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

using namespace std;

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

void display_array(const vector<int>& A){
    for(int e : A) cout << e << " ";
    cout << endl;
}

struct my_pair{
    double x;
    int y;
    bool operator<(const my_pair& rhs) const
    {
        return x < rhs.x;
    }

    bool operator>(const my_pair& rhs) const {
        return rhs < *this;
    }
};


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

```cpp


template<typename T>
int quick_select_idx(vector<T> &arr, int k){
    unsigned int seed = chrono::steady_clock::now().time_since_epoch().count();
    mt19937 gen(seed);
    int lo = 0, hi = arr.size() - 1;
    while(lo < hi){
        int pivot_idx = uniform_int_distribution<int>{lo, hi}(gen);
        int new_pivot_idx = partition(arr, lo, hi, pivot_idx);
        if(new_pivot_idx == k - 1) return new_pivot_idx;
        else if(new_pivot_idx < k - 1) lo = new_pivot_idx + 1;
        else hi = new_pivot_idx - 1;
    }
    return lo;
}

double median_selection(vector<int>& arr){
    bool odd = arr.size() % 2 == 1;
    double median;
    if(odd){
        int median_idx = quick_select_idx(arr, arr.size() / 2 + 1);
        median = arr[median_idx];
    } else {
        int median_idx1 = quick_select_idx(arr, arr.size() / 2);
        int median_idx2 = quick_select_idx(arr, arr.size() / 2 + 1);
        median = (arr[median_idx1] + arr[median_idx2]) / 2.0;
    }
    return median;
}

// O(n)
vector<int> k_closest_to_median(vector<int>& arr, int k){
    double median = median_selection(arr); // O(n)
    vector<my_pair> diff;
    // O(n)
    for(int e : arr){
        diff.push_back({abs(e - median), e});
    }
    // O(n)
    int kth_idx = quick_select_idx(diff, k);
    //O(n)
    vector<int> result;
    for(int i = kth_idx; i >= 0;i--){
        result.push_back(diff[i].y);
    }
    return result;
}


int main() {
    unsigned int seed = chrono::steady_clock::now().time_since_epoch().count();
    mt19937 gen(seed);
    string userinput;
    while(get_line("Enter positive integer n: ", userinput)){
        int n = stoi(userinput);
        uniform_int_distribution<int> uniform_int_distribution(-100, 100);

        vector<int> A;
        for(int i = 0;i < n;i++)
            A.push_back(uniform_int_distribution(gen));

        display_array(A);

        string second_input;
        get_line("Enter a number between 1 to n: ", second_input);
        int kth = stoi(second_input);
        auto result = k_closest_to_median(A, kth);
        display_array(result);
    }
}
```