a. Sort this number using descending order A = {200,500,700,600,350,550,400}.

b.

INSERTION-SORT (A)

```
1   for j = 2 to A.length
2       key = A[j]
3       // Insert A[j] into the sorted sequence A[1. . j – 1]
4       i = j - 1
5       while i >=0 and A[i] < key
6           A[i + 1] = A[i]
7           i= i - 1
8       A[i + 1] = key
```

c. Correctness proof:

Loop Invariant: *At the start of each iteration of the for loop of lines 1–8, the subarray* $A[1\ldots(j-1)]$ *consists of the elements originally in* $A[1\ldots(j-1)]$, *but in sorted order.*

Initialization: We start by showing that the loop invariant holds before the first loop iteration, when j = 2. The subarray $A[1\ldots(j-1)]$, therefore, consists of just the single element $A[1]$, which is in fact the original element in $A[1]$. Moreover, this subarray is sorted (trivially, of course), which shows that the loop invariant holds prior to the first iteration of the loop.

Maintenance: Next, we tackle the second property: showing that each iteration maintains the loop invariant. Informally, the body of the **for** loop works by moving $A[j-1]$, $A[j-2]$, $A[j-3]$, and so on by one position to the right until it finds the proper position for $A[j]$ (lines 4–7), at which point it inserts the value of $A[j]$ (line 8). The subarray $A[1..j]$ then consists of the elements originally in

$A[1..j]$, but in sorted order. Incrementing j for the next iteration of the for loop then preserves the loop invariant. A more formal treatment of the second property would require us to state and show a loop invariant for the while loop of lines 5–7. At this point, however, we prefer not to get bogged down in such formalism, and so we rely on our informal analysis to show that the second property holds for the outer loop.

Termination: Finally, we examine what happens when the loop terminates. The condition causing the **for** loop to terminate is that $j < A.length = n$. Because each loop iteration increases j by 1, we must have $j = n + 1$ at that time. Substituting $n + 1$ for j in the wording of loop invariant, we have that the subarray $A[1..n]$ consists of the elements originally in $A[1..n]$, but in sorted order. Observing that the subarray $A[1..n]$ is the entire array, we conclude that the entire array is sorted. Hence, the algorithm is correct.

We shall use this method of loop invariants to show correctness later in this chapter and in other chapters as well.

d.

e. Incremental design technique used in insertion sort. Insertion sort builds the sorted sequence one element at a time.

f.

Line by line simulation

j = 2

key = A[2] = 500

i = j-1 = 2-1 = 1

1>=0 and A[1] 200 < 500

A[1+1] = A[1]

A[2] = A[1]

i = 1-1 = 0

A[0+1] = 500

A[1] = 500

j = 3

key = A[3] = 700

i = j-1 = 3-1 = 2

2>=0 and A[*2*] 200 < 700

A[2+1] = A[2]

A[3] = A[2]

i = 2-1 = 1

A[1+1] = 700

A[2] = 700

i = j-1 = 2-1 = 1

1>=0 and A[1] 500 < 700

A[1+1] = A[1]

A[2] = A[1]

i = 1-1 = 0

A[0+1] = 700

A[1] = 700

j = 4

key = A[4] = 600

i = j-1 = 4-1 = 3

3>=0 and A[*3*] 200< 600

A[3+1] = A[3]

A[4] = A[3]

i = 3-1 = 2

A[2+1] = 600

A[3] = 600

i = j-1 = 3-1 = 2

2>=0 and A[2] 500 < 600

A[2+1] = A[2]

A[3] = A[2]

i = 2-1 = 1

A[1+1] = 600

A[2] = 600

j = 5

key = A[5] = 350

i = j-1 = 5-1 = 4

4>=0 and A[4] 200< 350

A[4+1] = A[4]

A[5] = A[4]

i = 4-1 = 3

A[3+1] = 350

A[4] = 350

j = 6

key = A[6] = 550

i = j-1 = 6-1 = 5

5>=0 and A[*5*] 200< 550

A[5+1] = A[5]

A[6] = A[5]

i = 5-1 = 4

A[4+1] = 550

A[5] = 550

i = j-1 = 3-1 = 2

2>=0 and A[2] 600 > 550

i = j-1 = 4-1 = 3

3>=0 and A[3] 500 < 550

A[3+1] = A[3]

