ECE/ME/EMA/CS 759
High Performance Computing for Engineering Applications
Assignment 2

Date Assigned: September 16, 2013
Date Due: September 23, 2013 (11:59 PM)

1. There are two related functions that copy/move memory around in C: memcpy and memmove (assume neither of these functions is an alias of the other). When should you use one or the other? Are there any performance issues that should dictate your choice?

Answer:
With memcpy, the destination cannot overlap the source at all. With memmove it can. This means that memmove might be very slightly slower than memcpy, as it cannot make the same assumptions.

For example, memcpy might always copy addresses from low to high. If the destination overlaps after the source, this means some addresses will be overwritten before copied. memmove would detect this and copy in the other direction - from high to low - in this case. However, checking this and switching to another (possibly less efficient) algorithm takes time.

So, memcpy is safer to use when it comes to performance issues.

2. Write a program that reads in a set of words/numbers provided as command line arguments. Within a function that you come up with, count the total number of characters in the input excluding white spaces. Return this value and print it out in the main function. Note: the set of words/numbers can contain combinations of letters and digits only. Don't use other special characters. Here's a possible scenario:
>> myProgram this is a test 4 you 2
The output of the program should be 16, since there are 16 characters in the input provided above (excluding white spaces). For this problem you might want to use the variables argv and argc that you have access to in the main function of your program.

Answer:
Please find the code saved as p2.c in the zip folder and also in 'hw2' directory in my Euler account.
After compiling, type ./p2 this is a test 4 you 2 (for example) and it will give the output as "Length of string: 16"

Code:

```
#include<stdio.h>
#include<string.h>

void main(int argc, char* argv[])
{
int len = 0;
char ctr;

if  (argc<2)
printf("\n Invalid string!");

else
```

```
for(ctr = 1; ((ctr<argc) && (ctr!=' '));ctr++)
{
len = len + strlen(argv[ctr]);
}
printf("\n Length of string is %d", len);
}
```

3. Write a C or C++ program that takes as an argument an input file, it opens the file, reads all the integer numbers (no more than 1,000,000 of them) and sorts them from the smallest to the largest. The input file contains one integer per line.

a. Without consulting any online source or book, use an algorithm that youdesign or are familiar with to generate a program that prints out four things: (*i*) the number of integers read in, (*ii*) the smallest integer read in, (*iii*) the largest one, and (*iv*) the amount of time in milliseconds (ms) required to sort these integers.

Answer: Please find the code named 'p3.c' (in the .zip folder). The binary file will be generated as **"prateekfile.bin"** after executing 'p3.c'. This file will be having random number of integers. For sorting these integers and checking the sorting time, compile the code named 'p4.c' (in the .zip folder) and then execute it as: './p4 prateekfile.bin'. It will print the number of integers read in, the smallest integer, the largest integer and the amount of time in milliseconds (ms) our sorting method took.

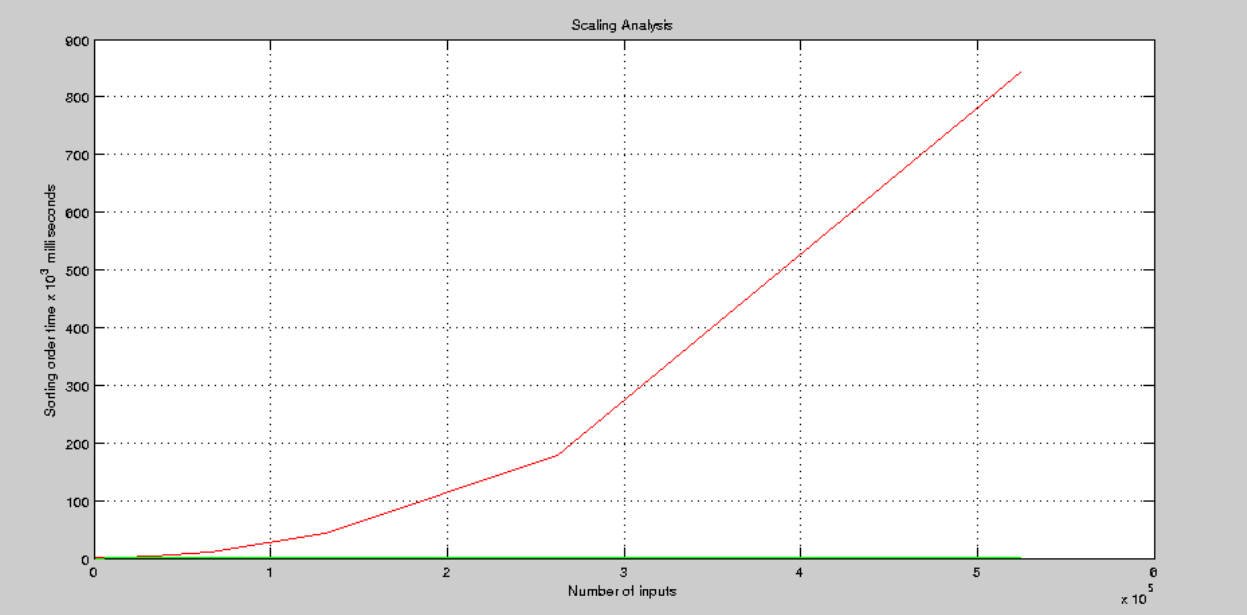
Note:
For generating nine set of files from 2^10 to 2^19, the value in 'line 22: x = pow(2,12);' keep on changing the value of second figure in the parathesis (marked red here). For example for generating the file for 2^10 it should be 'x= pow(2,10); and then compile p4.c and execute the same for getting the sort values and time, for 2^11, it should be 'x=pow(2,11); so on and so forth till it reaches 'x=pow(2,19);'

