```cpp
#include <iostream>
#include <iomanip>
#include <cmath>
#include <ctime> //Random Number Generator

// Goal to match Average Case with Monte Carlo
using namespace std;

int insertionA10(int A[], int n); // in practise they have to be sorted

int main()
{
  double MonteCarlo;
  double average_calculation;
  int repeation = 10000;
  int bound = 10000;
  int n[] = {100, 500, 1000, 2500, 3000, 3500}; // values for n

  srand(time(NULL)); // Utilizes the computer's internal clock to determine the seed, which constantly changes as time progresses. It's crucial to note that if the seed remains constant, the sequence of numbers will repeation with each execution of the program.
```

```
  cout << setw(5)<< "Size Input" << setw(27) << "Average
Calculated" << setw(22) << "Monte Carlo" << endl;

  cout <<
"============================================================
=======" << endl;

  for (int i = 0; i < sizeof(n); i++) {
    long long int total_steps = 0;
    int size_input = n[i];
    int *arr = new int[size_input];

    // Generate ten thousand random sequences
    for (int j = 0; j < repeation; j++) {
      for (int k = 0; k < size_input; k++) {
        arr[k] = rand() % bound;}
      int insertion = insertionA10(arr, size_input); //
calling insertionA10
      total_steps += insertion; }  // update total_steps

    // Monte Carlo
    MonteCarlo = static_cast<double>(total_steps) /
repeation;
    // The average calculated value Ai(n) = n^2/4 + 3n/4
```

```
    average_calculation = (pow(size_input, 2) / 4) + (3 *
size_input / 4);
    cout << setprecision(1) << fixed << setw(7) <<
size_input << setw(25) << average_calculation << setw(26)
<< MonteCarlo << endl;}
  return 0;
}
```

```
int insertionA10(int A[], int n)
{
  int steps = 0;
  A[0] = -32768; // The smallest attainable integer using a
2-byte
  int i, j, temp;
  for (i = 1; i <= n; i++) {
    j = i;
    while (A[j] < A[j - 1]) { //introducing temporary
variable
      temp = A[j]; // assigning first value to temporary
      A[j] = A[j - 1]; // assigning value of second to
first
      A[j - 1] = temp; // assigning temporary variable to
second
      j--;
      steps++; }// need to add counter in while loop
```

```
    steps++;} // need to add another counter for edge cases becouse it is not count first or last comparison because they might not need comparison
  return steps;
}
```

```
// In summary, the fact that the real average values closely match the calculated averages across various input sizes indicates that the Monte Carlo simulation effectively approximates the practical performance of algorithm A10. This serves as strong validation for the accuracy of the simulation approach.
```