1.) Consider the following pseudo-code. Suppose that the code begins executing via a call to `practicePointers()`.

```c
const int size = 5;
void doStuff(char** theArray, char* newName, char* newName2)
{
    for(int c=0; c<size; c++)
    {
        theArray[c] = (c%2 == 1) ? (newName+c):(newName2+c);
    }
}
void practicePointers()
{
    char ptr[size] = {'a', 'b', 'c', 'd', 'e'};
    char* ptr2 = "fghij"; // "'f', 'g', 'h', 'i', 'j"
    char** ptrArray = new char*[size];
    doStuff(ptrArray, ptr2, ptr);
    char newArray[size];
    // A
    for(int c=0; c<size; ++c)
        newArray[c] = **(ptrArray+c*sizeof(char));
    //B
    ++(*ptr);
    doStuff(ptrArray, ptr2, ptr);
    //C
    for(int c=0; c<size; ++c)
        newArray[c] = **(ptrArray+c*sizeof(char));
    //D
}
```

Now, answer each of the questions below. For parts (a) through (d), show your work by drawing out the memory locations that are created, along with the values stored in the memory at each step.

a.) (2.5 points) What are each of the entries in `ptrArray` at point "A" of the program’s execution?
b.) (2.5 points) What are each of the entries in `newArray` at point "B" of the program's execution?

c.) (2.5 points) What are each of the entries in `ptrArray` at point "C" of the program's execution?

d.) (2.5 points) What are each of the entries in `newArray` at point "D" of the program's execution?

e.) (2 points) Suppose that the code given above is a part of a larger program. Identify a problem in the code listed above, which the programmer should be careful of if he/she wants to avoid future debugging difficulties.

2.) (10 points) Suppose that there are two algorithms, Algorithm 1 and Algorithm 2. Algorithm 1 has order \( n \log(n) \), and Algorithm 2 has order \( n \). Algorithm 1 is being run on a computer having a great amount of memory, and a high processor speed. Algorithm 2 is being run on a TI-82 graphing calculator. Is it possible for Algorithm 1 take less computing time than Algorithm 2 for all size of data? If so, use the definition of Big-O to determine how much faster a single computation must execute on the system running Algorithm 1 to ensure that Algorithm 1 always takes less computing time than Algorithm 2. If not, prove that it is not possible using the definition of Big-O.

3.) (10 points) Prove that \( \log(n!) \) is in the exact order of \( n \log(n) \).

4.) Moved to HW2.

5.) (10 points) Recall the definition of a power set: “If X is a set, then the power set of X, denoted by \( P(X) \), is the set whose elements are the subsets of X.”

If \( S \) is the set \( \{a, b, c\} \), then the power set of \( S \) is

\[
P(S) = \{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.
\]

Write pseudo-code for a C++ function `nthPowerSet(int* nameOfSet, int n)` that computes the “\( n \)th” power set of `nameOfSet`, and returns it in the form of an array.

For example, if \( n \) is 2 and the passed-in array `nameOfSet` has the values \( \{1, 2, 3\} \), then your function should return the “power set of the power set” of the original array, \( \{1, 2, 3\} \).

So the call to `nthPowerSet` in the following code segment:

```cpp
int n=2;
int* nameOfSet = new int[3];
nameOfSet[0] = 1;
nameOfSet[1] = 2;
```
nameOfSet[2] = 3;
nthPowerSet(nameOfSet, n);

should return an array containing the values, \( P(P(\{1, 2, 3\})) = P(\{\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 3\}, \{1, 2, 3\}\}) = \{\{\}, \{\{1\}\}, \{\{2\}\}, \{\{3\}\}, \{\{1, 2\}\}, \{\{2, 3\}\}, \{\{1, 3\}\}, \{\{1, 2, 3\}\}\}) = \{\{\}, \{\{1\}\}, \{\{2\}\}, \{\{3\}\}, \{\{1, 2\}\}, \{\{2, 3\}\}, \{\{1, 3\}\}, \{\{1, 2, 3\}\}\}) = \{\{\}, \{\{1\}\}, \{\{2\}\}, \{\{3\}\}, \{\{1, 2\}\}, \{\{2, 3\}\}, \{\{1, 3\}\}, \{\{1, 2, 3\}\}\}) \}

You may assume that only positive integers can be used as elements of the `nameOfSet`. To denote a null element, your algorithm should use the numerical value of -1.

6.) (10 points) Using the operation count accounting rules discussed in class, give an estimate of the complexity of the algorithm shown below:

```c
int fn(int a, int b)
{
    int c = 0;
    a = (a >= 0) ? a : ((-1)*a);
    b = (b >= 0) ? b : ((-1)*b);

    if (b > a)
    {
        int tmp = b;
        b = a;
        a = tmp;
    }
    int temp;
    while (b > 0)
    {
        temp = b;
        b = a % b;
        a = temp;

        for(int j=(a-b); j>=0; j--)
        {
            c *= j/2;
        }
    }
    return (a+c);
}
```

7.) Assume that you are the head of database operations at ITCS and are responsible to maintain a database of all students enrolled at the University. You decide to implement a separate-chaining hash table structure with a hash table array of size 100. (Refer to figure 8.3 for a graphical representation of a hash table array of size 18, having 12 data entries). Your task is to develop a hashing algorithm to map a student body of 40,000 students into your hash table (of size 100) with the following priorities (starting with the topmost priority).

   i. Minimize collisions
ii. Spread keys evenly
iii. Computational performance

Use the student’s Last Name for the hash key.

a.) (4.5 points) Develop and explain your hash key algorithm.
b.) (4.5 points) Draw a graphical representation of your data structure to demonstrate that your algorithm works. (Similar to figure 8.3 with close to 20 data entries and hash table array of size 5.)
c.) **Deleted.**