```cpp
/**
 * @file main.cpp
 * @Synopsis Generates arrays containing random values, and records execution time for maximum subarray sum
    with three algorithms
 * @author Tyson Cross / Group D
 * @version 0.3
 * @date 2016-08-17
 */

#include <iostream>
#include <sstream>
#include <fstream>
#include <iomanip>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <chrono>

using namespace std;

//////////////////////////
//      Functions
//////////////////////////

// Brute Force Approach
int brute(const int* a, const int length){                          // Provide pointer to array, length len
    int maximum = a[0];                                             // first max is first element
    for (int i = 0; i < length; i++){                               // run through, iterating by steps (default
        1)
        int max_local = a[i];                                       // current max is current element
        for(int j = i; j < length; ++j){                            // from current element until end of
            elements,
            if(i!=j) max_local += a[j];                             // avoid adding starting element twice, add
                elements so far
            if (max_local > maximum) maximum = max_local;           // compare current max to total maximum.
        }
    }
    if (maximum < 0) maximum = 0;
    return maximum;
}

// Divide and conquer: Recursive algorthm to split, compare left/right/overlap
int divide(const int* a, const int start, const int end){
    if (start > end ) return 0;                                     // index error check
    if (start==end) return max(0,a[start]);                         // if there is one element, return it (or
        the zeroth element)
    int middle = (start + end) / 2;                                 // find midpoint
    /* Find the max on the left */
    int sum = 0;
    int max_left = 0;
    for (int i = middle; i >= start; i--){                          // start... <--[midpoint]
        sum += a[i];
        if (max_left < sum) max_left = sum;
    }
    /* Find the max on the right */
    sum = 0;
    int max_right = 0;
    for (int i = (middle + 1); i <= end; i++){                      // [midpoint.--> .... end
        sum += a[i];
        if (max_right < sum) max_right = sum;
    }
    int max_intersection = max_left + max_right;                    // if both sides are positive, then the
        whole subarray = current max
    /* Recursion to continue to split up */
    int max_A = divide(a, start, middle);                           // continue to sub divide recursively
    int max_B = divide(a, (middle+1), end);                         //
    return max(max(max_intersection, max_A), max_B);                // final return of max value
}

// Elegant linear time algorithm by Jay Kadane (1984)
 int kadane(const int* a, const int len){
    int maximum = 0, max_local = 0;                                 // the maximum, and the running total so far
    for (int i = 0; i < len; i++){                                  // single traversal of the elements
        max_local += a[i];                                          // sum the elements so far from the current
            subarray
        if (max_local < 0) max_local = 0;                           // check if the sum is negative - zeroth
            element
        if (maximum < max_local) maximum = max_local;               // check if the current sub-array or the
            maximum is larger
    }
    return maximum;
}

// Helper Functions

signed int rand_int(){ return (rand() % 101) - 50; }                // range -50 to 50
```

```
void printArray(const int* a, const int length){
    cout << "{ ";
    for (int i = 0; i < length; i++) {
        cout << setw(3) << a[i] << " ";
    }
    cout << " }" << endl;
}

/** ========================================================================================== **/

////////////////////////////
//      Main
////////////////////////////

int main(int argc, char* argv[]){
    /* Profiling */ clock_t startTime = clock();               // overall system time elapsed, not very
        precise

    int sum_brute = 0, sum_divide = 0 , sum_kadane = 0;
    int max_num;
    int start = 1;                                             // Needs to be 1 (erroneous for Brute and
        Divide at 0)
    int steps = 1;                                             // Default number of array sizes to skip
        when incrementing

    /* Check for input arguement for maximum size of array input length */
    if (argc < 2) {
        cerr << "Error: Usage is " << argv[0] << " MAX_ARRAY_SIZE <ITERATION_SKIP_SIZE>" << endl;
        return 1;
    }

    /* convert input argument to integer */
    istringstream ss(argv[1]);                                 // stream the second optional integer
    if (!(ss >> max_num)){                                     // check
        cerr << "Invalid number of iterations" << argv[1] << '\n';
        return 1;
    }

    /* optional stepping of iterations */
    if (argc > 2) {                                            // optional arguement
        istringstream ss_steps(argv[2]);
        if ((!(ss_steps >> steps))||(steps < 1)||(steps > max_num)){
            cerr << "Invalid number of steps " << argv[2] << '\n';
            return 1;
        }
    }

    /* Output setup */
    string fileOutName = "output.txt";
    ofstream outputFile(fileOutName, ios::out | ios::trunc);
    if (!outputFile.is_open()) { cerr << "Unable to open file:" << fileOutName << endl; return -1;}

    /* Initialize random seed */
    srand (static_cast<unsigned int>(time(NULL)));

    /* Output titles */
    int width = 20;
    outputFile << setw(width) << left << "#Input Size";
    outputFile << setw(width) << left << "Brute Force";
    outputFile << setw(width) << left << "Divide and Conquer";
    outputFile << setw(width) << left << "Kadane's Algorithm";
    outputFile << endl;

    for (int i = start; i <= max_num; i+=steps){

        int length = i;

        /* Generate the increasing lengths of arrays with random numbers */
        signed int* num_array = nullptr;
        for (int i = start; i <= max_num; i+=steps){
            num_array = new signed int[i];
            for (int j = 0; j < i ; j++){
                num_array[j]=rand_int();
            }
        }

        auto time_brute_start = chrono::high_resolution_clock::now();
        sum_brute = brute(num_array,length);
        auto time_brute_end = chrono::high_resolution_clock::now();
        auto time_brute = chrono::duration_cast<chrono::microseconds>(time_brute_end - time_brute_start); //
            INT version

        auto time_divide_start = chrono::high_resolution_clock::now();
        sum_divide = divide(num_array,0,length-start);
        auto time_divide_end = chrono::high_resolution_clock::now();
```

```
        auto time_divide = chrono::duration_cast<chrono::microseconds>(time_divide_end -
            time_divide_start); // INT version

        auto time_kadane_start = chrono::high_resolution_clock::now();
        sum_kadane = kadane(num_array,length);
        auto time_kadane_end = chrono::high_resolution_clock::now();
        auto time_kadane = chrono::duration_cast<chrono::microseconds>(time_kadane_end -
            time_kadane_start); // INT version

        /* Output Time */
        outputFile << setw(width) << left << length;
        outputFile << setw(width) << left << std::setprecision(7) << fixed << time_brute.count();
        outputFile << setw(width) << left << std::setprecision(7) << fixed << time_divide.count();
        outputFile << setw(width) << left << std::setprecision(7) << fixed << time_kadane.count();
        outputFile << endl;

        //printArray(num_array,length);

        // Confirm Algorithms are correct:
        if ((sum_brute!=sum_divide)||(sum_brute!=sum_kadane)||(sum_divide!=sum_kadane)){
            cerr << "Warning : Algorithms have different maximum subarray sums for length " << i << endl;
            printArray(num_array,length);
            cout << "Brute Force gives " << sum_brute << endl;
            cout << "Divide & Conquer gives " << sum_divide << endl;
            cout << "Kadane's Algorithm gives " << sum_kadane << endl;
            outputFile.close();
            return 1;
        }
        delete num_array;
    }

    outputFile.close();

    /* Profiling */
    cout << "Executable Runtime: " << double( clock() - startTime) / (double) CLOCKS_PER_SEC << " seconds."
        << endl;
    cout << "Processing complete.";

    return 0;
}
```