	11	11	11	11	12
B:	2	4	**10**	13	14	**18**	19	20	**22**	24	29	49
R_B	1	1	**4**			**5**			**5**		7	11

step 2 of stage 2

A:	1	5	7	**9**	**11**	25	27	30	40	42	47	51
R_A	0	2	2	2	3	10	10	11	11	11	11	12
B:	2	4	**10**	13	14	**18**	19	20	**22**	24	29	49
R_B	1	1	**4**	5	5	**5**	5	5	**5**	5	7	11

step 3 of stage 2

bug !!! how to fix ???

SM-13b

stage 1: (partition) Determine $R_A[i \times m]$, $1 \le i \le n/m$. Then, partition A and B into A_i's and B_i's respectively, where $A_i = \{A[(i-1) \times m+1], \ldots, A[i \times m]\}$ and $B_i = \{B[R_A[(i-1) \times m]+1], \ldots, B[R_A[i \times m]]\}$.

SM-14a

say, $\le 2\lg n$

stage 2: For each pair of A_i and B_i, $1 \le i \le n/m$, if $|B_i| = O(\log n)$, then rank all elements in A_i and B_i sequentially. Otherwise, apply the partition technique used in **stage 1** to partition A_i and B_i into $A_{i,j}$'s and $B_{i,j}$'s of size $O(\log n)$ (in this case, B_i plays the role of A and A_i plays the role of B). And then, for each pair of $A_{i,j}$ and $B_{i,j}$, rank all elements in $A_{i,j}$ and $B_{i,j}$ sequentially.

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

* *Partition*:(1) break up the given problem into p independent subproblems of almost equal size, and then (2) solve the p subproblems concurrently, where p is the number of processors available.

Remark: The main work in the divide-and-conquer strategy usually lies in the merging of sub-solutions, whereas the main work in the partition strategy lies in carefully partitioning the problem into independent sub-problems.

Problem SM-7: Inserting a sorted sequence into a 2-3 tree
Input: a 2-3 tree T holding n items $a_1 < a_2 < \ldots < a_n$, and a sequence $b_1 < b_2 < \ldots < b_k$ (Assume k is much smaller that n.)

(the balanced ST that is easiest to implement)

Output: the 2-3 tree obtained from T by inserting $b_1, b_2, \ldots, b_k$

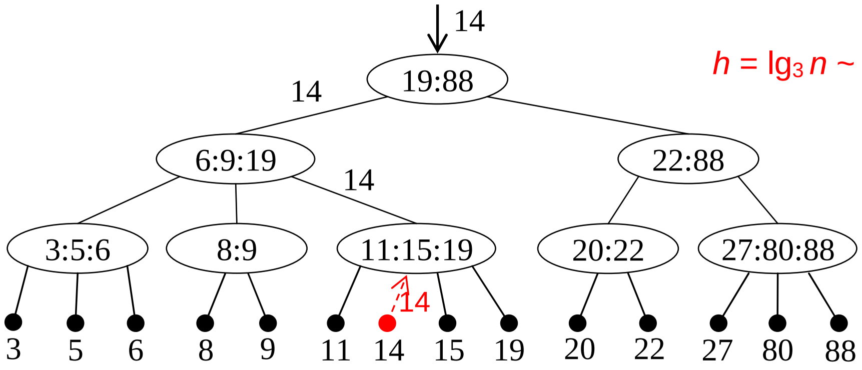
Model: CREW PRAM of k processors

(1) A 2-3 tree is a rooted tree in which (i) each internal node has either 2 or 3 children, and (ii) every path from the root to a leaf is of the same length. Clearly, if the number of leaves is n the height of the tree is $O(\log n)$. $\log_3 n \sim \log_2 n$

(2) A sorted list $a_1, a_2, \ldots, a_n$, can be represented by a 2-3 tree T, where the leaves hold the data items in a left-to-right order. And, an internal node will hold the largest data items stored in its subtrees. Searching for a data item on T can be performed in $O(\log n)$ time.

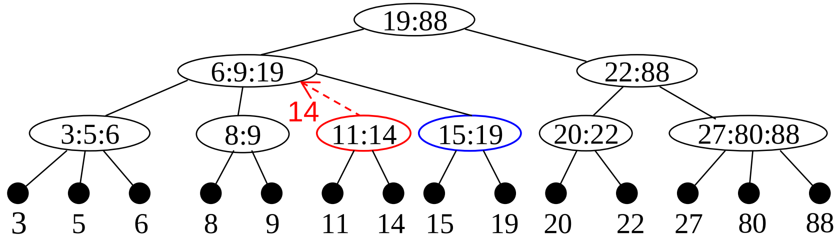
(3) Inserting an item b into a 2-3 tree in $O(\log n)$ time

step 1: (1) locate 14, (2) create a leaf to hold 14, and (3) then insert it into <11:15:19>.

