```cpp
1  #include <iostream>
2  #include <vector>
3  #include <chrono>
4  #include <random>
5
6  using namespace std;
7
8  bool get_line(const string& prompt, string& userinput){
9      cout << prompt;
10     getline(cin, userinput);
11     return !userinput.empty();
12 }
13
14 void display_array(const vector<int>& A){
15     for(int e : A) cout << e << " ";
16     cout << endl;
17 }
18
19 struct my_pair{
20     double x;
21     int y;
22     bool operator<(const my_pair& rhs) const
23     {
24         return x < rhs.x;
25     }
26
27     bool operator>(const my_pair& rhs) const {
28         return rhs < *this;
29     }
30 };
31
32
33 template<typename T>
34 int partition(vector<T> &arr, int lo, int hi, int
   pivot_idx){
35     T pivot_value = arr[pivot_idx];
36     int left = lo - 1;
37     int right = hi;
38     swap(arr[pivot_idx], arr[right]);
39     while(left < right){
40         while(arr[++left] < pivot_value) if(left == right)
   break;
41         while(arr[--right] > pivot_value) if(left == right
   ) break;
42         if(left >= right) break;
43         swap(arr[left], arr[right]);
44     }
45     swap(arr[left],arr[hi]);
46     return left;
47 }
```

```cpp
48
49
50  template<typename T>
51  int quick_select_idx(vector<T> &arr, int k){
52      unsigned int seed = chrono::steady_clock::now().
    time_since_epoch().count();
53      mt19937 gen(seed);
54      int lo = 0, hi = arr.size() - 1;
55      while(lo < hi){
56          int pivot_idx = uniform_int_distribution<int>{lo,
    hi}(gen);
57          int new_pivot_idx = partition(arr, lo, hi,
    pivot_idx);
58          if(new_pivot_idx == k - 1) return new_pivot_idx;
59          else if(new_pivot_idx < k - 1) lo = new_pivot_idx
    + 1;
60          else hi = new_pivot_idx - 1;
61      }
62      return lo;
63  }
64
65  double median_selection(vector<int>& arr){
66      bool odd = arr.size() % 2 == 1;
67      double median;
68      if(odd){
69          int median_idx = quick_select_idx(arr, arr.size()
    / 2 + 1);
70          median = arr[median_idx];
71      } else {
72          int median_idx1 = quick_select_idx(arr, arr.size()
    / 2);
73          int median_idx2 = quick_select_idx(arr, arr.size()
    / 2 + 1);
74          median = (arr[median_idx1] + arr[median_idx2]) / 2
    .0;
75      }
76      return median;
77  }
78
79  // O(n)
80  void 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)
```

```cpp
90          for(int i = kth_idx; i >= 0;i--){
91              cout << diff[i].y << " ";
92          }
93          cout << endl;
94  }


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