

```

/* ----- */
/* NAME : Bryan Wandrych           User ID: bdwandry */
/* DUE DATE : 11/3/2021           */
/* PROGRAM ASSIGNMENT 3           */
/* FILE NAME : thread-main.cpp     */
/* PROGRAM PURPOSE : To conduct an sorting algorithm based on */
/* even-odd paradigm concurrently. This files function is to */
/* deal with reading a file using CPP calls and functions and */
/* setup/iterate through even and odd sort. Throughout the */
/* iteration process, it will create the compute threads.    */
/* ----- */
#include <iostream>
#include "thread.h"
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
using namespace std;

//Global variables that are created to be used by all different files linked together.
int arrSize;
int * arrData;
char mainBuff[10000];

/* ----- */
/* FUNCTION: printArray           */
/* This function is used for no more than printing the array*/
/* data at different points in time. Its mains purpose is to*/
/* consolitdate code into a single function.                */
/* PARAMETER USAGE :           */
/* option = Prints out different statement depending on     */
/* the specified value passed through.                      */
/* value = A specific value used for the 3rd option of      */
/* of printing.                                             */
/* ----- */
/* FUNCTION CALLED :           */
/* Main                      */
/* ----- */
int printArray(int option, int val) {

    //Prints out beginning of the programs information
    if (option == 1) {
        write(1, "Concurrent Even-Odd Sort\n", 26);
        sprintf(mainBuff, "Number of input data = %d\n", arrSize);
        write(1, mainBuff, strlen(mainBuff));
        memset(mainBuff, 0, 10000);
        write(1, "Input Array:\n", 16);
    }

    //Prints out end of the programs information
    if (option == 2) {
        //write(1, "Final result after iteration j:\n", 35);
        sprintf(mainBuff, "Final result after iteration %d:\n", val);
        write(1, mainBuff, strlen(mainBuff));
        memset(mainBuff, 0, 10000);
    }

    //Prints out iteration i of the programs information
    if (option == 3) {
        sprintf(mainBuff, "Result after iteration %d:\n", val);
        write(1, mainBuff, strlen(mainBuff));
        memset(mainBuff, 0, 10000);
    }
}

```

```

//Tries to do a basic format print of the array data based on the option passed
through.
for (int i = 0; i < arrSize; i++) {
    if ((i % 20 == 0) && (i != 0)) {
        sprintf(mainBuff + strlen(mainBuff), "\n    ");
    }

    if ((option == 1) || (option == 2)) {
        if ((arrData[i] < 10) && (arrSize > 20)) {
            sprintf(mainBuff + strlen(mainBuff), "%d    ",
arrData[i]);
        } else {
            sprintf(mainBuff + strlen(mainBuff), "%d    ",
arrData[i]);
        }
    }

    if ((option == 3)) {
        if ((arrData[i] < 10) && (arrSize > 20)) {
            sprintf(mainBuff + strlen(mainBuff), "%d ", arrData[i]);
        } else {
            sprintf(mainBuff + strlen(mainBuff), "%d ", arrData[i]);
        }
    }
}

write(1, mainBuff, strlen(mainBuff));
memset(mainBuff, 0, 10000);
write(1, "\n", 1);
return 1;
}

/* ----- */
/* FUNCTION: startThreadmentorEven */
/* The porpose of this program is to spin up all the */
/* threads and iterate through all of the even indices of */
/* the array data. */
/* ----- */
/* PARAMETER USAGE :           */
/* Reads data from the global array data and array size */
/* FUNCTION CALLED :           */
/* Main                      */
/* ----- */
int startThreadmentorEven() {
    EvenOddThread *evenoddtthread[arrSize];
    //-----
    //Even iteration
    write(1, "    Even Pass:\n", 15);

    //creates n/2 number of threads
    for (int j = 1; j < arrSize; j += 2) {
        evenoddtthread[j] = new EvenOddThread(j);
        evenoddtthread[j]->Begin();
    }

    //waits for n/2 number of threads
    for (int j = 1; j < arrSize; j += 2) {
        evenoddtthread[j]->Join();
    }
    return 1;
}

/* ----- */

