Algorithm 9.1  Parallel formulation of bitonic sort on a hypercube with \( n = 2^d \) processes. In this algorithm, \( label \) is the process's label and \( d \) is the dimension of the hypercube.
Algorithm 9.2  Sequential bubble sort algorithm.
1. procedure ODD-EVEN(n)
2. begin
3.     for i := 1 to n do
4.         begin
5.             if i is odd then
6.                 for j := 0 to n/2 − 1 do
7.                     compare-exchange(a_{2j+1}, a_{2j+2});
8.             if i is even then
9.                 for j := 1 to n/2 − 1 do
10.                compare-exchange(a_{2j}, a_{2j+1});
11.         end for
12.     end ODD-EVEN

Algorithm 9.3  Sequential odd-even transposition sort algorithm.
Algorithm 9.4  The parallel formulation of odd-even transposition sort on an $n$-process ring.
Algorithm 9.5  The sequential quicksort algorithm.
Algorithm 9.6  The binary tree construction procedure for the CRCW PRAM parallel quicksort formulation.

1. procedure BUILD_TREE (A[1...n])
2. begin
3.   for each process i do
4.     begin
5.       root := i;
6.       parent_i := root;
7.       leftchild[i] := rightchild[i] := n + 1;
8.     end for
9.   repeat for each process i ≠ root do
10.    begin
11.      if (A[i] < A[parent_i]) or
12.         (A[i] = A[parent_i] and i < parent_i) then
13.         begin
14.             leftchild[parent_i] := i;
15.             if i = leftchild[parent_i] then exit
16.             else parent_i := leftchild[parent_i];
17.         end for
18.     else
19.         begin
20.             rightchild[parent_i] := i;
21.             if i = rightchild[parent_i] then exit
22.             else parent_i := rightchild[parent_i];
23.         end else
24.     end repeat
25. end BUILD_TREE
procedure ENUM_SORT (n)
begin
for each process $P_{i,j}$ do
  $C[j] := 0$;
for each process $P_{i,j}$ do
    $C[j] := 1$;
  else
    $C[j] := 0$;
for each process $P_{i,j}$ do
end ENUM_SORT

Algorithm 9.7 Enumeration sort on a CRCW PRAM with additive-write conflict resolution.
1. procedure RADIX_SORT(A, r)
2. begin
3. for i := 0 to b/r − 1 do
4. begin
5. offset := 0;
6. for j := 0 to 2^r − 1 do
7. begin
8. flag := 0;
9. if the i th least significant r-bit block of A[P_k] = j then
10. flag := 1;
11. index := prefix_sum(flag)
12. if flag = 1 then
13. rank := offset + index;
14. offset := parallel_sum(flag);
15. endfor
16. each process P_k send its element A[P_k] to process P_rank;
17. endfor
18. end RADIX_SORT

Algorithm 9.8  A parallel radix sort algorithm, in which each element of the array A[1 . . . n] to be sorted is assigned to one process. The function prefix_sum() computes the prefix sum of the flag variable, and the function parallel_sum() returns the total sum of the flag variable.
1. procedure HYPERCUBE.QUICKSORT \( (B, n) \)
2. begin
3. \( id := \) process’s label;
4. for \( i := 1 \) to \( d \) do
5. begin
6. \( x := \) pivot;
7. partition \( B \) into \( B_1 \) and \( B_2 \) such that \( B_1 \leq x < B_2 \);
8. if \( i^{th} \) bit is 0 then
9. begin
10. send \( B_2 \) to the process along the \( i^{th} \) communication link;
11. \( C := \) subsequence received along the \( i^{th} \) communication link;
12. \( B := B_1 \cup C \);
13. endif
14. else
15. send \( B_1 \) to the process along the \( i^{th} \) communication link;
16. \( C := \) subsequence received along the \( i^{th} \) communication link;
17. \( B := B_2 \cup C \);
18. endelse
19. endfor
20. sort \( B \) using sequential quicksort;
21. end HYPERCUBE.QUICKSORT

Algorithm 9.9 A parallel formulation of quicksort on a \( d \)-dimensional hypercube. \( B \) is the \( n/p \)-element subsequence assigned to each process.