1. (*15 Points*) State the precise mathematical definitions for $O(g(n))$, $\Omega(g(n))$, and $\Theta(g(n))$.

   Solution:

   $f(n)$ is $O(g(n))$ if there exist positive numbers c and N such that $0 \leq f(n) \leq cg(n)$ for all $n \geq N$.

   $f(n)$ is $\Omega(g(n))$ if there exist positive numbers c and N such that $f(n) \geq cg(n) \geq 0$ for all $n \geq N$.

   $f(n)$ is $\Theta(g(n))$ if there exist positive numbers c_1, c_2, and N such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq N$.

2. (*15 Points*) Complexity Proofs

   (a) Using the definition of $O(g(n))$, prove that $4n^2 + 2n + 5$ is $O(n^2)$.

   Solution: Need to find constants $c, n_0 > 0$ such that $4n^2 + 2n + 5 \leq n^2$ for all $n \geq n_0$. In other words, $4 + 2\frac{1}{n} + 5\frac{1}{n^2} \leq c$. Since $\frac{1}{n}$ is decreasing we can choose $c = 11$ and $n_0 = 1$.

   (b) Using the definition of $O(g(n))$, prove that 2^n is $O(n!)$.

   Solution: Need to find constants $c, n_0 > 0$ such that $2^n \leq cn!$ for all $n \geq n_0$.

   $$2^n = 2 \cdot 2 \cdot 2 \cdot 2 \cdots 2 \leq 2 \cdot 2 \cdot 3 \cdot 4 \cdots n = 2 \cdot 1 \cdot 2 \cdot 3 \cdot 4 \cdots n = 2n!$$

   So we can let $c = 2$ and $n_0 = 1$.

3. (*10 Points*) Find the exact number of times the inner loop body is executed and state the computational complexity of the loop.

   (a)
```
for (i = 0; i < n - 1; i++)
  for (j = i+1; j < n; j++) {
    tmp = a[i][j];
    a[i][j] = a[j][i];
    a[j][i] = tmp;
  }
```

   Solution: Taking the two loops together. When $i = 0$, the j loop is done $n - 1$ times, when $i = 1$, the j loop is done $n - 2$ times, and so on. The last iteration is when $i = n - 2$ in which case the j loop is done one time. So the total is,

   $$\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

   So the complexity is n^2.

   (b)
```
for (i = 0; i < n; i *= 2)
  for (j = 0; j < n; j++)
    for (k = a[i][j] = 0; k < n; k++)
      a[i][j] += b[i][k] * c[k][j];
```

   Solution: Trick question, if $n > 0$ this is an infinite loop. If $n \leq 0$ then the loop is not executed at all.

4. (*15 Points*) Quick Sort:

   (a) What is the computational complexity of the Quick Sort in the best and worst cases?
   Solution: Best is $n\lg(n)$ and worst is n^2.

   (b) Given the following array as the initial array,

7	2	5	15	8	6	3	9	1	12

   i. We will use the pivot as the middle element of the array, as we did in our implementation in class. What is the pivot? Also, display the array after the partition stage and before the recursive calls.
   Solution: The pivot is 8. The array after partitioning is

7	2	5	1	3	6	8	9	15	12

   ii. Show the portions of the array that are sent to the two recursive calls to the quick sort function. In each case, state the pivot and display the subarrays after their partitioning and before the next recursive calls.
   Solution: For the left portion of the array the subarray is

7	2	5	1	3	6

   The pivot here is 5, and after it is partitioned we have

3	2	1	5	7	6

   For the right portion of the array the subarray is

8	9	15	12

   The pivot here is 9, and it is not altered by the partitioning process.

5. (*10 Points*) Shell Sort: Given the following array, display the result of the Shell sort after one pass using 5 subarrays. Then display the result of this output on a second pass using 3 subarrays.

