

Taylor Coogan  
ELE 408  
February 28, 2017

## Homework #2

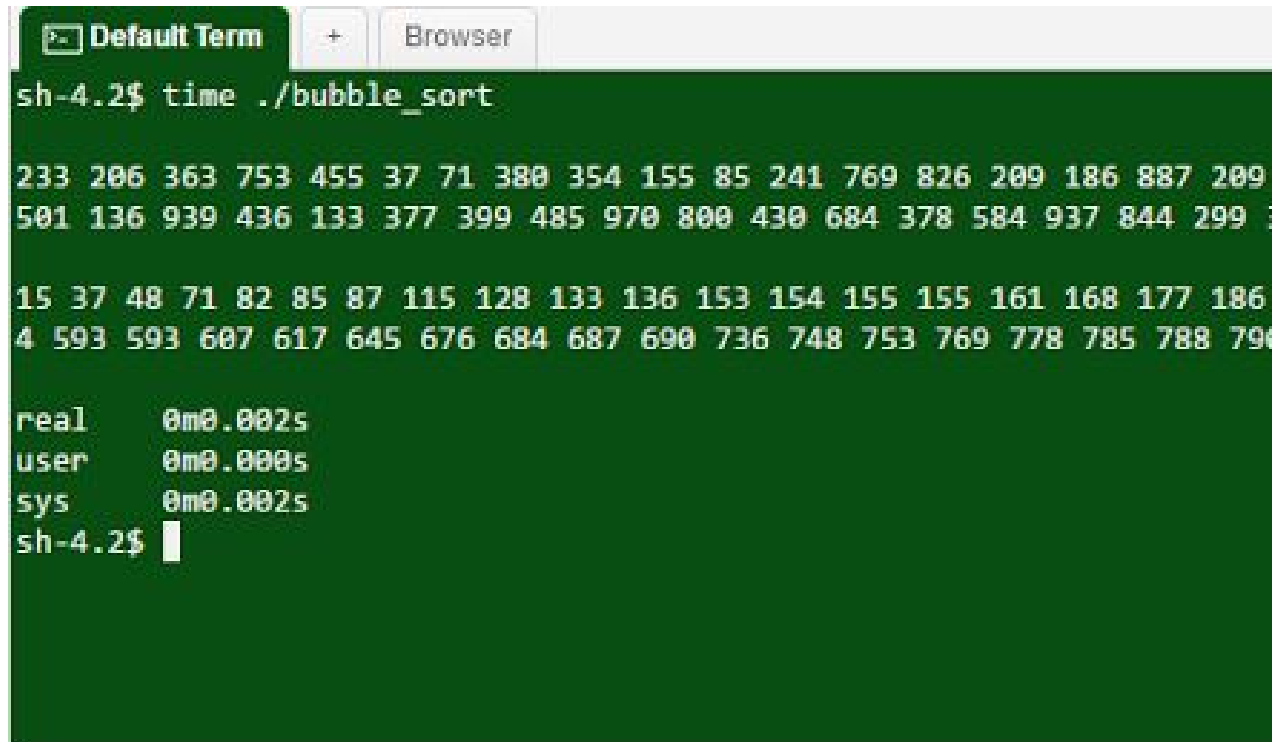
**Selection Sort:** For the selection sort program I followed the algorithm explained in the ELE 408 slides. This means that to sort the array, I started with the first unsorted number and then went through the rest of the array to find a smallest number. If a smallest number is smaller than the unsorted number, we switch the unsorted number and found smallest number. Then we repeat this process for the next unsorted number, all the way up to the end of the array,  $n-1$  times. I generated the random numbers by looping through the array and generating a random modulo 1000 because we only want numbers 1 - 1000. I seeded the random number generator with the time so that it was a more effectively random. Then I printed the results to the console.

*(The images were hard to display here without cropping, the full images are in the zip file)*

```
sh-4.2$ gcc main.c -o selection_sort
sh-4.2$ time ./selection_sort

407 774 641 219 362 619 744 627 555 658 965 413 155 91 45 51 80 694 761 85
3 460 32 469 288 516 55 24 305 606 172 954 952 366 303 379 606 764 700 40
6 24 24 32 35 45 51 55 56 74 80 91 105 108 131 131 148 153 154 155 172 19
618 619 627 641 658 687 694 700 732 744 761 764 765 774 789 816 817 828 8
real    0m0.002s
user    0m0.001s
sys     0m0.001s
sh-4.2$
```

**Bubble Sort:** For the bubble sort program, everything is the same except the sorting algorithm. This time we just go through each value in the array, and if it is smaller than the one before it, we switch the two, then continue to the end. We repeat this process  $n-1$  times so that the array is completely sorted.