b. In order to gauge the performance of your sort algorithm consult a book, online source, or use a library that has a sort function implemented to sort the same collection of integers as at a) above to verify your answer.

Answer: I have implemented 'Qsort'algorithm for gauging the performance of mysort. After executing the code './p4 prateekfile.bin', the output of the code will print the time taken by Qsort in milliseconds and also the amount of time my sorting took (in ms).

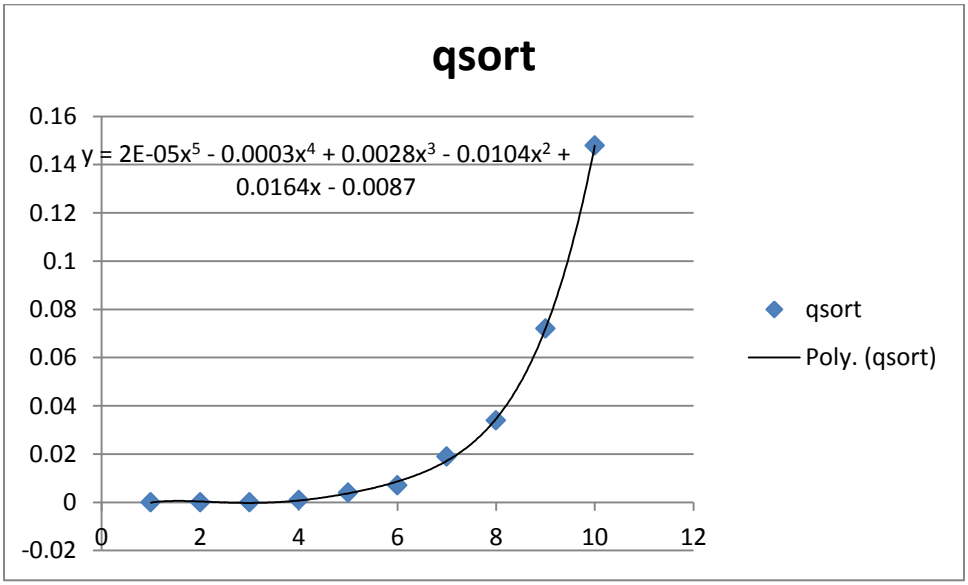
c. To understand the behavior of the two approaches at a) and b) above, run them on a set of nine input files containing 2^10, 2^11,…, 2^19 integers generated randomly. Record the time in ms (milliseconds) and run a standard regression to approximate the curves that give ordering time versus number of integers for the two algorithms. This is called a scaling analysis. Provide the plot in **png** format and indicate the regression function. Feel free to post this plot in case you want to help other colleagues get an idea of what they should obtain at the end of this exercise.

