Module 2

Sorting is a process of arranging elements in a group in a particular order, i.e., ascending order, descending order, alphabetic order, etc. Sorting a list of elements is a very common operation. A sequential sorting algorithm may not be efficient enough when we have to sort a huge volume of data. Therefore, parallel algorithms are used in sorting.

Enumeration Sort

Enumeration sort is a method of arranging all the elements in a list by finding the final position of each element in a sorted list. It is done by comparing each element with all other elements and finding the number of elements having smaller value.

Therefore, for any two elements, a_i and a_j any one of the following cases must be true −

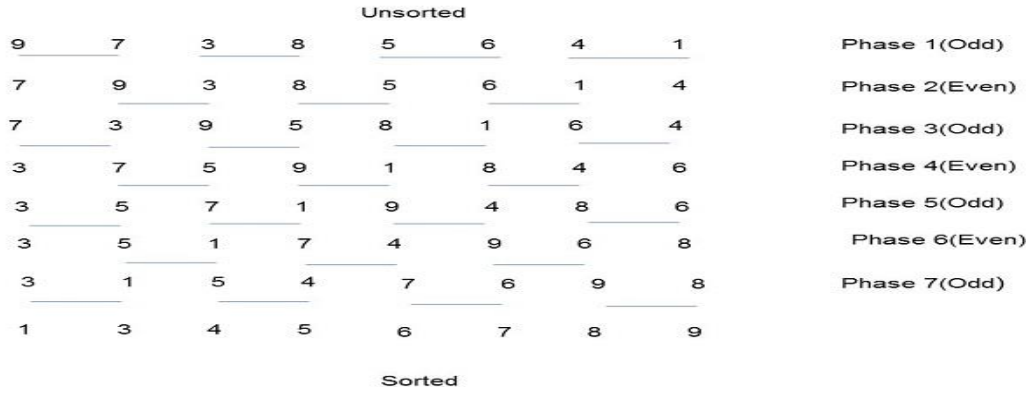
- $a_i < a_j$
- $a_i > a_j$
- $a_i = a_j$

Algorithm

```
procedure ENUM_SORTING (n)

begin
   for each process P1,j do
      C[j] := 0;

   for each process Pi, j do

      if (A[i] < A[j]) or A[i] = A[j] and i < j) then
         C[j] := 1;
      else
         C[j] := 0;

   for each process P1, j do
      A[C[j]] := A[j];

end ENUM_SORTING
```

Odd-Even Transposition Sort

Odd-Even Transposition Sort is based on the Bubble Sort technique. It compares two adjacent numbers and switches them, if the first number is greater than the second number to get an ascending order list. The opposite case applies for a descending order series. Odd-Even transposition sort operates in two phases − **odd phase** and **even phase**. In both the phases, processes exchange numbers with their adjacent number in the right.

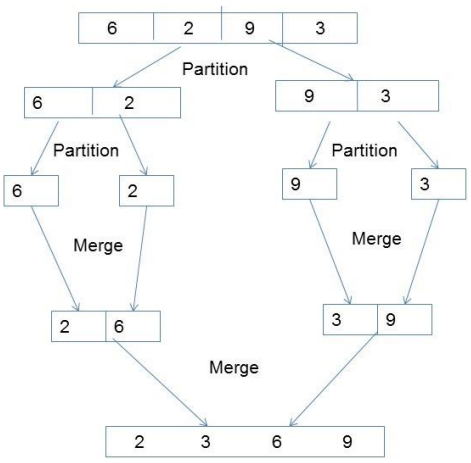


Algorithm

```
procedure ODD-EVEN_PAR (n)

begin
   id := process's label

   for i := 1 to n do
   begin

      if i is odd and id is odd then
         compare-exchange_min(id + 1);
      else
         compare-exchange_max(id - 1);

      if i is even and id is even then
         compare-exchange_min(id + 1);
      else
         compare-exchange_max(id - 1);

   end for

end ODD-EVEN_PAR
```

Parallel Merge Sort

Merge sort first divides the unsorted list into smallest possible sub-lists, compares it with the adjacent list, and merges it in a sorted order. It implements parallelism very nicely by following the divide and conquer algorithm.



Algorithm

```
procedureparallelmergesort(id, n, data, newdata)

begin
   data = sequentialmergesort(data)

      for dim = 1 to n
         data = parallelmerge(id, dim, data)
      endfor

   newdata = data
end
```

Hyper Quick Sort

Hyper quick sort is an implementation of quick sort on hypercube. Its steps are as follows −

- Divide the unsorted list among each node.
- Sort each node locally.
- From node 0, broadcast the median value.
- Split each list locally, then exchange the halves across the highest dimension.
- Repeat steps 3 and 4 in parallel until the dimension reaches 0.

