```cpp
#include <iostream>
#include <vector>
#include <chrono>
#include <functional>
#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;
```

```
48 }
49
50
51 template<typename T>
52 int quick_select_idx(vector<T> &arr, int k){
53     unsigned int seed = chrono::steady_clock::now().
   time_since_epoch().count();
54     mt19937 gen(seed);
55     int lo = 0, hi = arr.size() - 1;
56     while(lo < hi){
57         int pivot_idx = uniform_int_distribution<int>{lo,
   hi}(gen);
58         int new_pivot_idx = partition(arr, lo, hi,
   pivot_idx);
59         if(new_pivot_idx == k - 1) return new_pivot_idx;
60         else if(new_pivot_idx < k - 1) lo = new_pivot_idx
   + 1;
61         else hi = new_pivot_idx - 1;
62     }
63     return lo;
64 }
65
66 double median_selection(vector<int>& arr){
67     bool odd = arr.size() % 2 == 1;
68     double median;
69     if(odd){
70         int median_idx = quick_select_idx(arr, arr.size()
   / 2 + 1);
71         median = arr[median_idx];
72     } else {
73         int median_idx1 = quick_select_idx(arr, arr.size()
    / 2);
74         median = (double) (arr[median_idx1] + arr[
   median_idx1 + 1]) / 2;
75     }
76     return median;
77 }
78
79 // O(n)
80 vector<int> k_closest_to_median(vector<int>& arr, int k){
81     double median = median_selection(arr); // O(n)
82     vector<my_pair> diff;
83     // O(n)
84     for(int e : arr){
85         diff.push_back({abs(e - median), e});
86     }
87     // O(n)
88     int kth_idx = quick_select_idx(diff, k);
89     //O(n)
90     vector<int> result;
```

```
 91     for(int i = kth_idx; i >= 0;i--){
 92         result.push_back(diff[i].y);
 93     }
 94     return result;
 95 }
 96 
 97 
 98 int main() {
 99     unsigned int seed = chrono::steady_clock::now().
    time_since_epoch().count();
100     mt19937 gen(seed);
101     string userinput;
102     while(get_line("Enter positive integer n: ",
    userinput)){
103         int n = stoi(userinput);
104         uniform_int_distribution<int>
    uniform_int_distribution(-100, 100);
105 
106         vector<int> A;
107         for(int i = 0;i < n;i++)
108             A.push_back(uniform_int_distribution(gen));
109 
110         display_array(A);
111 
112         string second_input;
113         get_line("Enter a number between 1 to n: ",
    second_input);
114         int kth = stoi(second_input);
115         auto result = k_closest_to_median(A, kth);
116         display_array(result);
117     }
118 }
```