```

```

/* FUNCTION: startThreadmentorOdd */
/* The purpose of this program is to spin up all the */
/* threads and iterate through all of the odd indices of */
/* the array data. */
/* PARAMETER USAGE : */
/* Reads data from the global array data and array size */
/* FUNCTION CALLED : */
/* Main */
/* ----- */
int startThreadmentorOdd() {
    EvenOddThread *evenoddthread[arrSize];
    // -----
    // Odd iteration
    write(1, "    Odd Pass:\n", 14);
    //creates n/2 number of threads
    for (int j = 2; j < arrSize; j += 2) {
        evenoddthread[j] = new EvenOddThread(j);
        evenoddthread[j]->Begin();
    }

    //waits for n/2 number of threads
    for (int j = 2; j < arrSize; j += 2) {
        evenoddthread[j]->Join();
    }
    return 1;
}

/* ----- */
/* FUNCTION: main */
/* Checks to see if all array elements are in order using */
/* ascending order. */
/* PARAMETER USAGE : */
/* Global ArrData: Searching through the array */
/* Global ArrSize: Knowing how many elements there are */
/* FUNCTION CALLED : */
/* main */
/* ----- */
int CheckIfNoSwaps() {
    for (int i = 0; i < arrSize-1; i++) {
        if (arrData[i] > arrData[i+1]) {
            return 1;
        }
    }
    return 0;
}

/* ----- */
/* FUNCTION: main */
/* This function is the main function to this program */
/* It will start out by reading in the inputfile passed */
/* through STDIN and starts even/odd sort process iterator */
/* PARAMETER USAGE : */
/* Reads in STDIN usage of a data file with 2 lines */
/* of inputs */
/* FUNCTION CALLED : */
/* N/A */
/* ----- */
int main(int argc, char *argv[]) {
    //Gets the Array Size from Stdin
    cin >> arrSize;

    //Gets the array elements from
    //int arrData[arrSize];

    arrData = (int *)malloc(sizeof(int) * arrSize);
    for (int i = 0; i < arrSize; i++) {
        int tempVar;
        cin >> tempVar;
        arrData[i] = tempVar;
    }

    //Main Printing out First Information
    printArray(1, 0);

    //Starting threadmentor and the iteration
    int swapped = 1;
    int i = 0;
    while (swapped == 1) {
        //Checks to see if all array elements are in order using ascending order.
        swapped = CheckIfNoSwaps();
        if (swapped == 0) {
            break;
        }

        //Print out
        sprintf(mainBuff, "Iteration %d:\n", i);
        write(1, mainBuff, strlen(mainBuff));
        memset(mainBuff, 0, 10000);

        //Starts the function to create/sort even elements
        startThreadmentorEven();

        //Starts the function to create/sort odd elements
        startThreadmentorOdd();

        //Iterates I to keep track of how many swaps it takes to sort.
        i++;

        //Prints out the array information
        printArray(3, i);
    }

    printArray(2, i);
    return 1;
}

