Tania Akter
V00810640

8.8

a) It is O(logn)
b) The worst case scenario for non-balanced tree is O(n)

8.9

a) Asymptotic time complexity is O(n^2)
b) Asymptotic space complexity is O(1)

8.10

a) Asymptotic time complexity is O(n). Making the assumption that the CPU is optimized to do the std::swap function in-place, since there will be no extra allocation of memory, the space complexity is O(1);
b) The time complexity is O(n) due to the for-loop. Since the elements are copied into a vector, there are is a space overhead of n, making the space complexity O(n);
c) Looking at both the time complexity and the space complexity, they both have the same time complexity but reverse_array_1 has a better space complexity. Hence, reverse_array_1 is a better function to use.

8.12

Overall speedup for each parts:

A:
1/((1-0.05)+(0.05/10)) = 1.047

B:
1/((1-0.5)+(0.5/1.05)) = 1.024

C :
1/((1-0.1)+(0.1/3)) = 1.071

The overall speedup is the most in case of C, hence C should be optimized.

8.13

a) Here, input is the number of bits in the integer. Hence, the time complexity is O(n). There aren't any extra space being used, so the space complexity is O(1).
b)

Unsigned int hamming_2(unsigned int n)

```
{
        return( arr([n & 0xFFFF] + arr[n >>16] ));
}
```

Here, arr is an array storing the bitcounts of all the unsigned integers, from min to max inclusive. Since the array lookup is constant, the time complexity here is O(1).

This is a significant increase in the time complexity, which is an advantage. However, the space complexity will be unreasonably large due to storing the bit counts in an array lookup table; this is disadvantage.

c) The input parameter is an unsigned int, which gives a large number as an input value. Although the numbers are not going to infinity, I believe the n is sufficiently large enough to consider asymptotic analysis in situation like this.