Task 2.1P

Name- Anneshu Nag
Student ID- 2210994760

1. Checking the number of Operations performed (Worst, Best & Average Cases).

a) Calculating the worst case for the scenarios

S. No	Algorithm	Operation count per line	Total

i.	num = rand();
	double a = num;
	if(a < 0.5) a += num;
	if(a < 1.0) a += num;
	if(a < 1.5) a += num;
	if(a < 2.0) a += num;
	if(a < 2.5) a += num;
	if(a < 3.0) a += num;
		1
		1
		1 + 1
		1 + 1
		1 + 1
		1 + 1
		1 + 1
		1 + 1
			W = 14
			B = 8
			A = 11

ii.	int count = 0;
	for (int i = 0; i < N; i++){
	if(rand() < 0.5){
	count += 1;
	}
	}
		1
		1 + (N+1) + N
		N x (1)
		N x (1)
			W = 4N + 3
			B = 3N + 3
			A = 7N/2 + 3

iii.	int count = 0;
	for (int i = 0; i < N; i++) {
	if (unlucky){
	for (j = N; j > i; j--){
	count = count + i + j;
	}
	}
	}
		1
		1 + (N+1) + N
		N x (1)
		N x (1 + ((N+1)/2 + 1) + (N + 1)/2)
		N X (N+1)/2 x (1)
			W = $3N^2/2 + 13N/2 + 3$
			B = 3N + 3
			A = $3N^2/4 + 19N/4 + 3$

iv.	int count = 0;
	int i = N;
	if (unlucky) {
	while (i > 0){
	count += i;
	i /= 2;
	}
	}
		1
		1
		1
		log N + 1
		log N
		log N
			W = 3logN + 4
			B = 3
			A = $3\log N/2 + 4\frac{1}{2}$

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;
	}
		1
		1 + (N+1) + N
		N x (1)
		N x (1)
		N x (1)
		
		1
		1 + (num + 1) + num
		num x (1)
			W = 8N + 6
			B = $4N + 6$
			A = $6N + 6$

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]);
	}
	}
	}
		1 + N + (N-1) = 2N
		(N-1) x (1 + (N/2 + 1) + N/2) = N^2 - N – 2
		(N-1) x (N/2) x (1) = $N^2/2 – N/2$
		(N-1) x (N/2) x (1) = $N^2/2 – N/2$
			W = $2N^2$ - 2
			B = $3N^2/2$ + N/2 - 2
			A = $7N^2/4$ + N/4 - 2

b) Describing the equations for the best, worst and average cases in Big-Θ notation.

i. **Best Case = 8**
Here Ω (1) and O (1) so Θ (1).

Worst Case = 14
Here Ω (1) and O (1) so Θ (1).

Average Case = 8
Here Ω (1) and O (1) so Θ (1).

ii. **Best Case = 3N + 3**
Here Ω (N) and O (N) so Θ (N).

Worst Case = 4N + 3
Here Ω (N) and O (N) so Θ (N).

Average Case = 7N/2 + 3
Here Ω (N) and O (N) so Θ (N).

iii. **Best Case = 3N + 3**
Here Ω (N) and O (N) so Θ (N).

Worst Case = $3N^2/2$ + 13N/2 + 3
Here Ω (N^2) and O (N^2) so Θ (N^2).

Average Case = $3N^2/4$ + 19N/4 + 3
Here Ω (N^2) and O (N^2) so Θ (N^2).

iv. **Best Case = 3**
Here Ω (1) and O (1) so Θ (1).

Worst Case = 3logN + 4
Here, Ω (log N) and O (log N) so Θ (log N).

Average Case = 3logN/2 + 9/2
Here, Ω (log N) and O (log N) so Θ (log N).

v. **Best Case = 4N + 6**
Here Ω (N) and O (N) so Θ (N).

Worst Case = 8N + 6
Here Ω (N) and O (N) so Θ (N).

Average Case = 6N + 6
Here Ω (N) and O (N) so Θ (N).

vi. **Best Case = $3N^2/2 + N/2 - 2$**
Here Ω (N^2) and O (N^2) so Θ (N^2).

Worst Case = $2N^2 - 2$
Here Ω (N^2) and O (N^2) so Θ (N^2).

Average Case = $7N^2/4 + N/4 - 2$
Here Ω (N^2) and O (N^2) so Θ (N^2).

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

i. **Worst Case: O (1)**

ii. **Worst Case: O (N)**

iii. **Worst Case: O (N^2)**

iv. **Worst Case: O (log N)**

v. **Worst Case: O (N)**

vi. **Worst Case: O (N^2)**

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

i. **Best Case: Ω (1)**

ii. **Best Case: Ω (N)**

iii. **Best Case: Ω (N)**

iv. **Best Case: Ω (1)**

v. **Best Case: Ω (N)**

vi. **Best Case: Ω (N^2)**

e) Describe each algorithm's overall performance using Big-Θ notation.