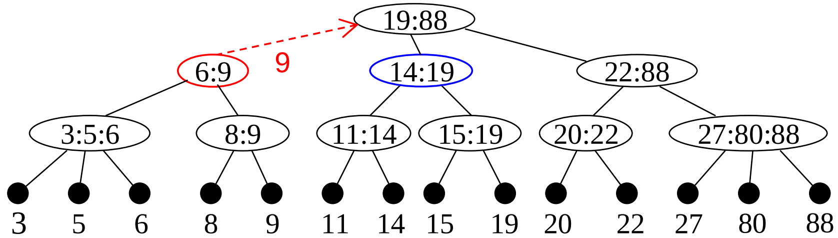


$h = \lg_3 n \sim \lg_2 n$

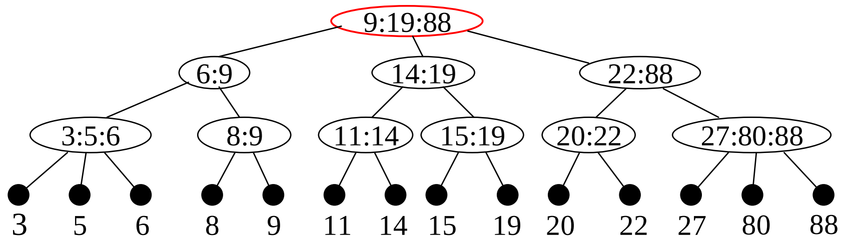
step 2: (1) split the node <11:15:19> into two nodes <11:14> and <15:19>, and then (2) insert <11:14> into <6:9:19>.



step 3: (1) split the node <6:9:19> into two nodes <6:9> and <14:19>, and then (2) insert <6:9> into <19:88>.

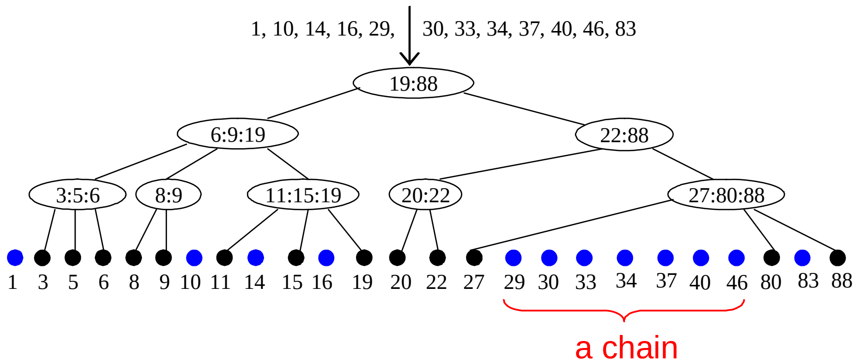


* after step 3



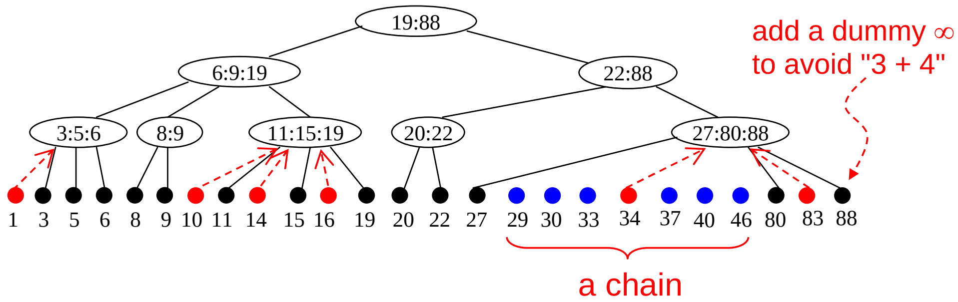
(4) Inserting a sorted sequence into a 2-3 tree

step 1: (1) locate b_i's, and (2) create leaves to hold them.

