1.1. For each algorithm below calculate its time complexity.

a)

```
for i in range(n):
  j = 0
  while j * j < i:
    j += 1
```

b)

```
for i in range(n):
  j = i
  while j > 0:
    j = j // 2
```

c)

```
def f(n):
  if n == 0:
    return 1
  else
    return 5 * f(n // 3)
```

d)

```
def f(n):
  if n == 0:
    return 1
  else
    return f(n // 3) + f(n // 3)
```

1.2. Prove using mathematical induction that if $T(n) = 2T(n/2) + n$, then $T(n) = \Omega(n \log n)$ (lower bound).

1.3. Prove using mathematical induction that if $T(n) = 2T(n/2 + 20) + n$, then $T(n) = O(n \log n)$.

1.4. Prove using mathematical induction that if $T(n) = \log n \cdot T(n/\log n) + n$, then $T(n) = O(n \log n)$.

1.5. Prove using mathematical induction that if $T(n) = 2T(\sqrt{n}) + 1$, then $T(n) = O(\log n)$.

1.6. You are given two arrays a and b sorted in non-decreasing order. Determine if there is a number that occurs in both arrays. Time $O(n)$.

1.7. You are given two arrays a and b sorted in non-decreasing order. Find i and j such that the difference $|a_i - b_j|$ is minimal. Time $O(n)$.

1.8. You are given two arrays a and b sorted in non-decreasing order and a number S. Find such i and j such that the sum $a_i + b_j = S$. Time $O(n)$.

1.9. You are given two arrays a and b sorted in non-decreasing order. Find the number of pairs (i, j) such that $a_i = b_j$. Time $O(n)$.

1.10. You are given two arrays a and b sorted in non-decreasing order. Find the number of pairs (i, j) such that $a_i > b_j$. Time $O(n)$.

1.11. Given an array a. The pair (i, j) such that $i < j$ and $a_i > a_j$ is called inversion. Find the number of inversions in array a. Time $O(n \log n)$

1.12. Show that, with the correct implementation, merge sort is stable (that is, does not change the order of equal elements).

1.13. Show how to implement merge sort with a single additional array of size n (without constructing new arrays in each recursive call).

1.14. Show how to implement merge sort without recursion.