We know that Big-Θ notation = Big-Ω ∩ Big-O so Big-Θ must be the intersection of Big-O and Big-Ω.

i. **Since there is O (1) and Ω (1), therefore Θ (1).**

ii. **Since there is O (N) and Ω (N), therefore Θ (N).**

iii. **Since there is O (N^2) and Ω (N), therefore we can't find Big-Θ for this case.**

iv. **Since there is O (log N) and Ω (1), therefore we can't find Big-Θ for this case.**

v. **Since there is O (N) and Ω (N), therefore Θ (N).**

vi. **Since there is O (N^2) and Ω (N^2), therefore 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.

i. **Overall the algorithm performance can be best represented by T (n) = Θ (1) as O (1) and Ω (1) are the same and Big-Θ gives the information regarding the tight bound i.e. it provides information about both the upper and lower bound, thereby making it more precise.**

ii. **Overall the algorithm performance can be best represented by T (n) = Θ (N) as O (N) and Ω (N) are the same and Big-Θ gives the information regarding the tight bound i.e. it provides information about both the upper and lower bound, thereby making it more precise.**

iii. **The algorithm performance can be represented by both T (n) = O (N^2) or Ω (N) as the performance of the algorithm varies from linear (best case) to quadratic (worst case).**

    The best way to represent the performance for this algorithm would be T (n) = O (N^2) as it tells about the upper bound/worst case of the algorithm.

iv. **The algorithm performance can be represented by both T (n) = O (log N) or Ω (1) as the performance of the algorithm varies from constant (best case) to logarithmic (worst case).**

    The best way to represent the performance for this algorithm would be T (n) = O (log N) as it tells about the upper bound/worst case of the algorithm.

v. **Overall the algorithm performance can be best represented by T (n) = Θ (N) as O (N) and Ω (N) are the same and Big-Θ gives the information regarding the tight bound i.e. it provides information about both the upper and lower bound, thereby making it more precise.**

vi. **Overall the algorithm performance can be best represented by T (n) = Θ (N^2) as O (N^2) and Ω (N^2) are the same and Big-Θ gives the information regarding the tight bound i.e. it provides information about both the upper and lower bound, thereby making it more precise.**

2. Discussing why Big-O is the most common asymptotic notation and why people do not use Big-Θ.

   Big-O notation is widely used for describing the upper bound of an algorithm's performance in the worst-case scenario, providing programmers with insights into its efficiency. This notation offers a straightforward and concise way to convey the algorithm's scalability without increasing complexity. It is due to these factors it is the most common asymptotic notation used by people.

   Meanwhile, people do not prefer Big-Θ notation over Big-O notation because Big-Θ gives a tight bound of the algorithm which is comprehensive and involves more mathematical reasoning. This is good in case of theoretical knowledge and research but in practical algorithm-design it does not help to solve the efficiency of the algorithm that makes is unsuitable for much use.

3. Checking if Θ (n^2) algorithm always takes longer to run than Θ (log n) algorithm.

   The above statement is false. It is because in practical scenario there will be a constant involved in the algorithm. The constant of the Θ (log n) algorithm could be a lot higher than the constant of the Θ (n^2) algorithm, so for small value of 'n', the Θ (log n) algorithm could take longer to run if the constant used along with it is extremely large.

4. Check whether the following statements are true or false.
   a) $n^2 + 16^{13}n = O(n^2)$
      True.
      Here, the definition of Big-O is satisfied for the above equation. O (n^2) sets the upper bound of the growth rate of the function that is proportional to n^2 or slower.
      There exists 'C' for which f (n) ≤ C (g (n)) is satisfied for the above equation.

   b) n log n = O (n)
      False.
      Here, the definition of Big-O isn't satisfied for the above equation. We know that 'n log n' is a greater growth function compared to 'n'.

   c) $n^2 + n + 10^6 = \Theta(n^3)$
      False.
      Here, the definition of Big-Θ isn't satisfied for the above equation. It doesn't satisfy the condition when Ω (n^3) thereby not fulfilling the condition for Big-Θ which states that it must satisfy both Big-O and Big-Ω to satisfy Big-Θ.

   d) $n \log n + 51n^2 = \Omega(n)$
      True.
      Here, the definition of Big-Ω is satisfied for the above equation. Ω (n) sets the lower bound of the equation and 'n^2' has greater growth rate than 'n'.
      The equation satisfies C (g (n)) ≤ f (n).