A[4] = A[3]

i = 3-1 = 2

A[2+1] = 550

A[3] = 550

j = 7

key = A[7] = 400

i = j-1 = 7-1 = 6

6>=0 and A[*6*] 200< 400

A[6+1] = A[6]

A[7] = A[6]

i = 6-1 = 5

A[5+1] = 400

A[6] = 400

i = j-1 = 5-1 = 4

4>=0 and A[4] 500 >*400*

i = j-1 = 6-1 = 5

5>=0 and A[5] 350 < 400

A[5+1] = A[5]

A[6] = A[5]

i = 5-1 = 4

A[4+1] = 400

A[5] = 400

Sorted Array is:

700	600	550	500	400	350	200
1	2	3	4	5	6	7

g.

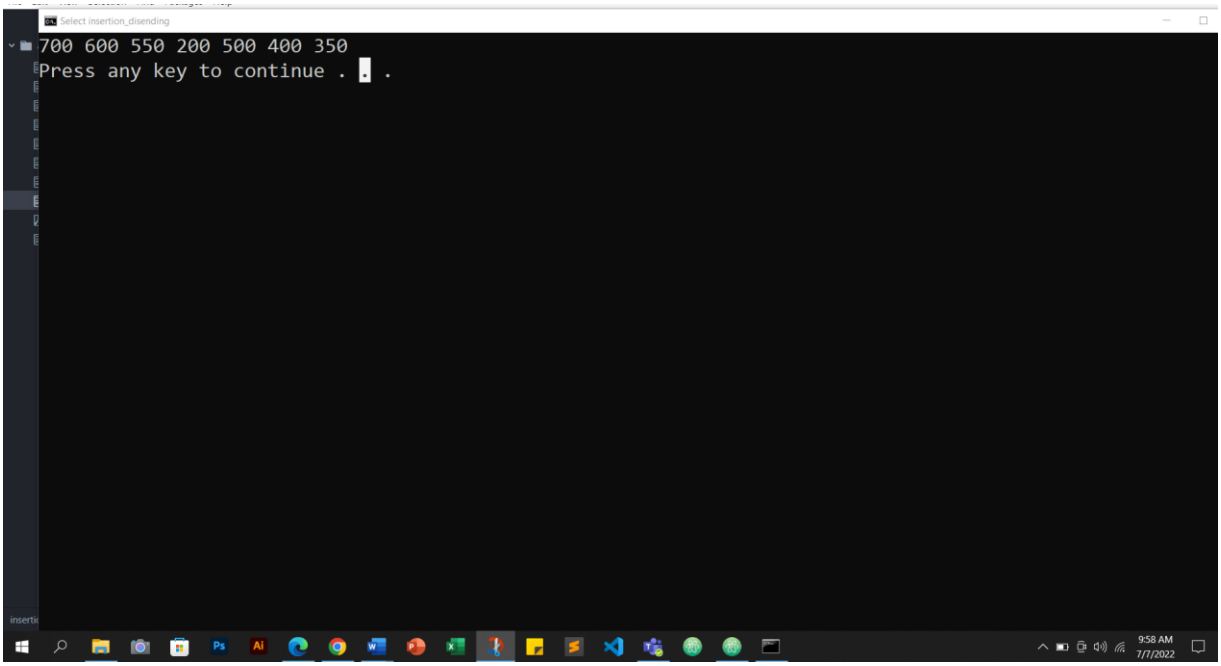
```
#include <iostream>
#define length 7
using namespace std;

void Insertion_Sort(int A[])
{
   int i, j, key;
   for (j = 2; j < length; j++)
   {
      key = A[j];
      i = j - 1;
      while (i >= 0 && A[i] < key)
      {
         A[i + 1] = A[i];
         i = i - 1;
      }
      A[i + 1] = key;
   }
}

void print(int A[])
{
   int i;
   for (i = 0; i < length; i++)
   {
      cout << A[i] << " ";
   }
}

int main()
{
   int A[] = {200,500,700,600,350,550,400};
   Insertion_Sort(A);
   print(A);

   return 0;
}
```


Select insertion_disending
700 600 550 200 500 400 350
Press any key to continue . . .
9:58 AM
7/7/2022