Bitonic Sort

Background

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?

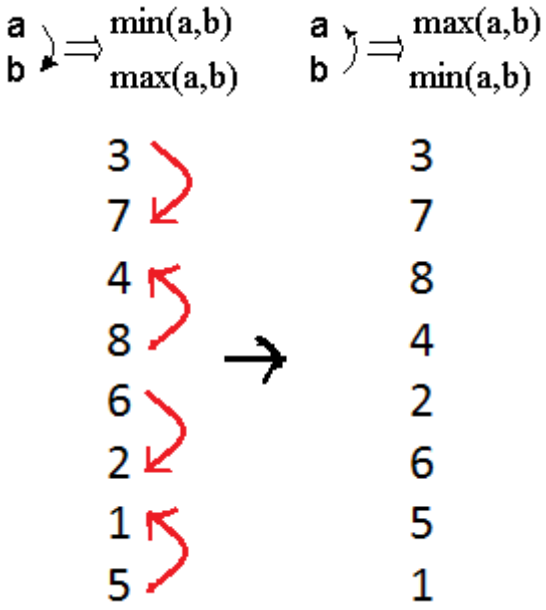
We start by forming 4-element bitonic sequences from consecutive 2-element sequence. Consider 4-element in sequence x0, x1, x2, x3. We sort x0 and x1 in ascending order and x2 and x3 in descending order. We then concatenate the two pairs to form a 4 element bitonic sequence.

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:

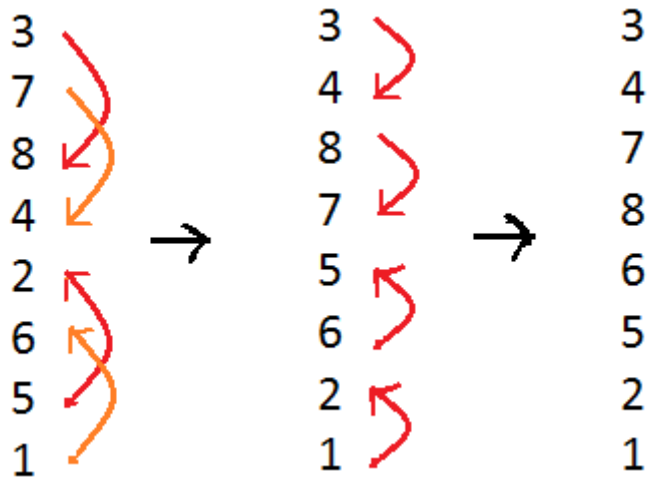
Convert the following sequence to bitonic sequence: 3, 7, 4, 8, 6, 2, 1, 5

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}

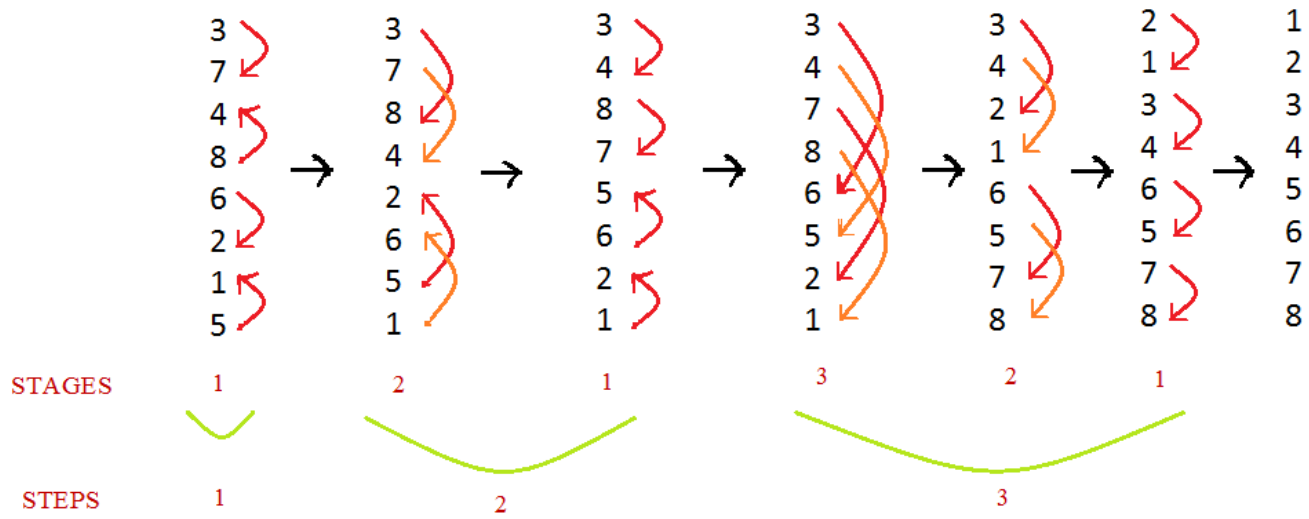
2. Creating one sorted sequence from bitonic sequence : After first step, first half is sorted in increasing order and second half in decreasing order.

