```cpp
#include <iostream>
#include <vector>
#include <limits>
#include <random>
#include <chrono>
#include <algorithm>

using namespace std;

bool get_line(const string& prompt, string& userinput){
    cout << prompt;
    getline(cin, userinput);
    return !userinput.empty();
}

template<typename T>
void partial_sort(vector<T>& arr, int lower_bound, int upper_bound){
    for(int i = lower_bound;i <= upper_bound;i++){
        for(int j = i;j > 0 && arr[j] < arr[j - 1];j--){
            swap(arr[j], arr[j - 1]);
        }
    }
}

template<typename T>
int median_of_three(vector<T>& arr, int lo, int hi){
    int mid = (lo + hi) / 2;
    if(arr[lo] > arr[hi])
        swap(arr[lo], arr[hi]);
    if(arr[lo] > arr[mid])
        swap(arr[lo], arr[mid]);
    if(arr[mid] > arr[hi])
        swap(arr[mid], arr[hi]);
    return mid;
}

template<typename T>
int partition(vector<T> &arr, int lo, int hi){
    int i = lo - 1, j = hi;
    T pivot_value = arr[hi];
    while(true){
        while(arr[++i] < pivot_value) if(i == hi) break;
        while(arr[--j] > pivot_value) if(j == lo) break;
        if(i >= j) break;
        swap(arr[i], arr[j]);
    }
    swap(arr[i], arr[hi]);
    return i;
}
```

```cpp

template<typename T>
void quick_sort(vector<T>& arr, int lo, int hi){
    const int CUTOFF_TO_INSERTION_SORT = 3;
    if(hi <= lo + CUTOFF_TO_INSERTION_SORT){
        partial_sort(arr, lo, hi);
        return;
    }
    int pivot_idx = median_of_three(arr, lo, hi);
    swap(arr[pivot_idx], arr[hi]);
    int new_pivot_idx = partition(arr, lo, hi);
    quick_sort(arr, lo, new_pivot_idx - 1);
    quick_sort(arr, new_pivot_idx + 1, hi);
}

template<typename T>
void quick_sort(vector<T>& arr){
    quick_sort(arr, 0, arr.size() - 1);
}

bool AreSame(double a, double b)
{
    return fabs(a - b) < numeric_limits<double>::epsilon()
;
}

void display_arr(const vector<double>& arr){
    for(double e : arr) cout << e << " ";
    cout << endl;
}

template<typename T>
double find_max_crossing_subarray(const vector<T>& arr,
int low, int mid, int high){
    vector<T> left_sum(mid - low + 1);
    double sum = 0;
    int counter = left_sum.size() - 1;
    for(int i = mid;i >= low;i--){
        sum += arr[i];
        left_sum[counter--] += sum;
    }

    vector<T> right_sum(high - mid);
    sum = 0;
    counter = 0;
    for(int i = mid + 1;i <= high;i++){
        sum += arr[i];
        right_sum[counter++] += sum;
    }
    quick_sort(left_sum);
    quick_sort(right_sum);
    reverse(right_sum.begin(), right_sum.end());
    int i = 0;
    int j = 0;
    double s_min = numeric_limits<double>::max();
    while(i < left_sum.size() && j < right_sum.size()){
        double s = left_sum[i] + right_sum[j];
        if(s <= 0) i++;
        else if(s < s_min) s_min = s, j++;
        else j++;
    }
    if(AreSame(s_min, numeric_limits<double>::max()))
        return numeric_limits<double>::lowest();
    return s_min;
}

template<typename T>
double find_MPSS(const vector<T> &arr, int lo, int hi){
    if(lo == hi)
        return max(arr[lo], T());
    int mid = (lo + hi) / 2;
    double mss_l = find_MPSS(arr, lo, mid);
    double mss_r = find_MPSS(arr, mid + 1, hi);
    double mss_m = find_max_crossing_subarray(arr, lo,
mid, hi);
    if(mss_l > 0 && mss_r > 0 && mss_m > 0)
        return min(min(mss_l, mss_r), mss_m);
    return max(mss_l, max(mss_r, mss_m));
}

template<typename T>
double divide_and_conquer_approach(const vector<T> &arr){
    return find_MPSS(arr, 0, arr.size() - 1);
}

int main() {
    string userinput;
    unsigned seed = chrono::steady_clock::now().
time_since_epoch().count();
    mt19937 gen(seed);
    const int LOWER_BOUND = -20;
    const int UPPER_BOUND = 20;
    uniform_real_distribution<double>
uniform_real_distribution(LOWER_BOUND, UPPER_BOUND);
    while(get_line("Enter a positive integer: ",
userinput)){
        int n = stoi(userinput);
        vector<double > arr;
        for(int i = 0;i < n;i++)
            arr.push_back(uniform_real_distribution(gen))
;
        display_arr(arr);
        cout <<"MPSS: "<< divide_and_conquer_approach(arr) << endl;
    }
}
```