**Part-1**

For this assignment your program take input from standard input, sorts the data and dumps the sorted data out to standard output.  The input will have the following format.  Each record is 64 bytes long and the first 8 characters are the key.  You should sort the data by the key.

#define DATALEN 56

#define KEYLEN 8

typedef struct \_RECORD {

       TCHAR key[KEYLEN];

       TCHAR data[DATALEN];

} RECORD;

#define RECSIZE sizeof (RECORD)

**unsorted.txt**

b10371d9. Record Number: 00000000.abcdefghijklmnopqrstuvwxyz x
91fdc79e. Record Number: 00000001.abcdefghijklmnopqrstuvwxyz x
fe14527f. Record Number: 00000002.abcdefghijklmnopqrstuvwxyz x
6d04024c. Record Number: 00000003.abcdefghijklmnopqrstuvwxyz x
b6305295. Record Number: 00000004.abcdefghijklmnopqrstuvwxyz x
fffabfaa. Record Number: 00000005.abcdefghijklmnopqrstuvwxyz x
cba5979b. Record Number: 00000006.abcdefghijklmnopqrstuvwxyz x
c694f738. Record Number: 00000007.abcdefghijklmnopqrstuvwxyz x
5e108311. Record Number: 00000008.abcdefghijklmnopqrstuvwxyz x
c51d2c76. Record Number: 00000009.abcdefghijklmnopqrstuvwxyz x

Below is an example run of the sort program from the command prompt.  The 1 in the argument list tells the program to start up (1) process.  The number of processes should always be a power of 2.  Example 1, 2, 4, 8, 16, 32 …

The file unsorted.txt becomes standard input.
the file sorted.txt becomes standard output.

Sortmp 1 <unsorted.txt >sorted.txt

Once the above command has ran and completed the sort, the following file should be created and full of sorted data.  Compare this file to the unsorted.txt file.

**Sorted.txt**

5e108311. Record Number: 00000008.abcdefghijklmnopqrstuvwxyz x
6d04024c. Record Number: 00000003.abcdefghijklmnopqrstuvwxyz x
91fdc79e. Record Number: 00000001.abcdefghijklmnopqrstuvwxyz x
b10371d9. Record Number: 00000000.abcdefghijklmnopqrstuvwxyz x
b6305295. Record Number: 00000004.abcdefghijklmnopqrstuvwxyz x
c51d2c76. Record Number: 00000009.abcdefghijklmnopqrstuvwxyz x
c694f738. Record Number: 00000007.abcdefghijklmnopqrstuvwxyz x
cba5979b. Record Number: 00000006.abcdefghijklmnopqrstuvwxyz x
fe14527f. Record Number: 00000002.abcdefghijklmnopqrstuvwxyz x
fffabfaa. Record Number: 00000005.abcdefghijklmnopqrstuvwxyz x

Other example runs and there explanations.

Sortmp 2 <unsorted.txt >sorted.txt

For this example the parent processes starts up two child processes.  The first half of the data will be given to child process (1) as input and the second half of data will be given to child process (2) as input.  Both child processes will sort their respective data and output a sorted list.  There sorted list will go to standard output which the parent processes would have forced to a file that the parent has access to.  The parent will create two temp files that will be used by the child processes for output.  The parent should wait for both processes to complete their sorting.  Once each child is done the parent can merge the two sorted list which should only take O(N) time.  The merged list should then get dumped to standard output.

Sortmp 4 <unsorted.txt >sorted.txt  This has the same processes as above but in this example each child will have two children.  So each of their children will sort half of the child list.  (¼ of the entire list)  The only sorting will take place at the bottom level of the tree.  At every other level only merging of sorted lists will take place.

**Part-2**

**Sortmp <unsorted.txt >sorted.txt**

**For this run there are no command line options telling you how many child processes to run at the bottom level of the tree.  Use the instruction to get the physical cores of your computer.  If the physical cores are 12 then use the next power of 2 for the number of children at the bottom level.  So if the computer has 12 physical cores there should be 16 processes at the bottom level of the tree.**