```

```
/* ----- */
/* NAME : Bryan Wandrych           User ID: bdwandry */
/* DUE DATE : 11/3/2021           */
/* PROGRAM ASSIGNMENT 3           */
/* FILE NAME : thread-main.cpp     */
/* PROGRAM PURPOSE : To conduct an sorting algorithm based on */
/* even-odd paradigm concurrently. This files function is to */
/* create a link and shared memory between the threads. As well*/
/* as creating a link to threadmentor in the backend.          */
/* ----- */
#include "ThreadClass.h"

//Global variables that share values between Main and thread.cpp class
extern int arrSize;
extern int * arrData;

/* ----- */
/* FUNCTION: EvenOddThread (Class Definition)           */
/* This is the header file that describes the threadmentor's */
/* class. This class will be shared with all files when */
/* threads are called.                                     */
/* ----- */
/* PARAMETER USAGE :                                     */
/* int in - used to pass through k                       */
/* FUNCTION CALLED :                                     */
/* thread.cpp                                             */
/* Threadmentor                                          */
/* ----- */
class EvenOddThread : public Thread {
public:
    EvenOddThread(int in);

private:
    int index;
    void ThreadFunc();
};
```

```
/* ----- */
/* NAME : Bryan Wandrych           User ID: bdwandry */
/* DUE DATE : 11/3/2021           */
/* PROGRAM ASSIGNMENT 3           */
/* FILE NAME : thread-main.cpp    */
/* PROGRAM PURPOSE : To conduct an sorting algorithm based on */
/* even-odd paradigm concurrently. This files function, in */
/* separate threads, to do the array swapping if the conditions*/
/* are met if x[k-1] is less than x[k]. This will run */
/* threadmentor in the backend for thread-management. */
/* ----- */
#include <iostream>
#include "thread.h"
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

/* ----- */
/* FUNCTION: EvenOddThread */
/* This is the constructor to the EvenOddThread class */
/* described in the thread.h. It will supply definitions */
/* for pass through arguments passed from main. */
/* PARAMETER USAGE : */
/* int in = Passed from Main of the index for each */
/* specificied thread. */
/* FUNCTION CALLED: */
/* ThreadMentor */
/* ----- */
EvenOddThread::EvenOddThread(int in) {
    index = in;
}

/* ----- */
/* FUNCTION: ThreadFunc */
/* This is the executing code for the specific thread */
/* created. It will check the indices of the input array */
/* to see if x[x-1] > x[x], then swap, or else do nothing */
/* PARAMETER USAGE : */
/* ThreadMentor - used for managing threads */
/* index - defined from the constructor */
/* FUNCTION CALLED : */
/* ThreadMentor */
/* ----- */
void EvenOddThread::ThreadFunc() {
    Thread::ThreadFunc();
    Delay();
    //Print outs
    char threadBuf[10000];
    sprintf(threadBuf, "          Thread %d Created\n", index);
    write(1, threadBuf, strlen(threadBuf));
    memset(threadBuf, 0, 10000);
    Delay();

    sprintf(threadBuf, "          Thread %d compares %d and %d\n", index,
arrData[index - 1], arrData[index]);
    write(1, threadBuf, strlen(threadBuf));
    memset(threadBuf, 0, 10000);
    Delay();

    //Does the swapping
    if (arrData[index - 1] > arrData[index]) {
        Delay();
        sprintf(threadBuf, "          Thread %d swaps %d and %d\n", index,
```

```
arrData[index - 1], arrData[index]);
        write(1, threadBuf, strlen(threadBuf));
        memset(threadBuf, 0, 10000);
        Delay();

        int temp = arrData[index];
        arrData[index] = arrData[index - 1];
        arrData[index - 1] = temp;
        Delay();
    }

    //Print outs
    sprintf(threadBuf, "          Thread %d exits\n", index);
    write(1, threadBuf, strlen(threadBuf));
    memset(threadBuf, 0, 10000);
    Delay();
    Exit();
}
```

```
CC      = c++
FLAGS   =
CFLAGS  = -g -O2 -Wno-write-strings -Wno-cpp -w