Algorithm

```
procedure HYPERQUICKSORT (B, n)
begin

   id := process's label;

   for i := 1 to d do
      begin
      x := pivot;
      partition B into B1 and B2 such that B1 ≤ x < B2;
      if ith bit is 0 then

      begin
         send B2 to the process along the ith communication link;
         C := subsequence received along the ith communication link;
         B := B1 U C;
      endif

      else
         send B1 to the process along the ith communication link;
         C := subsequence received along the ith communication link;
         B := B2 U C;
         end else
      end for

   sort B using sequential quicksort;

end HYPERQUICKSORT
```

Bitonic Merge Sort

Bitonic Sort is a classic parallel algorithm for sorting.

- Bitonic sort does O (n Log ^{2}n) comparisons.
- The number of comparisons done by Bitonic sort are more than popular sorting algorithms like Merge Sort [does O(nLogn) comparisons], but Bitonice sort is better for parallel implementation because we always compare elements in predefined sequence and the sequence of comparison doesn't depend on data. Therefore, it is suitable for implementation in hardware and parallel processor array.

To understand Bitonic Sort, we must first understand what is Bitonic Sequence and how to make a given sequence Bitonic.

Bitonic Sequence

A sequence is called Bitonic if it is first increasing, then decreasing. In other words, an array arr[0..n-i] is Bitonic if there exists an index i where 0<=i<=n-1 such that

x0 <= x1 …..<= xi and xi >= xi+1….. >= xn-1

1. A sequence, sorted in increasing order is considered Bitonic with the decreasing part as empty. Similarly, decreasing order sequence is considered Bitonic with the increasing part as empty.
2. A rotation of Bitonic Sequence is also bitonic.

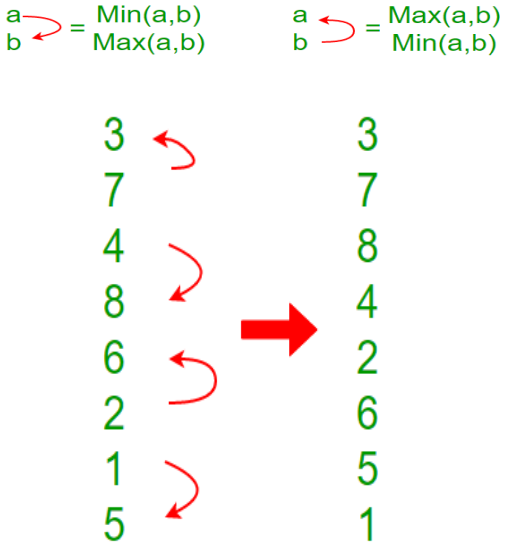
How to form a Bitonic Sequence from a random input?

We start by forming 4-element bitonic sequences from consecutive 2-element sequence. Consider 4-element in sequence x0, x1, x2, x3. We sort x0 and x1 in ascending order and x2 and x3 in descending order. We then concatenate the two pairs to form a 4 element bitonic sequence.
Next, we take two 4 element bitonic sequences, sorting one in ascending order, the other in descending order (using the Bitonic Sort which we will discuss below), and so on, until we obtain the bitonic sequence.

Example:

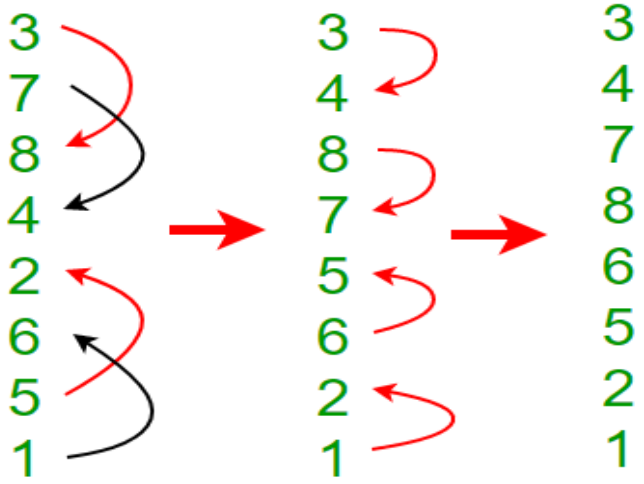
Convert the following sequence to bitonic sequence: 3, 7, 4, 8, 6, 2, 1, 5

Step 1: Consider each 2-consecutive elements as bitonic sequence and apply bitonic sort on each 2- pair elements. In next step, take two 4 element bitonic sequences and so on.



