Answer on Question#39537 - Programming - C++

Bitonic sort

```
#include <iostream>
#include "time.h"
#include "math.h"
#include <fstream>
#include <iomanip>
#include <string>

using namespace std;

void bitonic_sort_threaded(bool up, int x[], int left, int right, int k);
void bitonic_merge_threaded(bool up, int x[], int left, int right, int k);
void bitonic_sort(bool up, int x[], int left, int right);
void bitonic_merge(bool up, int x[], int left, int right);
void bitonic_compare(bool up, int x[], int left, int right);


int main(void)
{
   int n;
   cin >> n;
   int left = 0;
   int right = n - 1;

   int *A = new int[n];

   for (int i = 0; i < n; i++)
   {
      cin >> A[i];
   }

    bitonic_sort(true, A, left, (left + right) / 2);
    bitonic_sort(false, A, (left + right) / 2 + 1, right);
    bitonic_merge(true, A, left, right);

    for (int i = 0; i < n; i++)
    {
       cout << "\n" <<  A[i];
    }
```

```
    delete[] A;



   system("pause");
   return 0;
}



void bitonic_sort(bool up, int x[], int left, int right)
{
   if (right - left > 0)
   {
      bitonic_sort(true, x, left, (left + right) / 2);
      bitonic_sort(false, x, (left + right) / 2 + 1, right);
      bitonic_merge(up, x, left, right);
   }
}

void bitonic_merge(bool up, int x[], int left, int right)
{
   if (right - left > 0)
   {
      bitonic_compare(up, x, left, right);
      bitonic_merge(up, x, left, (left + right) / 2);
      bitonic_merge(up, x, (left + right) / 2 + 1, right);
   }
}

void bitonic_compare(bool up, int x[], int left, int right)
{
   int dist = (right - left + 1) / 2;
   for (int i = left; i < left + dist; i++)
   {
      if ((x[i] > x[i + dist]) == up)
      {
         int tmp = x[i];
         x[i] = x[i + dist];
         x[i + dist] = tmp;
      }
   }
}
```

Mergesort

```
#include <iostream>
#define ARRAY_SIZE 5

using namespace std;

void MergeSort(int arr[], int p, int r);
void Merge(int arr[], int p, int q, int r);

int main(void)
{
   int array[ARRAY_SIZE];
   int i = 0;
   cout << "\nEnter the Array elements : \n";
   for (i = 0; i < ARRAY_SIZE; i++)
   {
      cin >> array[i];
   }

   /*calling MergeSort()*/
   MergeSort(array, 0, ARRAY_SIZE - 1);

   /*After sorting, printing the array elements*/
   cout << "\nAfter sorting array elements : ";
   for (i = 0; i < ARRAY_SIZE; i++)
   {
      cout << " \n" << array[i];
   }

   system("pause");
   return 0;
}

/*It does the merge-sort on the array
*p is the starting index of array
*r is the ending index of array*/

void MergeSort(int arr[], int p, int r)
{
   int q;
   if (p < r)
   {
      q = (r + p) / 2;
      MergeSort(arr, p, q);
```

```
      MergeSort(arr, q + 1, r);
      Merge(arr, p, q, r);
   }
}

/*Merge() does the merging of two sorted subarrays.
*arr[p..q] and arr[q+1..r] are the two sorted arrays.
*It does the marging without using sentinales*/
void Merge(int arr[], int p, int q, int r)
{
   int  *L, *R;
   int i = 0, j = 0, n1, n2, k = p;

   n1 = q - p + 1;
   n2 = r - q;
   L = new int[n1];
   R = new int[n2];

   for (i = 0; i<n1; i++)
      L[i] = arr[p + i];
   for (j = 0; j<n2; j++)
      R[j] = arr[q + j + 1];

   /*reset i and j*/
   i = 0;
   j = 0;
   /*merging the items in sorted order
   *till we find end of any array L or R*/
   while (i < n1 && j < n2)
   {
      if (L[i] < R[j])
      {
         arr[k] = L[i];
         i = i + 1;
      }
      else
      {
         arr[k] = R[j];
         j = j + 1;
      }
      k++;
   }

   /*check whether any array has some elements left
   *if some items left then put them to the final O/P array*/
```

```
    if (i != n1)
    for (; i < n1; i++)
    {
        arr[k] = L[i];
        k++;
    }
    else if (j != n1)
    for (; j < n2; j++)
    {
        arr[k] = R[j];
        k++;
    }

    delete[] L;
    delete[] R;
}
```