DFLAGS  = -DPACKAGE=\"threadsystem\" -DVERSION=\"1.0\" -DPTHREAD=1 -DUNIX_MSG_Q=1
-DSTDC_HEADERS=1
IFLAGS  = -I/local/eit-linux/apps/ThreadMentor/include
TMLIB   = /local/eit-linux/apps/ThreadMentor/Visual/libthreadclass.a
TMLIB_NV = /local/eit-linux/apps/ThreadMentor/NoVisual/libthreadclass.a
OBJ_FILE = thread.o thread-main.o
EXE_FILE = prog3
${EXE_FILE}: ${OBJ_FILE}
            ${CC} ${FLAGS} -o ${EXE_FILE} ${OBJ_FILE} ${TMLIB} -lpthread

thread.o: thread.cpp
            ${CC} ${DFLAGS} ${IFLAGS} ${CFLAGS} -c thread.cpp

thread-main.o: thread-main.cpp
            ${CC} ${DFLAGS} ${IFLAGS} ${CFLAGS} -c thread-main.cpp

noVisual: ${OBJ_FILE}
            ${CC} ${FLAGS} -o ${EXE_FILE} ${OBJ_FILE} ${TMLIB_NV} -lpthread

clean:
            rm -f ${OBJ_FILE} ${EXE_FILE}
```

```

===== COMPILATION =====
rm -f thread.o thread-main.o prog3
c++ -DPACKAGE=\"threadsystem\" -DVERSION=\"1.0\" -DPTHREAD=1 -DUNIX_MSG_Q=1
-DSTDC_HEADERS=1 -I/local/eit-linux/apps/ThreadMentor/include -g -O2 -Wno-write-strings
-Wno-cpp -w -c thread.cpp
c++ -DPACKAGE=\"threadsystem\" -DVERSION=\"1.0\" -DPTHREAD=1 -DUNIX_MSG_Q=1
-DSTDC_HEADERS=1 -I/local/eit-linux/apps/ThreadMentor/include -g -O2 -Wno-write-strings
-Wno-cpp -w -c thread-main.cpp
c++ -o prog3 thread.o thread-main.o
/local/eit-linux/apps/ThreadMentor/NoVisual/libthreadclass.a -lpthread
Compilation done.
===== TEST 1 =====
Concurrent Even-Odd Sort

Number of input data = 8
Input Array:
 7  1  3  2  8  4  5  9
Iteration 0:
  Even Pass:
    Thread 1 Created
    Thread 1 compares 7 and 1
    Thread 1 swaps 7 and 1
    Thread 3 Created
    Thread 1 exits
    Thread 3 compares 3 and 2
    Thread 3 swaps 3 and 2
    Thread 5 Created
    Thread 3 exits
    Thread 5 compares 8 and 4
    Thread 5 swaps 8 and 4
    Thread 7 Created
    Thread 5 exits
    Thread 7 compares 5 and 9
    Thread 7 exits
  Odd Pass:
    Thread 2 Created
    Thread 2 compares 7 and 2
    Thread 2 swaps 7 and 2
    Thread 4 Created
    Thread 2 exits
    Thread 4 compares 3 and 4
    Thread 4 exits
    Thread 6 Created
    Thread 6 compares 8 and 5
    Thread 6 swaps 8 and 5
    Thread 6 exits
Result after iteration 1:
 1  2  7  3  4  5  8  9
Iteration 1:
  Even Pass:
    Thread 1 Created
    Thread 3 Created
    Thread 1 compares 1 and 2
    Thread 1 exits
    Thread 3 compares 7 and 3
    Thread 3 swaps 7 and 3
    Thread 3 exits
    Thread 5 Created
    Thread 7 Created
    Thread 5 compares 4 and 5
    Thread 7 compares 8 and 9
    Thread 7 exits

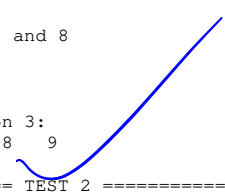
```

```

    Thread 5 exits
  Odd Pass:
    Thread 4 Created
    Thread 6 Created
    Thread 2 Created
    Thread 4 compares 7 and 4
    Thread 6 compares 5 and 8
    Thread 6 exits
    Thread 2 compares 2 and 3
    Thread 4 swaps 7 and 4
    Thread 2 exits
    Thread 4 exits
Result after iteration 2:
 1  2  3  4  7  5  8  9
Iteration 2:
  Even Pass:
    Thread 1 Created
    Thread 1 compares 1 and 2
    Thread 3 Created
    Thread 1 exits
    Thread 3 compares 3 and 4
    Thread 5 Created
    Thread 3 exits
    Thread 5 compares 7 and 5
    Thread 5 swaps 7 and 5
    Thread 5 exits
    Thread 7 Created
    Thread 7 compares 8 and 9
    Thread 7 exits
  Odd Pass:
    Thread 4 Created
    Thread 4 compares 4 and 5
    Thread 2 Created
    Thread 4 exits
    Thread 2 compares 2 and 3
    Thread 6 Created
    Thread 2 exits
    Thread 6 compares 7 and 8
    Thread 6 exits
Result after iteration 3:
 1  2  3  4  5  7  8  9
Final result after iteration 3:
 1  2  3  4  5  7  8  9
===== TEST 2 =====
Concurrent Even-Odd Sort

Number of input data = 16
Input Array:
22 -5 12 14 78 -3 -53 13 41 51 5 8 2 6 -7 2
Iteration 0:
  Even Pass:
    Thread 1 Created
    Thread 1 compares 22 and -5
    Thread 1 swaps 22 and -5
    Thread 1 exits
    Thread 3 Created
    Thread 3 compares 12 and 14
    Thread 3 exits
    Thread 5 Created
    Thread 5 compares 78 and -3
    Thread 5 swaps 78 and -3
    Thread 5 exits
    Thread 7 Created