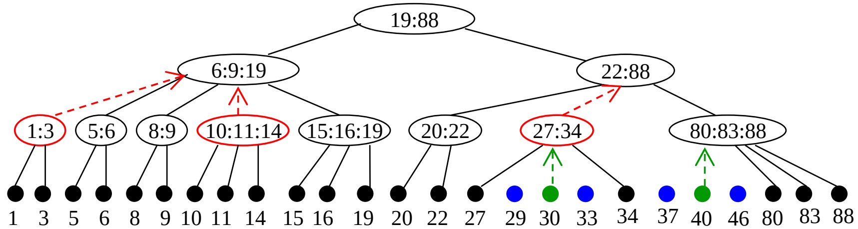


step 2: Let us call a **chain** the order set of elements among the b_i's that have to fit between two consecutive leaves of T. Insert the medians of all **chains** into appropriate parents. (An internal node at height 1 may now have at most six children.)

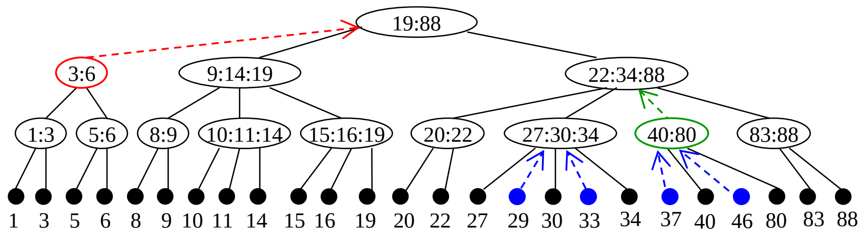
insert a dummy ∞



step 3: (1) split nodes at height 1 that have more than 3 children and insert them into appropriate parents. (An internal node at height 2 may now have at most six children.) And, (2) then start the inserting of the medians of current chains.



step 4: (1) split nodes at height 1 and 2 that have more than 3 children and insert them into appropriate parents. And, then (2) start the inserting of the medians of current chains.



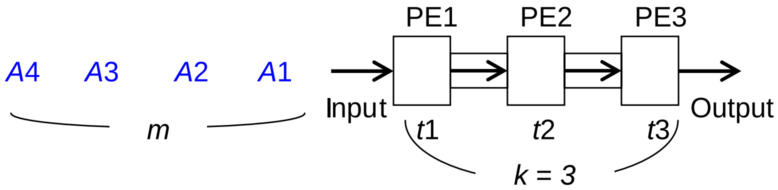
* $T(n, k) = O(\log n + \log k)$

* *Pipelining*: Breaking up a task into a sequence of subtasks t_1, t_2, …, t_k, such that, once t_1 is completed, the sequence corresponding to a new task can begin and can proceed at the same rate as the previous task.

align to the longest delay (2, 3, 1000 => 1000)

pipeline

SM-19a



time : $3m \Rightarrow 3 + (m - 1)$
time : $km \Rightarrow k + (m - 1)$ (Speedup = k)

* Implementation details? SM-19e
* Q: How to adapt the algorithm to EREW PRAM ?

* sequential: O(n)
* EREW: bal. binary tree method, n PEs, O(lg n) time

Problem SM-8: Finding maximum
Input: $A[1\ldots n]=\{3, 7, 8, 4\}$ (Assume all elements are distinct.)
Output: **max**$\{ A[1], A[2], \ldots, A[n] \}$
Model: CRCW PRAM of n^2 processors
(For simplicity, we assume that each processor is indexed with a unique pair of (i, j).)

SM-20a

step 1: (1) $P_{i,1}$ sets $MARK[i]$ as 0. (2) $P_{i,j}$ sets $C[i, j]$ as 0 if $(A[i] \ge A[j])$ and 1 otherwise.

$P_{1,3}$ (comparison matrix)

$$MARK[i] \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad C[i,j] \quad \begin{array}{c|cccc} & 3 & 7 & 8 & 4 \\ \hline 3 & 0 & 1 & 1 & 1 \\ 7 & 0 & 0 & 1 & 0 \\ 8 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 \end{array}$$

step 2: (1) $P_{i,j}$ sets $MARK[i] = 1$ if $C[i, j] = 1$. (2) $P_{i,1}$ sets $max = A[i]$ if $MARK[i] = 0$.

(comparison matrix)

$$\text{max } \boxed{8} \qquad MARK[i] \quad \begin{array}{c|c} P_1 & 1 \\ P_2 & 1 \\ P_3 & \underline{0} \\ P_4 & 1 \end{array} \qquad C[i,j] \quad \begin{array}{c|cccc} & 3 & 7 & 8 & 4 \\ \hline 3 & 0 & 1 & 1 & 1 \\ 7 & 0 & 0 & 1 & 0 \\ \underline{8} & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 \end{array}$$

* OR operation can be done in $O(1)$ time using n PEs.