Task 2.1P

Question 1

Review the following algorithms (assume any undeclared variables are declared earlier)

i.
```
double a = 0;
a += rand();
if( a < 0.5 ) a += rand();
if( a < 1.0 ) a += rand();
if( a < 1.5 ) a += rand();
if( a < 2.0 ) a += rand();
if( a < 2.5 ) a += rand();
if( a < 3.0 ) a += rand();
```

a) What is the number of operations of the best, worst and average cases?
- Best Case: In the ideal scenario, `rand()` yields results so that, following the initial addition, `a` is either greater than or equal to 3.0 One operation would come from this.
- Worst Case: When `rand()` yields results that satisfy every requirement, it is considered the worst case scenario. This would necessitate seven procedures.
- Average Case: The number of requirements `a` will meet is about 3-4 times on average, assuming `rand` produces evenly distributed random numbers. As a result, 4-5 surgeries are performed on average.

b) Describe the best, worst and average case using Big-Θ notation.
- Best Case: Θ(1)
- Worst Case: Θ(1)
- Average Case: Θ(1)

c) Describe each algorithm's overall performance using the tightest possible class in Big-O notation.
- Overall Performance: O(1)

d) Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation.
- Overall Performance: Ω(1)

e) Describe each algorithm's overall performance using Big-Θ notation.
- Overall Performance: Θ(1)

f) Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation.
- Since this algorithm has constant time complexity, Θ (1) best captures its performance.

ii.
```
int count = 0;
for (int i = 0; i < N; i++)
{
    int num = rand();
    if( num < 0.5 ){
        count += 1;
    }
}
```

a) What is the number of operations of the best, worst and average cases?
   - Best Case: Assuming perfect conditions, `num < 0.5` never occurs, leading to `N` operation.
   - Worst Case: In the worst scenario, there are `N` operations because `num < 0.5 occurs each time.
   - Average Case: If `rand () ` produces evenly distributed random numbers on average, then `num < 0.5` will be satisfied in half of the `N` repetitions. As a results, there are `N` operation.

b) Describe the best, worst and average case using Big-Θ notation.
   - Best Case: Θ(N)
   - Worst case: Θ(N)
   - Average Case: Θ(N)

c) Describe each algorithm's overall performance using the tightest possible class in Big-O notation.
   - Overall Performance: O(N)

d) Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation
   - Overall Performance: Ω(N)

e) Describe each algorithm's overall performance using Big-Θ notation.
   - Overall Performance: Θ(N)

f) Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation.
   - This algorithm's linear time complexity makes Θ (N) the ideal technique to characterize its running performance.

iii.
```
int count = 0;
for (int i = 0; i < N; i++)
{
    if (unlucky)
    {
        for (j = N; j > i; j--)
        {
            count = count + i + j;
        }
    }
}
```

a) What is the number of operations of the best, worst and average cases?
   - Best Case: The optimal scenario is when `unlucky` is not true. `N` operations follow from this.
   - Worst Case: In the worst scenario, `unlucky is accurate. As a consequence, the inner loop executes `N (N+1)/2` times. Therefore, the number of operations is a function of N^2
   - Average Case: The average instance in the scenario where `unlucky` is true occasionally will depend on the likelihood that `unlucky` will be true. `N/2 + N^2/2` would be the average amount of operations if `unlucky` is true 50% of the time.

b) Describe the best, worst and average case using Big-Θ notation.
   - Best Case: Θ(N)
   - Worst Case: Θ(N^2)
   - Average Case: Θ(N^2)

c) Describe each algorithm's overall performance using the tightest possible class in Big-O notation.
   - Overall Performance: O(N^2)

d) Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation
   - Overall Performance: Ω(N)

e) Describe each algorithm's overall performance using Big-Θ notation.
   - Overall Performance: Θ(N^2)

f) Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation.
   - Since this method depicts the worst-case situation and dominates other cases, Θ (N^2) best captures its performance.