```



```
Thread 7 compares -53 and 13
Thread 7 exits
Thread 9 Created
Thread 9 compares 41 and 51
Thread 9 exits
Thread 11 Created
Thread 11 compares 5 and 8
Thread 11 exits
Thread 13 Created
Thread 13 compares 2 and 6
Thread 15 Created
Thread 13 exits
Thread 15 compares -7 and 2
Thread 15 exits
Odd Pass:
Thread 2 Created
Thread 4 Created
Thread 4 compares 14 and -3
Thread 2 compares 22 and 12
Thread 4 swaps 14 and -3
Thread 6 Created
Thread 2 swaps 22 and 12
Thread 6 compares 78 and -53
Thread 4 exits
Thread 2 exits
Thread 6 swaps 78 and -53
Thread 8 Created
Thread 6 exits
Thread 8 compares 13 and 41
Thread 8 exits
Thread 10 Created
Thread 10 compares 51 and 5
Thread 10 swaps 51 and 5
Thread 10 exits
Thread 12 Created
Thread 14 Created
Thread 14 compares 6 and -7
Thread 12 compares 8 and 2
Thread 14 swaps 6 and -7
Thread 12 swaps 8 and 2
Thread 14 exits
Thread 12 exits
Result after iteration 1:
-5 12 22 -3 14 -53 78 13 41 5 51 2 8 -7 6 2
Iteration 1:
Even Pass:
Thread 1 Created
Thread 3 Created
Thread 1 compares -5 and 12
Thread 3 compares 22 and -3
Thread 1 exits
Thread 5 Created
Thread 5 compares 14 and -53
Thread 3 swaps 22 and -3
Thread 7 Created
Thread 3 exits
Thread 5 swaps 14 and -53
Thread 5 exits
Thread 7 compares 78 and 13
Thread 7 swaps 78 and 13
Thread 7 exits
Thread 11 Created
Thread 11 compares 51 and 2
Thread 11 swaps 51 and 2
```

```
Thread 11 exits
Thread 9 Created
Thread 9 compares 41 and 5
Thread 9 swaps 41 and 5
Thread 9 exits
Thread 13 Created
Thread 13 compares 8 and -7
Thread 13 swaps 8 and -7
Thread 13 exits
Thread 15 Created
Thread 15 compares 6 and 2
Thread 15 swaps 6 and 2
Thread 15 exits
Odd Pass:
Thread 2 Created
Thread 2 compares 12 and -3
Thread 4 Created
Thread 4 compares 22 and -53
Thread 2 swaps 12 and -3
Thread 2 exits
Thread 6 Created
Thread 4 swaps 22 and -53
Thread 6 compares 14 and 13
Thread 8 Created
Thread 4 exits
Thread 8 compares 78 and 5
Thread 6 swaps 14 and 13
Thread 8 swaps 78 and 5
Thread 6 exits
Thread 10 Created
Thread 8 exits
Thread 10 compares 41 and 2
Thread 10 swaps 41 and 2
Thread 10 exits
Thread 12 Created
Thread 14 Created
Thread 14 compares 8 and 2
Thread 12 compares 51 and -7
Thread 12 swaps 51 and -7
Thread 14 swaps 8 and 2
Thread 12 exits
Thread 14 exits
Result after iteration 2:
-5 -3 12 -53 22 13 14 5 78 2 41 -7 51 2 8 6
Iteration 2:
Even Pass:
Thread 1 Created
Thread 1 compares -5 and -3
Thread 1 exits
Thread 3 Created
Thread 3 compares 12 and -53
Thread 5 Created
Thread 3 swaps 12 and -53
Thread 5 compares 22 and 13
Thread 7 Created
Thread 3 exits
Thread 5 swaps 22 and 13
Thread 7 compares 14 and 5
Thread 5 exits
Thread 7 swaps 14 and 5
Thread 7 exits
Thread 9 Created
Thread 9 compares 78 and 2
Thread 9 swaps 78 and 2
```

```
Thread 9 exits
Thread 11 Created
Thread 11 compares 41 and -7
Thread 11 swaps 41 and -7
Thread 11 exits
Thread 13 Created
Thread 13 compares 51 and 2
Thread 15 Created
Thread 13 swaps 51 and 2