Note: x0 and x1 are sorted in ascending order and x2 and x3 in descending order and so on

Step 2: Two 4 element bitonic sequences: **A** (3,7,8,4) and **B** (2,6,5,1) with comparator length as 2



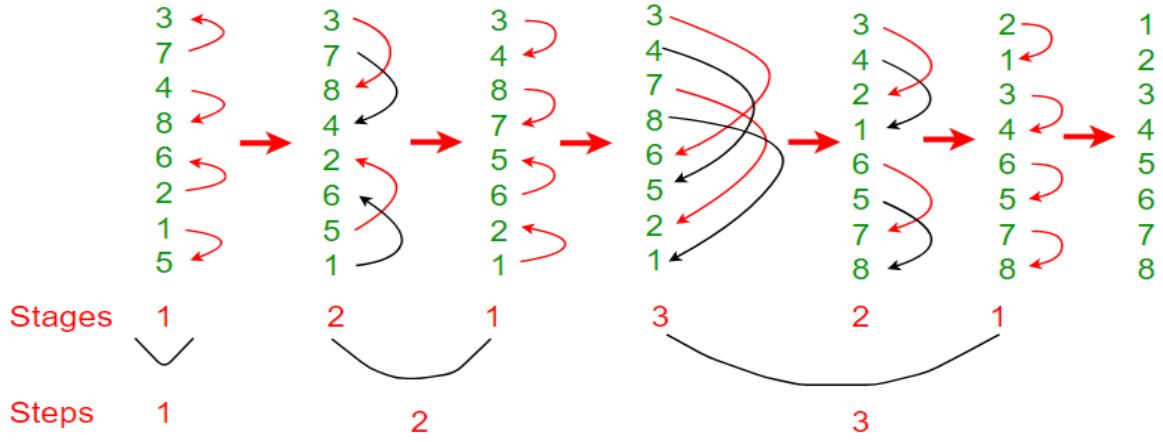
After this step, we'll get Bitonic sequence of length 8.

3, 4, 7, 8, 6, 5, 2, 1

Bitonic Sorting

It mainly involves two steps.

1. Forming a bitonic sequence (discussed above in detail). After this step we reach the fourth stage in below diagram, i.e., the array becomes {3, 4, 7, 8, 6, 5, 2, 1}
2. Creating one sorted sequence from bitonic sequence: After first step, first half is sorted in increasing order and second half in decreasing order. We compare first element of first half with first element of second half, then second element of first half with second element of second and so on. We exchange elements if an element of first half is smaller. After above compare and exchange steps, we get two bitonic sequences in array. See fifth stage in below diagram. In the fifth stage, we have {3, 4, 2, 1, 6, 5, 7, 8}. If we take a closer look at the elements, we can notice that there are two bitonic sequences of length n/2 such that all elements in first bitnic sequence {3, 4, 2, 1} are smaller than all elements of second bitonic sequence {6, 5, 7, 8}. We repeat the same process within two bitonic sequences and we get four bitonic sequences of length n/4 such that all elements of leftmost bitonic sequence are smaller and all elements of rightmost. See sixth stage in below diagram, arrays are {2, 1, 3, 4, 6, 5, 7, 8}. If we repeat this process one more time, we get 8 bitonic sequences of size n/8 which is 1. Since all these bitonic sequence are sorted and every bitonic sequence has one element, we get the sorted array.



Analysis of Bitonic Sort

To form a sorted sequence of length n from two sorted sequences of length n/2, log(n) comparisons are required (for example: log (8) = 3 when sequence size. Therefore, the number of comparisons T(n) of the entire sorting is given by:

$T(n) = \log(n) + T(n/2)$

The solution of this recurrence equation is

$T(n) = \log(n) + \log(n)-1 + \log(n)-2 + \ldots + 1 = \log(n) \cdot (\log(n)+1) / 2$

As, each stage of the sorting network consists of n/2 comparators. Therefore total? $(n \log^2 n)$ comparators.

Message Passing Interface (MPI) is a standardized and portable message-passing standard designed by a group of researchers from academia and industry to function on a wide variety of parallel computing architecture.

Parallel Virtual Machine (PVM) is a message passing system that enables a network of Unix computers to be used as a single distributed memory parallel computer. This network is referred to as the virtual machine.