7	2	5	15	8	6	3	9	1	12

Solution:

6	2	5	1	8	7	3	9	15	12

1	2	5	3	8	7	6	9	15	12

6. (*15 Points*) Heap Sort:

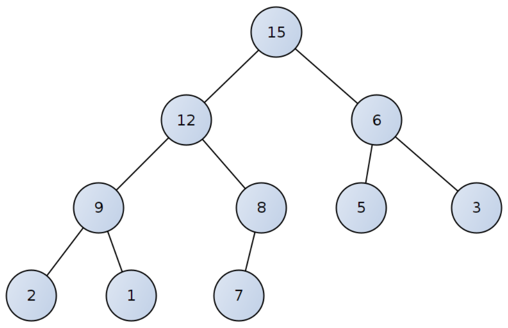
   (a) What is the computational complexity of the Heap Sort in the worst cases?
   Solution: Worst case is $n \lg(n)$.

   (b) Given the following array, Display both the binary tree representation and the resulting array after the initial heap construction is finished but before the sorting algorithm phase starts.

7	2	5	15	8	6	3	9	1	12

   Solution:

15	12	6	9	8	5	3	2	1	7



7. (*10 Points*) Lambda Expressions

   (a) Write a lambda expression that will find the minimum of two values. It should work with the following block of code and produce the corresponding output.

```
cout << Min(7, 3) << endl;
cout << Min(2, 31) << endl;
cout << Min(2.35, 31) << endl;
string str1 = "Hello";
string str2 = "World";
cout << Min(str1, str2) << endl;
cout << endl;
```

Output:

```
3
2
2.35
Hello
```

   Solution:

```
auto Min = [](auto a, auto b) { return (a < b) ? a : b; };
```

   (b) Write a lambda expression that will find the factorial of an input long integer. It should work with the following block of code and produce the corresponding output. Write the function using iteration to find the factorial, do not use recursion.

```
cout << factorial(3) << endl;
cout << factorial(5) << endl;
cout << factorial(10) << endl;
```

Output:

```
6
120
3628800
```

   Solution: The first is what I would expect you to do, the second is a one line recursive solution which requires the functional library.

```
auto factorial = [](long a) {
  long f = 1;
  for (long i = 1; i <= a; i++)
    f *= i;
  return f;
};

function<long(long)> factrec = [&factrec](long n) { return n > 1 ? n * factrec
    (n - 1) : 1; };
```

8. (*10 Points*) Function Pointers: Given the following program.

```cpp
#include <algorithm>
#include <iostream>

using namespace std;

template <class T, class R> using TtoR = R (*)(const T &);
template <class T, class R> using TtoRA = R *(*)(const T *);
template <class T> void print(const T &a) { cout << a << " "; }
double itod(const int &a) { return a; }
int dtoi(const double &a) { return a; }
double *itodA(const int *a) {
  double *arr = new double[7];
  transform(a, a + 7, arr, itod);
  return arr;
}
int *dtoiA(const double *a) {
  int *arr = new int[7];
  transform(a, a + 7, arr, dtoi);
  return arr;
}

int main() {
  int arr1[7] = {2, 5, 6, 9, 10, 1, 1};
  double arr2[7] = {4.3, 4.9, 2.1, 9.125, 0.2, 3.14159, 2.7182818};
  double *arrP1;
  int *arrP2;

  // a.
  TtoR<string, void> f = print;
  f("Help Me!");
  // b.
  TtoR<int, void> f2 = print;
  f2(3.14159);
  // c.
  TtoR<int, double> f3 = itod;
  cout << f3(12345) << endl;
  // d.
  TtoR<double, int> f4 = dtoi;
  cout << f4(3.14159) << endl;
  // e.
  TtoRA<int, double> f5 = itodA;
  arrP1 = f5(arr1);
  for_each(arrP1, arrP1 + 7, print<double>);
  cout << endl;
  // f.
  TtoRA<double, int> f6 = dtoiA;
  arrP2 = f6(arr2);
  for_each(arrP2, arrP2 + 7, print<int>);
  cout << endl;

  delete [] arrP1;
  delete [] arrP2;
  return 0;
}
```

In the above program there are 6 blocks of code labeled with comments a–f. Of these 6 blocks of code which ones will compile and which ones will not. Of those that will compile what is the output of that block of code?

Solution: All of the blocks of code will compile and the output is,

```
Help Me! 3 12345
3
2 5 6 9 10 1 1
4 4 2 9 0 3 2
```