Thread 13 exits
Thread 15 compares 8 and 6
Thread 15 swaps 8 and 6
Thread 15 exits
Odd Pass:
Thread 2 Created
Thread 2 compares -3 and -53
Thread 2 swaps -3 and -53
Thread 4 Created
Thread 2 exits
Thread 6 Created
Thread 4 compares 12 and 13
Thread 6 compares 22 and 5
Thread 4 exits
Thread 6 swaps 22 and 5
Thread 8 Created
Thread 8 compares 14 and 2
Thread 6 exits
Thread 8 swaps 14 and 2
Thread 8 exits
Thread 10 Created
Thread 10 compares 78 and -7
Thread 10 swaps 78 and -7
Thread 10 exits
Thread 14 Created
Thread 12 Created
Thread 12 compares 41 and 2
Thread 14 compares 51 and 6
Thread 14 swaps 51 and 6
Thread 12 swaps 41 and 2
Thread 14 exits
Thread 12 exits
Result after iteration 3:
-5 -53 -3 12 13 5 22 2 14 -7 78 2 41 6 51 8
Iteration 3:
Even Pass:
Thread 1 Created
Thread 3 Created
Thread 1 compares -5 and -53
Thread 3 compares -3 and 12
Thread 5 Created
Thread 3 exits
Thread 1 swaps -5 and -53
Thread 5 compares 13 and 5
Thread 1 exits
Thread 7 Created
Thread 5 swaps 13 and 5
Thread 7 compares 22 and 2
Thread 5 exits
Thread 7 swaps 22 and 2
Thread 7 exits
Thread 11 Created
Thread 11 compares 78 and 2
Thread 9 Created
Thread 11 swaps 78 and 2

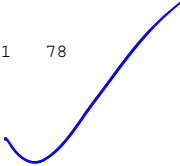
Thread 9 compares 14 and -7
Thread 9 swaps 14 and -7
Thread 11 exits
Thread 9 exits
Thread 13 Created
Thread 13 compares 41 and 6
Thread 13 swaps 41 and 6
Thread 15 Created
Thread 15 compares 51 and 8
Thread 13 exits
Thread 15 swaps 51 and 8
Thread 15 exits
Odd Pass:
Thread 2 Created
Thread 4 Created
Thread 2 compares -5 and -3
Thread 2 exits
Thread 4 compares 12 and 5
Thread 6 Created
Thread 4 swaps 12 and 5
Thread 6 compares 13 and 2
Thread 4 exits
Thread 8 Created
Thread 6 swaps 13 and 2
Thread 8 compares 22 and -7
Thread 8 swaps 22 and -7
Thread 6 exits
Thread 8 exits
Thread 10 Created
Thread 10 compares 14 and 2
Thread 10 swaps 14 and 2
Thread 10 exits
Thread 12 Created
Thread 14 Created
Thread 14 compares 41 and 8
Thread 12 compares 78 and 6
Thread 14 swaps 41 and 8
Thread 12 swaps 78 and 6
Thread 14 exits
Thread 12 exits
Result after iteration 4:
-53 -5 -3 5 12 2 13 -7 22 2 14 6 78 8 41 51
Iteration 4:
Even Pass:
Thread 1 Created
Thread 1 compares -53 and -5
Thread 3 Created
Thread 1 exits
Thread 3 compares -3 and 5
Thread 3 exits
Thread 5 Created
Thread 5 compares 12 and 2
Thread 5 swaps 12 and 2
Thread 7 Created
Thread 5 exits
Thread 7 compares 13 and -7
Thread 7 swaps 13 and -7
Thread 7 exits
Thread 9 Created
Thread 9 compares 22 and 2
Thread 9 swaps 22 and 2
Thread 9 exits
Thread 11 Created
Thread 11 compares 14 and 6
```



```
Thread 11 swaps 14 and 6
Thread 11 exits
Thread 13 Created
Thread 15 Created
Thread 13 compares 78 and 8
Thread 13 swaps 78 and 8
Thread 15 compares 41 and 51
Thread 13 exits
Thread 15 exits
Odd Pass:
Thread 2 Created
Thread 2 compares -5 and -3
Thread 2 exits
Thread 4 Created
Thread 4 compares 5 and 2
Thread 6 Created
Thread 4 swaps 5 and 2
