Q1

```
int a = 0, i = N;
while (i > 0) {
    a += i;
    i /= 2;
}
```

1. O(N)
2. O(Sqrt(N))
3. O(N / 2)
4. O(log N)

O(log N)

Q2

How is time complexity measured?	1. By counting the number of algorithms in an algorithm. 2. By counting the number of primitive operations performed by the algorithm on a given input size. 3. By counting the size of data input to the algorithm. 4. None of the above

2. By counting the number of primitive operations performed by the algorithm on a given input size

Q3

```
int i, j, k = 0;
for (i = n / 2; i <= n; i++) {
    for (j = 2; j <= n; j = j * 2) {
        k = k + n / 2;
    }
}
```

1. O(n)
2. O(N log N)
3. O(n^2)
4. O(n^2 Logn)

O(n Logn)

Q4

```
public static void fun1(int n) {
   for (int k = 0; k < n; k++) {
      System.out.println("Print");
  } // end for
}
```

1. O(n)
2. O(N log N)
3. O(n^2)
4. O(n^2 Logn)

O(n)

Q5

```
public static void fun2(int n) {
   for (int k = 0; k < n; k++){
      System.out.println("Println runs
in constant time");
   } // end for
   for (int k = 0; k < n; k++){
      System.out.println("Println runs
in constant time");
   } // end for
}
```

1. O(n)
2. O(N log N)
3. O(n^2)
4. O(n^2 Logn)

O(n)

Q6

`public static void fun3(int n) {`
`  for (int k = 0; k < n; k++){`
`  for (int j = 0; j < n; j++){`
`  System.out.println("QQQ");`
`  } // end for`
`  } // end for`
`}`
	1. O(n)
	2. O(N log N)
	3. O(n^2)
	4. O(n^2 Logn)

O(n^2)

Q7

`public static void fun5(int n) {`
`  for (int k = 0; k < n; k++){`
`  for (int j = 0; j < n; j++){`
`  for(int i = 0; i < n; i++)`
`  System.out.println("QQQ");`
`  } // end for`
`  } // end for`
`  } // end for`
`} // fun5`
	1. O(n)
	2. O(N log N)
	3. O(n^2)
	4. O(n^3)

O(n^3)

Q8 Selection Sort Time complexity

- Worst Case
- Best Case

---

1. **Worst case:** $O(n^2)$. Since we traverse through the remaining array to find the minimum for each element, the time complexity will become $O(n^2)$.
2. **Best case:** $O(n^2)$. Even if the array has already been sorted, our algorithm looks for the minimum in the rest of the array. As a result, the best-case time complexity is the same as the worst-case.

Q9 Insertion Sort Time complexity

- Wost Case
- Best Case

- **Worst case:** $O(n^2)$. In a worst case situation, our array is sorted in descending order. So, for each element, we have to keep traversing and swapping elements to the left.

- **Best case:** O(n). In the best case, our array is already sorted. So for each element, we compare our current element to the element at the left only once. Since the order is correct, we don't swap and move on to the next element. Hence the time complexity will be O(n).

Q10 Merge Sort Time complexity

- Worst Case
- Best Case

- **Worst case:** O(nlogn). The worst-case time complexity is the same as the best case.
- **Best case:** O(nlogn). We are dividing the array into two sub-arrays recursively, which will cost a time complexity of O(logn). For each function call, we are calling the partition function, which costs O(n) time complexity. Hence the total time complexity is O(nlogn).