Lab 3

Team members: Kerim Amansaryyev,
Muneeb Ahmed,
Mahri Ilmedova.

Question 1:

```
public static int beautiful(int[] A, int n){
    int sum = 0;
    for(int i = 0; i < n; i++){
        sum+=A[i];
    }
    return sum;
}
```

Algorithm analysis:
Here, as we loop through all the elements in an array the worst case is n, and the best case is also n as we will loop through all items.

Question 2:

2^n , 2^(n + 1), 2^(2n), 2^(2^n)

Question 3:

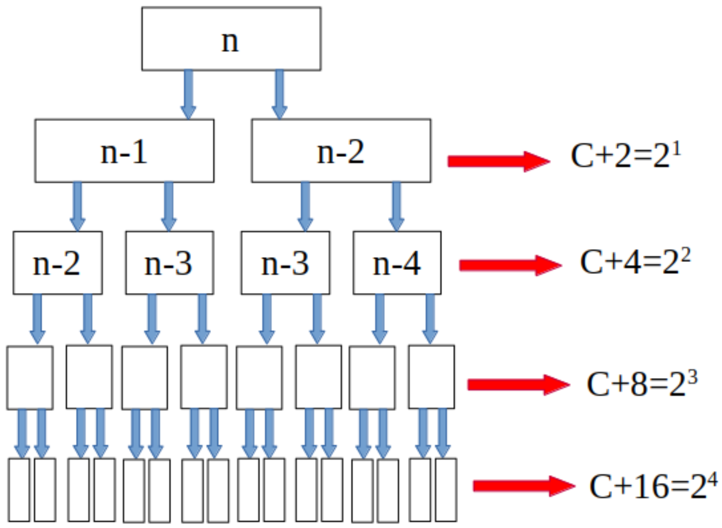
O(1) – constant operation, it can be any operation that costs constant
O(log n) – binary search
O(n) – linear search
O(n log n) – merge sort
O(n2) - insertion sort, bubble sort, selection sort
O(n3) - matrix multiplication
O(2n) - creating new array of size 2k

Question 4.

We cannot apply Master Theorem on Fibonacci Series. In order to get the Time Complexity for Fibonacci Series, consider the following: f(n) is equal to sum of f(n-1) and f(n-2), then:

$$T(n) = T(n-1) + T(n-2)$$

Now let's illustrate this formula so that it's more easy to understand it:



So, now as you can see on the illustration, you can easily now that the Time Complexity for T(n) is O(2n).

Question 5:

Example 1 (Case 1):

Consider the recurrence relation T(n) = 2T(n/2) + n. In this case, we have a = 2, b = 2, and f(n) = n.

We notice that n is O(n^(log_b a)), where log_2 2 = 1. So, this fits into Case 1 of the Master Theorem.

According to the Master Theorem, when T(n) falls into Case 1, the solution is T(n) = Theta(n^(log_b a)). Therefore, for our example, T(n) = Theta(n).

Example 2 (Case 2):

Let's look at the recurrence relation T(n) = 2T(n/2) + n^2. Here, a = 2, b = 2, and f(n) = n^2.

Comparing n^2 with n^(log_b a) (log_2 2 = 1), we find that f(n) = n^2 is Theta(n^(log_b a)), satisfying Case 2.

According to the Master Theorem, when T(n) falls into Case 2, the solution is T(n) = Theta(n^(log_b a) log n). So, for our example, T(n) = Theta(n log n).

Example 3 (Case 3):

Now, let's consider T(n) = 2T(n/2) + n^3. In this case, a = 2, b = 2, and f(n) = n^3.

When we compare n^3 with n^(log_b a) (log_2 2 = 1), we see that f(n) = n^3 is Omega(n^(log_b a)).

Additionally, we check if a f(n/b) <= kf(n) for some k < 1 and sufficiently large n. Here, 2(n/2)^3 = n^3/2 <= (1/2)n^3, which holds.

So, our recurrence relation falls into Case 3 of the Master Theorem.

According to the Master Theorem, when T(n) is in Case 3, the solution is T(n) = Theta(f(n)). Therefore, for our example, T(n) = Theta(n^3).