iv.

```
int count = 0;
int i = N;

if (unlucky)
{
    while (i > 0)
    {
        count += i;
        i /= 2;
    }
}
```

a) What is the number of operations of the best, worst and average cases?
   - Best Case: The loop is never carried out if `unlucky` is false. [O (1)]
   - Worst Case: The while loop is executed if `unlucky` is true. Logarithmic iterations results from halving `i` in each iteration. [O(log N)]
   - Average Case: Despite the fact that the loop operates logarithmically when `unlucky` is true, the logarithmic term nevertheless dominates the predicted complexity when `unlucky` is assumed to be true at random. [O(log N)]

b) Describe the best, worst and average case using Big-Θ notation.
   - Best Case: Θ(1)
   - Worst Case: Θ(log N)
   - Average Case: Θ(log N)

c) Describe each algorithm's overall performance using the tightest possible class in Big-O notation.
   - Overall performance: O(log N)

d) Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation
   - Overall performance: Ω(1)

e) Describe each algorithm's overall performance using Big-Θ notation.
   - Overall performance: Θ(log N)

f) Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation.
   - Θ(log N)

v.

```
int count = 0;

for (int i = 0; i < N; i++)
{
    int num = rand();
    if( num < 0.5 )
    {
        count += 1;
    }
}
int num = count;
for (int j = 0; j < num; j++)
{
     count = count + j;
}
```

a) What is the number of operations of the best, worst and average cases?
   - Best Case: `count` stays at 0 following the initial loop. [O(N)]
   - Worst Case: Since count is roughly N/2 the second loop iterates N/2 times. O(N+N/2) = O(N)
   - Average Case: Since the second loop iterates N/2 times, count is approximately N/2

b) Describe the best, worst and average case using Big-Θ notation
   - Best Case: Θ(N)
   - Worst Case: Θ(N)
   - Average Case: Θ(N)

c) Describe each algorithm's overall performance using the tightest possible class in Big-O notation
   - Overall performance: O(N)

d) Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation
   - Overall performance: Ω (N)

e) Describe each algorithm's overall performance using Big-Θ notation
   - Overall performance: Θ(N)

f) Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation
   - Θ(N)

vi.

```
for (int i = 0; i < N - 1; i++)
{
    for (int j = 0; j < N-i-1; j++)
    {
        if (a[j] > a[j+1])
        {
            Swap(a[j], a[j + 1]);
        }
    }
}
```

a) What is the number of operations of the best, worst and average cases?
   - Best Case: There's already a sort on this array. O (N) in the case of early departure optimization, but O (N 2) in the absence of it.
   - Worst Case: Reverse order sorting is applied to the array. O(N2)
   - Average Case: Array arranged at random O(N2)

b) Describe the best, worst and average case using Big-Θ notation
   - Best Case: $\Theta(N)$ with optimization, otherwise $\Theta(N^2)$
   - Worst Case: $\Theta(N^2)$
   - Average Case: $\Theta(N^2)$

c) Describe each algorithm's overall performance using the tightest possible class in Big-O notation
   - Overall performance: $O(N^2)$

d) Describe each algorithm's overall performance using the tightest possible class in Big- Ω notation
   - Overall performance: $\Omega(N)$ with optimization, otherwise $\Omega(N^2)$

e) Describe each algorithm's overall performance using Big-Θ notation
   - Overall Performance: Θ (N^2)

f) Selecting from one or more of the above which is the best way to succinctly describe the performance of each algorithm using asymptotic notation
   - $\Theta(N^2)$

Question 2
Arguably, the most commonly used asymptotic notation used is frequently Big-O. Discuss why this is so commonly the case.