Thread 6 compares 12 and -7
Thread 4 exits
Thread 6 swaps 12 and -7
Thread 10 Created
Thread 6 exits
Thread 8 Created
Thread 10 compares 22 and 6
Thread 8 compares 13 and 2
Thread 10 swaps 22 and 6
Thread 10 exits
Thread 8 swaps 13 and 2
Thread 8 exits
Thread 12 Created
Thread 14 Created
Thread 12 compares 14 and 8
Thread 14 compares 78 and 41
Thread 12 swaps 14 and 8
Thread 14 swaps 78 and 41
Thread 12 exits
Thread 14 exits
Result after iteration 5:
-53 -5 -3 2 5 -7 12 2 13 6 22 8 14 41 78 51
Iteration 5:
Even Pass:
Thread 1 Created
Thread 3 Created
Thread 1 compares -53 and -5
Thread 1 exits
Thread 3 compares -3 and 2
Thread 7 Created
Thread 3 exits
Thread 5 Created
Thread 7 compares 12 and 2
Thread 5 compares 5 and -7
Thread 7 swaps 12 and 2
Thread 7 exits
Thread 5 swaps 5 and -7
Thread 5 exits
Thread 11 Created
Thread 11 compares 22 and 8
Thread 11 swaps 22 and 8
Thread 11 exits
Thread 9 Created
Thread 9 compares 13 and 6
Thread 9 swaps 13 and 6
Thread 9 exits
Thread 13 Created
```

```
Thread 13 compares 14 and 41
Thread 13 exits
Thread 15 Created
Thread 15 compares 78 and 51
Thread 15 swaps 78 and 51
Thread 15 exits
Odd Pass:
Thread 2 Created
Thread 2 compares -5 and -3
Thread 2 exits
Thread 4 Created
Thread 6 Created
Thread 4 compares 2 and -7
Thread 4 swaps 2 and -7
Thread 6 compares 5 and 2
Thread 6 swaps 5 and 2
Thread 6 exits
Thread 4 exits
Thread 10 Created
Thread 8 Created
Thread 10 compares 13 and 8
Thread 10 swaps 13 and 8
Thread 8 compares 12 and 6
Thread 10 exits
Thread 8 swaps 12 and 6
Thread 8 exits
Thread 12 Created
Thread 14 Created
Thread 12 compares 22 and 14
Thread 14 compares 41 and 51
Thread 12 swaps 22 and 14
Thread 14 exits
Thread 12 exits
Result after iteration 6:
-53 -5 -3 -7 2 2 5 6 12 8 13 14 22 41 51 78
Iteration 6:
Even Pass:
Thread 1 Created
Thread 1 compares -53 and -5
Thread 3 Created
Thread 5 Created
Thread 1 exits
Thread 5 compares 2 and 2
Thread 7 Created
Thread 3 compares -3 and -7
Thread 5 exits
Thread 7 compares 5 and 6
Thread 3 swaps -3 and -7
Thread 7 exits
Thread 3 exits
Thread 9 Created
Thread 9 compares 12 and 8
Thread 9 swaps 12 and 8
Thread 9 exits
Thread 11 Created
Thread 11 compares 13 and 14
Thread 11 exits
Thread 13 Created
Thread 13 compares 22 and 41
Thread 13 exits
Thread 15 Created
Thread 15 compares 51 and 78
Thread 15 exits
Odd Pass:
```

```
Thread 2 Created
Thread 2 compares -5 and -7
Thread 2 swaps -5 and -7
Thread 2 exits
Thread 4 Created
Thread 6 Created
Thread 4 compares -3 and 2
Thread 8 Created
Thread 8 compares 6 and 8
Thread 4 exits
Thread 6 compares 2 and 5
Thread 6 exits
Thread 8 exits
Thread 10 Created
Thread 10 compares 12 and 13
Thread 10 exits
Thread 12 Created
Thread 14 Created
Thread 12 compares 14 and 22
Thread 14 compares 41 and 51
Thread 12 exits
Thread 14 exits
Result after iteration 7:
-53 -7 -5 -3 2 2 5 6 8 12 13 14 22 41 51 78
Final result after iteration 7:
-53 -7 -5 -3 2 2 5 6 8 12 13 14 22 41 51 78
```