   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 is {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.



Below are implementations of Bitonic Sort.

C++

```
/* C++ Program for Bitonic Sort. Note that this program
   works only when size of input is a power of 2. */
#include<bits/stdc++.h>
using namespace std;

/*The parameter dir indicates the sorting direction, ASCENDING
   or DESCENDING; if (a[i] > a[j]) agrees with the direction,
   then a[i] and a[j] are interchanged.*/
void compAndSwap(int a[], int i, int j, int dir)
{
    if (dir==(a[i]>a[j]))
        swap(a[i],a[j]);
}

/*It recursively sorts a bitonic sequence in ascending order,
  if dir = 1, and in descending order otherwise (means dir=0).
  The sequence to be sorted starts at index position low,
  the parameter cnt is the number of elements to be sorted.*/
```

```
void bitonicMerge(int a[], int low, int cnt, int dir)
{
    if (cnt>1)
    {
        int k = cnt/2;
        for (int i=low; i<low+k; i++)
            compAndSwap(a, i, i+k, dir);
        bitonicMerge(a, low, k, dir);
        bitonicMerge(a, low+k, k, dir);
    }
}

/* This function first produces a bitonic sequence by recursively
    sorting its two halves in opposite sorting orders, and then
    calls bitonicMerge to make them in the same order */
void bitonicSort(int a[],int low, int cnt, int dir)
{
    if (cnt>1)
    {
        int k = cnt/2;

        // sort in ascending order since dir here is 1
        bitonicSort(a, low, k, 1);

        // sort in descending order since dir here is 0
        bitonicSort(a, low+k, k, 0);

        // Will merge wole sequence in ascending order
        // since dir=1.
        bitonicMerge(a,low, cnt, dir);
    }
}

/* Caller of bitonicSort for sorting the entire array of
   length N in ASCENDING order */
void sort(int a[], int N, int up)
{
    bitonicSort(a,0, N, up);
}

// Driver code
int main()
{
    int a[]= {3, 7, 4, 8, 6, 2, 1, 5};
    int N = sizeof(a)/sizeof(a[0]);

    int up = 1;   // means sort in ascending order
    sort(a, N, up);

    printf("Sorted array: \n");
    for (int i=0; i<N; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Run on IDE

Java

```
/* Java program for Bitonic Sort. Note that this program
   works only when size of input is a power of 2. */
public class BitonicSort
{
    /* The parameter dir indicates the sorting direction,
       ASCENDING or DESCENDING; if (a[i] > a[j]) agrees
       with the direction, then a[i] and a[j] are
       interchanged. */
    void compAndSwap(int a[], int i, int j, int dir)
    {
        if ( (a[i] > a[j] && dir == 1) ||
             (a[i] < a[j] && dir == 0))
```

```
        {
            // Swapping elements
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }

    /* It recursively sorts a bitonic sequence in ascending
       order, if dir = 1, and in descending order otherwise
       (means dir=0). The sequence to be sorted starts at
       index position low, the parameter cnt is the number
       of elements to be sorted.*/
    void bitonicMerge(int a[], int low, int cnt, int dir)
    {
        if (cnt>1)
        {
            int k = cnt/2;
            for (int i=low; i<low+k; i++)
                compAndSwap(a,i, i+k, dir);
            bitonicMerge(a,low, k, dir);
            bitonicMerge(a,low+k, k, dir);
        }
    }

    /* This funcion first produces a bitonic sequence by
       recursively sorting its two halves in opposite sorting
       orders, and then  calls bitonicMerge to make them in
       the same order */
    void bitonicSort(int a[], int low, int cnt, int dir)
    {
        if (cnt>1)
        {
            int k = cnt/2;

            // sort in ascending order since dir here is 1
            bitonicSort(a, low, k, 1);

            // sort in descending order since dir here is 0
            bitonicSort(a,low+k, k, 0);

            // Will merge wole sequence in ascending order
            // since dir=1.
            bitonicMerge(a, low, cnt, dir);
        }
    }

    /*Caller of bitonicSort for sorting the entire array
      of length N in ASCENDING order */
    void sort(int a[], int N, int up)
    {
        bitonicSort(a, 0, N, up);
    }

    /* A utility function to print array of size n */
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver method
    public static void main(String args[])
    {
        int a[] = {3, 7, 4, 8, 6, 2, 1, 5};
        int up = 1;
        BitonicSort ob = new BitonicSort();
        ob.sort(a, a.length,up);
        System.out.println("\nSorted array");
        printArray(a);
    }
}
```

Run on IDE

Python

```python
# Python program for Bitonic Sort. Note that this program
# works only when size of input is a power of 2.

# The parameter dir indicates the sorting direction, ASCENDING
# or DESCENDING; if (a[i] > a[j]) agrees with the direction,
# then a[i] and a[j] are interchanged.*/
def compAndSwap(a, i, j, dire):
    if (dire==1 and a[i] > a[j]) or (dire==0 and a[i] > a[j]):
        a[i],a[j] = a[j],a[i]

# It recursively sorts a bitonic sequence in ascending order,
# if dir = 1, and in descending order otherwise (means dir=0).
# The sequence to be sorted starts at index position low,
# the parameter cnt is the number of elements to be sorted.
def bitonicMerge(a, low, cnt, dire):
    if cnt > 1:
        k = cnt/2
        for i in range(low , low+k):
            compAndSwap(a, i, i+k, dire)
        bitonicMerge(a, low, k, dire)
        bitonicMerge(a, low+k, k, dire)

# This funcion first produces a bitonic sequence by recursively
# sorting its two halves in opposite sorting orders, and then
# calls bitonicMerge to make them in the same order
def bitonicSort(a, low, cnt,dire):
    if cnt > 1:
          k = cnt/2
          bitonicSort(a, low, k, 1)
          bitonicSort(a, low+k, k, 0)
          bitonicMerge(a, low, cnt, dire)

# Caller of bitonicSort for sorting the entire array of length N
# in ASCENDING order
def sort(a,N, up):
    bitonicSort(a,0, N, up)

# Driver code to test above
a = [3, 7, 4, 8, 6, 2, 1, 5]
n = len(a)
up = 1

sort(a, n, up)
print ("\n\nSorted array is")
for i in range(n):
    print("%d" %a[i]),
```

Run on IDE

Output:

```
Sorted array:
1 2 3 4 5 6 7 8
```

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 $\Theta(n \log^2 n)$ comparators.

References:

1. https://www.youtube.com/watch?v=GEQ8y26blEY
2. http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm
3. https://en.wikipedia.org/wiki/Bitonic_sorter

This article is contributed by **Rahul Agarwal.** If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner Company Wise Coding Practice

Sorting

Recommended Posts:

Find whether an array is subset of another array | Added Method 3

(Login to Rate and Mark)

3.3 Average Difficulty : **3.3/5.0**
Based on **3** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!