- Upper Limit of Attention
  - An upper constraint on an algorithm's time or space complexity is provided by Big-O notation, which is especially helpful for comprehending the worst-case scenario. For resilient systems that must ensure performance, it is essential to know the maximum resources that an algorithm may consume.
- Clarity and Simplicity
  - When compared to other notation such as Big-Θ and Big- Ω, Big-O notation is easier to understand and less complex. It enables programmers and analysts to immediately understand the possible rate of increase in the amount of resources used by an algorithm
- Generally Recognized Standard
  - Comparing algorithms is made simple by Big-O notation. It concentrates on the dominating term abstracting away lower-order terms and constant factors, which makes comparing the scalability of various algorithms easy.
- Extra Careful Estimate
  - System are guaranteed to be constructed with a safety margin since Big-O gives an upper bound. In crucial applications where going over resource constraints can have dire repercussions, this cautions strategy is advantageous.
- Flexibility
  - Numerous issues and algorithms, from straight forward loops to intricate recursive functions, can be solved using Big-o notation. The tool's versatility in algorithm analysis stems from its universality.
- Historical priority
  - Big O notation has been the standard tool for complexity analysis for a long time because of its historical use in basic computer science textbooks and courses. Its widespread use even now is influenced use by this heritage.
- Pedagogical Intentions
  - Because of its ease of use and simplicity, Big-O notation is frequently the first asymptotic notation taught in computer science courses. Algorithm analysis gains a solid foundation tanks to this early introduction.
- Thorough understanding
  - The most crucial component of performance analysis in large-scale systems is frequently understanding how an algorithm will function as the input size increases. By emphasizing the upper bound, Big-O notation provides a clear picture of this behavior.

- Big-O notation is widely used because, to put it briefly, it is straightforward, standardized, easy to compare, conservative, versatile, historically precedented, teaches a lot, and can shed light on scalability. It is a vital tool for algorithm creation and analysis because of these factors.

Question 3

Is it true that $\theta\ (n)$ algorithm always takes longer to run than an $\theta(\log n)$ algorithm? Explain your answer.

- No,
- Fixed Elements and lower Order Elements
    - Constant factors are ignored in Θ notation. For small n, a Θ (log n) method with a high constant might be slower than a Θ (log n) algorithm with a small constant.
- The Size of the input
    - Since there is not much of a difference between n and log n log n for tiny input values, the actual runtimes may be near. When n is big, Θ (n) will typically take longer than Θ (log n).
- Change in Implementation
    - Real runtimes can be impacted by optimizations and particular implementations, which can occasionally make a Θ (n) method faster in practice.
- Best, Worst, Average case
    - Different algorithms may perform differently in average, worst, and best cases. There any be situations in which an average case Θ (n) method performs faster than a Θ (log n) algorithm.
- Essentially, although while Θ(n) grows faster than Θ(log n) asymptotically, this does not imply that a Θ(n) algorithm will always run longer in practice than a Θ(log n) algorithm, particularly for smaller input quantities or particular implementations.

Question 4

Answer whether the following statements are right or wrong and explain your answers.

- $2n^2 + 6^{13}\ n = O(n^2)$
   - Right
   - In Big-O notation, we consider the term with the highest growth rate. Here, $2n^2$ grows faster than 613n as n becomes large. Therefore, $2n^2$=613n is dominated by the $2n^2$ term, making $2n^2$+613n= $O(n^2)$

- $nlogn = O(n)$
   - Wrong
   - For n log n to be O(n), there would need to be a constant c such that n log n ≤ cn all sufficiently large n. However n log n grows faster than n because the log n term increases with n. Thus, n log n is not O(n)

- $n^3 + n^2 + 10^6 n = \theta\ (n^4)$
   - Wrong
   - In Big-Theta notation, we consider the term with the highest growth rate to describe an asymptotic tight bound. Here, the highest growth rate term is n^3, not n^4. Therefore, $n^3+n^2+10^6n$ is dominated by the n^3 term, making it $\Theta(n^3)$, not $\Theta(n^4)$

- $nlogn = \Omega(n)$
   - Right
   - Big-O mega notation describes an asymptotic lower bound. For n log n to be Ω(n), there must exist a constant c such that n log n ≥ cn for all sufficiently large n. Since log n is positive and increases with n, n log n grows faster than n. therefore, n log n= Ω(n)