Bryan Wandrych
bdwandry
11/3/21

1. Question: Are there any race conditions in this even-odd sort as suggested? Why?
No there are no race conditions to contend too or take care of in this project. The reasoning why is that we are dealing with Even elements ($2k$) and odd elements ($2k+1$) separately. And when it iterates through each of the elements, its only touch ($i-1$, i) elements for only that thread. So which means there no communications that are

needed to stop one thread from overlapping data from another thread.

2. Question: Prove rigorously that this algorithm does sort the input number correctly and takes no more than n iterations to sort an array of n numbers.

This algorithm at most will do n number of swaps.

//Assumption that there are no base cases that will check for a single a element in the array - my program will.

//Meaning 1 iteration is needed for i .

if $n = 1$ (meaning there is only one array element),

n will run and only even will try to swap.

Since, there other elements to contend too, there will only be one iteration of i needed, aka only at most n number of swaps.

But, for the most part, this should only take in between upperbound($n/2$) or upperbound($i/2$) iterations.

if $n = 2$,

element 0 and 1 will swap, this would only take one swap of even algorithm, then the data is sorted at 1 iteration of $n = 2$.

The main reasoning is that were allowing for more than one swapping of elements and breaking down even/odd element numbers.

3. Question: In each iteration, the main() does the creation and join for the completion of $n/2$ threads twice,

once for an even pass and the other for an odd pass.

Compared with simple comparisons, it requires a significant amount of time in creating and joining threads.

If you are allowed to use extra variables/arrays and busy waiting, can you just create $n/2$ threads and let them do both

the even pass and the odd pass in the same iteration without race conditions and still deliver correct results?

More precisely, thread T_k compares $x[k-1]$ and $x[k]$ in an even pass, and then compare $x[k]$ and $x[k+1]$ in an odd pass?

Suggest a solution and discuss its correctness.

Yes, I believe you could do this in a more efficient manner with extra variables. Here is my proposed algorithm below:

You could make this algorithm more efficient by having three separate arrays, one array will contain the

elements of even indices of the main array. The 2nd will contain the elements of odd indices of the main array.

The third would be a merged array of in order elements.

Next you could then spawn up $n/2$ and check the following:

Even array:

You'd check the index of second (index number = 1) element contained in its array (even).

You'd check the first index (index number = 0) of the odd array.

Compare their values.

Then in a third array that was created, you'd place your output into that array.

Then

Odd Array:

You'd check the index of Third (index number = 2)

element contained in its array (odd).

You'd check the Second index (index number = 1) of the

even array.

Compare their values.

Then in a third array that was created, you'd place your

output into that array.

Continue to loop until all elements from the original array are

stored into the third.

This approach would save time on creating/joining threads, because it would allow for one thread to do more than one computation at once.

However, it may be faster, but its less space efficient.

4. Question: Furthermore, can you just create $n/2$ threads at the very beginning and let them do all the even pass and odd pass comparisons?

In this way, you save more time on creating and joining threads.

Suggest a solution and discuss its correctness.

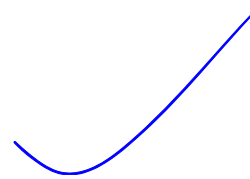
Since we already know that there is at most the upperbound($n/2$) threads created in this process, we could take each some of our iteration code

out of main and reduce the number of threads created by not needing to create a separate thread for each swap. Each thread could swap both even and odd elements.

This would turn the algorithm into a bubble sort, but it would still allow for more than one swap per thread, thus reducing time it takes to spinup or join (wait)

for other threads to continue. But, by doing this, you will have to allow for busy waiting and have mutex instantiated because more than one thread can interact

with more than one element.



CS3331 Program 3 Grade Report

You receive 0 point if any one of the following occurs
No further grading will be done

<i>Problem</i>	<i>Check All Apply</i>	<i>You Receive</i>
Not-compile		0
Compile-but-not-run		0
Meaningless and/or vague Program		0
Did not implement the indicated methods		0
Did not follow the required program structure		0
Other significant deviation from specification, e.g., maximum parallelism		0
Totally wrong and unacceptable output		0

This part applies only if you have a working program

<i>Item</i>		<i>Max Possible</i>	<i>You Receive</i>
Style & Doc.	Header in each file	1	1
	Good indentation	1	1
	Good comments	1	1
	Good use of function, variable names, etc & no GOTO	2	2
Spec	Handles input and argument list properly	1	1
	Correct output format	3	3
Correctness	Work on sample data	18	18
	Work on our data	18	18
README	Missing README – next two items receive 0	0	0
	Well-written README	2	2
	Answer questions properly	3	3
Total		50	50

Your Score: 50