A terminal window with a dark green background and light green text. The window has a title bar with a 'Default Term' tab, a '+' button, and a 'Browser' button. The terminal shows the command 'sh-4.2\$ time ./bubble\_sort' followed by two lines of 20 numbers each, representing an array of integers. Below the numbers, the execution time is displayed: 'real 0m0.002s', 'user 0m0.000s', and 'sys 0m0.002s'. The prompt 'sh-4.2\$' is visible at the bottom with a cursor.

```
sh-4.2$ time ./bubble_sort

233 206 363 753 455 37 71 380 354 155 85 241 769 826 209 186 887 209
501 136 939 436 133 377 399 485 970 800 430 684 378 584 937 844 299 3

15 37 48 71 82 85 87 115 128 133 136 153 154 155 155 161 168 177 186
4 593 593 607 617 645 676 684 687 690 736 748 753 769 778 785 788 790

real    0m0.002s
user    0m0.000s
sys     0m0.002s
sh-4.2$
```

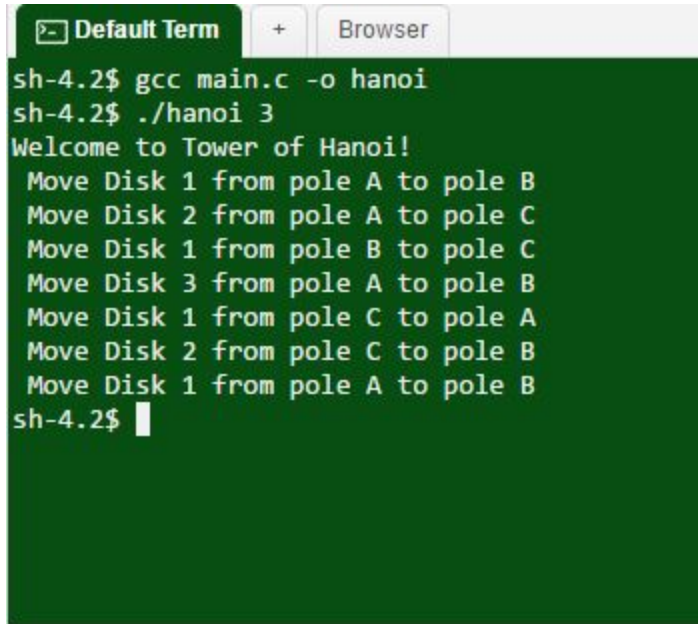
**Merge Sort:** For merge sort, everything is the same again except for the sorting algorithm. This time we recursively split the array in half and sort the numbers, then merge them back together. This is done recursively until there is only one number, which obviously cannot be split in half.

```
Default Term + Browser
sh-4.2$ gcc main.c -o mergesort
sh-4.2$ time ./mergesort
Original Array:
383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426 172 736 2
36 505 846 729 313 857 124 895 582 545 814 367 434 364 43 750 87 808 276 178 78

Results:
0 11 12 22 27 42 43 58 59 60 67 69 84 87 91 94 123 124 135 167 170 172 178 198
586 649 651 676 690 729 736 739 750 754 763 777 782 784 788 793 802 808 814 846

real    0m0.002s
user    0m0.001s
sys     0m0.000s
sh-4.2$
```

**Tower of Hanoi:** For this program we also use recursion. The user can input a number of disks to play the game with. This is taken as a command line argument and passed as the value for number of disks. Then the program recursively uses the hanoi function which basically takes breaks the process down into several “chunks” of the way that 2 disks are handled. The algorithm for two disks is repeated as many times as necessary for the actual number of disks until the game is completed. The results are printed to the console throughout the process.

A screenshot of a terminal window with a dark green background and white text. The window has a title bar with a 'Default Term' tab and a 'Browser' button. The terminal shows the following commands and output:

```
sh-4.2$ gcc main.c -o hanoi
sh-4.2$ ./hanoi 3
Welcome to Tower of Hanoi!
Move Disk 1 from pole A to pole B
Move Disk 2 from pole A to pole C
Move Disk 1 from pole B to pole C
Move Disk 3 from pole A to pole B
Move Disk 1 from pole C to pole A
Move Disk 2 from pole C to pole B
Move Disk 1 from pole A to pole B
sh-4.2$
```

**Monte Carlo:** For the monte carlo program, we estimate the value of Pi by randomly generating a large number of values within a 1x1 square. For every x,y coordinate generated, we check if the point is inside of a circle with radius 1. If the point is in the circle, we increment a counter. After we generate all the samples, we divide the number of points in the circle by the total number of points in the sample size. This should give us  $\pi/4$ . We multiply that value by 4 and it gives us an approximation of Pi. We notice that the higher the sample size, the closer the approximation to Pi becomes. I tried to create the best graph I could that shows this relationship.

```
sh-4.2$ gcc main.c -o montecarlo
sh-4.2$ ./montecarlo
Welcome to the Monte Carlo Party

Sample Size: 1000

Estimation of Pi: 3.160000
sh-4.2$
```

```
sh-4.2$ gcc main.c -o test
sh-4.2$ ./test
Welcome to the Monte Carlo Party

Sample Size: 100000000

Estimation of Pi: 3.141853
sh-4.2$
```



**Queue Problem:** For the queue problem, we use the queue length formula to simulate average queue length. The average queue length is the last queue length, plus arrival rate, minus service rate. So we simulate this by generating a random value by deciding on a large sample size, and then generating random values between 0 and the max for arrival time (and service time for some parts). For each sample we calculate the average queue length, and if it goes negative we just reset it to zero because we do not want a negative waiting time. And then after we run all the samples, we report that average waiting time and print it to a text file. For the second and third part, we change the distribution of the arrival time and service time to the Bernoulli distribution and M distribution and compare the times.