Answer: As metioned above on changing the values of line 22 in program code 'p3.c' from 12 to 19 and then executing the program code 'p4.c' will give the performance in terms of the amount of time taken in milliseconds by Qsort and my sort.
The plot (**png** format) for 'ordering time (in ms) versus number of integers is as follows'(also find the file 'Scaling Analysis_Plot' in .zip folder and also the curve in MS Excel showing the regression function for both the algorithms):

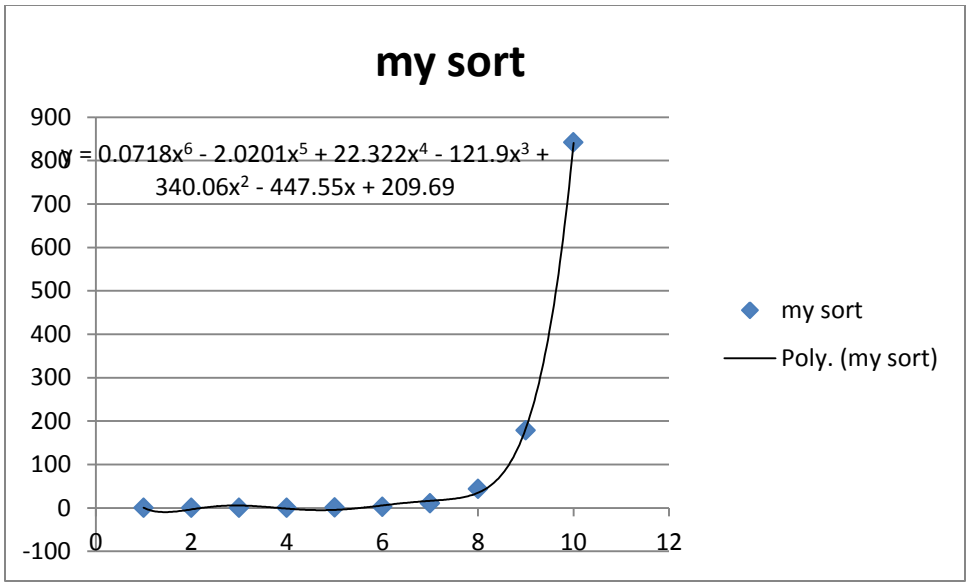


SCALING ANALYSIS PLOT

In Red: the curve for ordering time by my sort.
In Green: the curve for ordering time by Qsort.



Regression function of Qsort:
Y= 2*10^(-5) – 0.0003x^4 + 0.0028x^3 – 0.0104x^2 + 0.0164x – 0.0087



Regression function of my sort:
Y = 0.0718x^6 – 2.0201x^5 + 22.322x^4 – 121.9x^3 + 340.06x^2 – 447.55x + 209.69

4. This problem is identical to Problem 3, except that instead of the sorting operation you have to perform an exclusive scan operation. For a definition of the scan operation read pages 1 through 3 of the Blelloch 1990 paper available on the class website. Your program should be able to take one input argument which is the name of the text file storing the integers. The input file contains one integer per line.

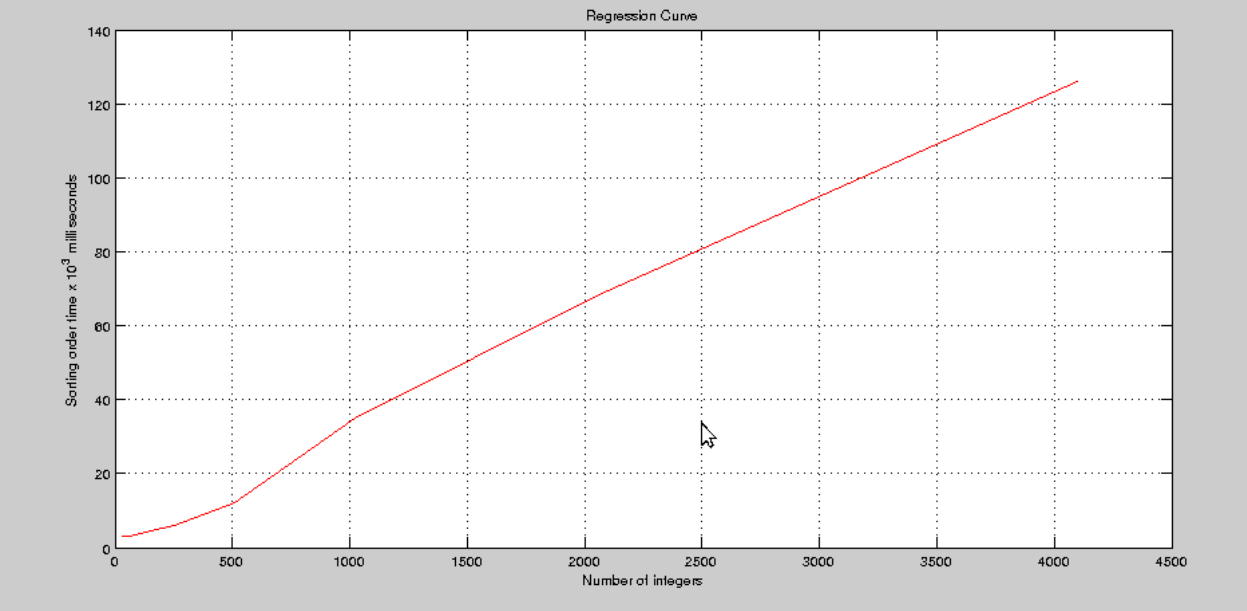
a. Implement an algorithm that executes an *exclusive scan* operation and prints out three things: (*i*) the number of integers read in, (*ii*) the last entry in the scanned array, and (*iii*) the amount of time in milliseconds (ms) required to perform the scan operation.

