HW #1:

Part A:

1) Move semantics is a process meant to decrease memory costs by reducing the amount of copying a program does. For small-scale objects such as an integer, it is not very time or cost consuming to just copy and delete after use. However, more complex objects such as vectors take up a lot more space and it's very inefficient to make the same vector twice, especially when only the original needs to be modified. With move semantics, we make an object an rvalue reference & move it around without wasting more memory.

2) When std::move is employed, the string stored at index 5 is converted to an rvalue reference. Because of this, it's actually moved to blurbs1 & removed from blurbs2. Without std::move, the string would be copied & pushed into blurbs1, while the original is still in blurbs2.

3) Overall, it doesn't make a difference. Foo can be assumed to return an r-value, which makes using std::move redundant

Part B:

1) Slowest → Fastest Growth Rate

$\frac{2}{n}$ 37 $\sqrt{N}$ N $N \log\log N$ $N \log N$ $N \log^2 N$ $N \log(N^2)$

($N \log^2 N$ and $N \log(N^2)$: same rate)

$N^{1.5}$ N^2 $N^2 \log N$ N^3 $2^{N/2}$ 2^N

2) (a) True (b) False (c) False (d) False

5) $f(N) = 1$ when N is even & N when N is odd.
$g(N) = 1$ when N is odd & N when N is even

6) (a): 2^{2^N} Starting @ Day 0.

(b) $2^{2^N} \geq D \Rightarrow 2^N \geq \log_2 D \Rightarrow N \geq \log_2(\log_2 D)$

$N = O(\log_2 \log_2 D)$

7) (1) (a) 1 (line 1) + 1 (initializing i) + n+1 (condition) + n (increment)
+ n (line 3) = $3n + 3$

↳ Runtime is $O(N)$

(b)
```
auto sum = 0;
for(int i = 0; i < n; i++){
    sum ++;
}
```

N=1 runtime = 6

N=2 runtime = 9

N=3 runtime = 12

(c) N=1 → 0.00191 ms

N=2 → 0.001345ms

N=3 → 0.001302ms

(2) (a) $O(N^2)$ because inner loop is N & outer loop is N.

(b)
```
int sum = 0;
for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
        sum++;
    }
}
```

N=1 time = 9

N=2 time = 14

N=3 time = 19

(c) N=1 → 0.0012 ms

N=2 → 0.001223 ms

N=3 → 0.00129ms

(3) (a) Inner loop has a complexity of n^2 & the outer loop is n. Therefore, complexity is $O(N^3)$

(b)
```
int sum=0;
for(int i=0; i<n; i++){
  for(int j=0; j<n*n; j++){
    sum++;
  }
}
```

N=1 → 1
N=2 → 22
N=3 → 33

(c) N=1 → 0.001188 ms
N=2 → 0.001281 ms
N=3 → 0.001233 ms

4) (a) Both layers have complexity of n making runtime $O(N^2)$

(b)
```
sum=0
for(i in range n):
  for(j in range i):
    sum++
```

(c) N=1 runtime = 13 → 0.001258 ms
N=2 runtime = 21 → 0.001257 ms
N=3 runtime = 26 → 0.001241 ms

5) (a) J can be as large as N^2; K can be as large as j, which is n^2.
This makes runtime $O(N^5)$

(b)
```
int sum=0;
for(int i=0; i<n; i++){
  for(int j=0; j<i*i; j++){
    for(int k=0; k<j; k++){
      sum++;
    }
  }
}
```

(c) N=1 time = 7 → 0.001258 ms
N=2 time = 14 → 0.001354 ms
N=3 time = 21 → 0.00128 ms

6) (a) $O(N^4)$

(b)
```
int sum = 0;
for (int i = 0; i <= n; i++) {
    for (j = 0; j < i*i; j++) {
        if (j % i == 0) {
            for (int k = 0; k < j; k++) {
                sum++;
            }
        }
    }
}
```

(c) N=1 time = 7 → 0.00124 ms

N=2 time = 21 → 0.001249 ms

N=3 time = 56 → 0.001256 ms

10) (a) $O(N)$ (adding each digit)

(b) $O(N^2)$ (multiplying each digit by each digit

(c) $O(N^2)$ by

11) (a) 5x as long or 2.5 ms

(b) $O(N \log N) \Rightarrow 0.5 \cdot 5 \log 5 = 5.8$ ms

(c) $0.5 \cdot 5^2 = 12.5$ ms

(d) $0.5 \cdot 5^3 = 62.5$ ms

12) (a) $\frac{60}{0.0005} = \frac{N}{100} \rightarrow N = \frac{60}{0.0005} \times 100 = 12{,}000{,}000$

(b) $\frac{60}{0.0005} = \frac{N}{100} \log \frac{N}{100} \quad \therefore N = 912{,}192$

(c) $\frac{60}{0.0005} = \left(\frac{N}{100}\right)^2 \quad \sqrt{\frac{60}{0.0005}} \cdot 100 = N = 34{,}641$

(d) $\frac{60}{0.0005} = \left(\frac{N}{100}\right)^3 \quad \sqrt[3]{\frac{60}{0.0005}} \cdot 100 = N = 4932$

Part C:
The best solution I can think of is adding all of the textbooks from the first vector to the second vector. Order doesnt matter. Once all objects have been added, we can perform a merge sort. This is the fastest sort available to us. However, it takes up more storage than other sorts. The complexity of adding the objects is $O(n)$ & the complexity of the mergesort is $O(n \log n)$. Therefore the complexity is $O(N^2 \log N)$