Answer:
For exclusive scan operation, the program code file named 'scan.c' is to be compiled and executed './scan prateekfile.bin'. For generating random inputs ranging from -10 to 10, just make a change in 'line 28: n= rand()%20-10;' and input files from 2^5 to 2^12 can be generated in the same manner by making changes in 'line22: x=pow(2,5);……x=pow(2,12);'. After making the above changes in 'p3.c' then compiling and running it. Compile the file'scan.c' and execute it. It will print the number of integers read in, the last entry in scanned array and the amount of time in milliseconds (microseconds * 1000) taken to perform the scan operation.

b. Run a scaling analysis much like you did for 3.c above but this time only for your algorithm. Test your code with an input file with 2^5, 2^6,…, 2^12 integers randomly generated with values between -10 and 10. Produce the plot and provide the regression function in the readme file.

Answer:

After compiling and running the program codes 'p3.c' and 'scan.c' for values ranging from (2^5 ….. 2^12), I plotted the Regression curve in .png format as follows:



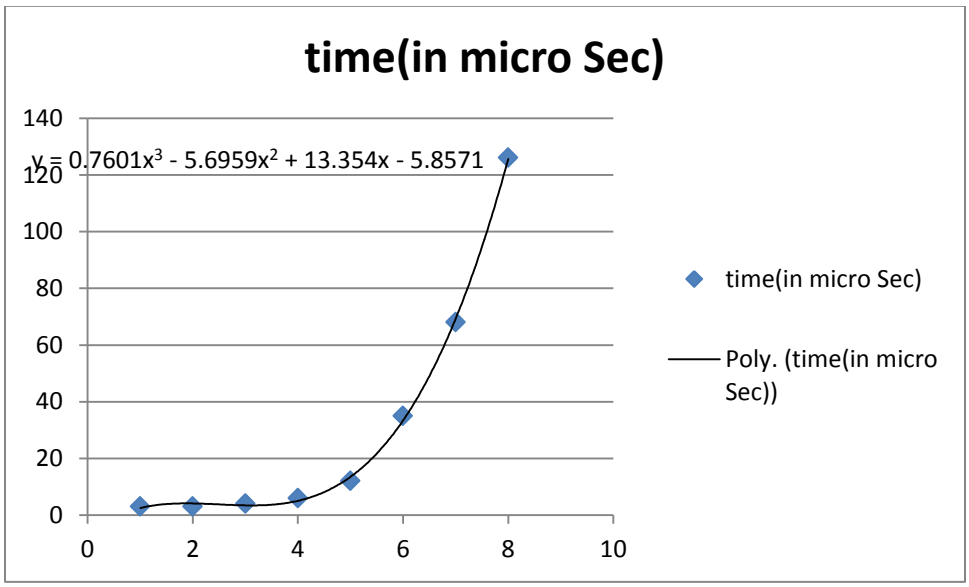
REGRESSION CURVE

In Red: the curve for Ordering time (in millisconds) versus number of integers.

ANALYSIS:

The Ordering time taken for values ranging from 2^5 to 2^12 is not a linear curve.

The following regression function is the outcome of the analysis:



Regression function:

Y= 0.7601x^3 – 5.6959x^2 + 13.354x – 5.8571