

Timus 1306

Sequence Median

Prof. Edson Alves – UnB/FGA

Given a sequence of N nonnegative integers. Let's define the median of such sequence. If N is odd the median is the element with stands in the middle of the sequence after it is sorted. One may notice that in this case the median has position $(N + 1)/2$ in sorted sequence if sequence elements are numbered starting with 1. If N is even then the median is the semi-sum of the two “middle” elements of sorted sequence. I.e. semi-sum of the elements in positions $N/2$ and $(N/2) + 1$ of sorted sequence. But original sequence might be unsorted.

Your task is to write program to find the median of given sequence.

Input

The first line of input contains the only integer number N – the length of the sequence. Sequence itself follows in subsequent lines, one number in a line. The length of the sequence lies in the range from 1 to 250000. Each element of the sequence is a positive integer not greater than $2^{31} - 1$ inclusive.

Output

You should print the value of the median with exactly one digit after decimal point.

Exemplo de entradas e saídas

Sample Input

4

3

6

4

5

Sample Output

4.5

Soluções $O(N \log N)$ e MLE

- Este problema é conceitualmente simples, mas traz uma dificuldade adicional: o limite de memória é curto para o tamanho máximo da entrada (apenas 1 MB)
- A solução mais simples seria ordenar os elementos da sequência em um vetor e computar a média a partir dos índices indicados no problema, mas é necessário mais de um 1 MB para armazenar todos os elementos
- Assim, soluções baseadas em `sort()` e `nth_element()` levam ao MLE
- Uma alternativa seria usar uma `priority_queue()` e armazenar apenas $N/2 + 1$ elementos, descartando os demais sempre que o tamanho da *heap* exceder este limite
- Esta solução ainda leva ao MLE, por conta do vetor subjacente
- A solução portanto, é utilizar esta abordagem, porém utilizando uma implementação customizada da *max heap*, onde os elementos são armazenados em um vetor estático

Solução AC com complexidade $O(N \log N)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX { 125002 };
6
7 class Heap {
8 private:
9     size_t N;
10    unsigned int xs[MAX];
11
12    size_t parent(int i) { return i/2; }
13    size_t left(int i) { return 2*i; }
14    size_t right(int i) { return 2*i + 1; }
15
16 public:
17     Heap() : N(0) {}
18
19     size_t size() const { return N; }
```

Solução AC com complexidade $O(N \log N)$

```
21 void insert(int x)
22 {
23     ++N;
24
25     xs[N] = x;
26
27     int i = N, p = parent(i);
28
29     while (p and xs[p] < xs[i]) {
30         std::swap(xs[p], xs[i]);
31         i = p;
32         p = parent(i);
33     }
34 }
35
36 unsigned int extract_max()
37 {
38     auto x = xs[1];
39     std::swap(xs[1], xs[N]);
40     --N;
```

Solução AC com complexidade $O(N \log N)$

```
42     int i = 1, n = left(i) > N ? 0 : left(i);
43
44     while (n) {
45         auto r = right(i) > N ? 0 : right(i);
46
47         if (r and xs[r] > xs[n])
48             n = r;
49
50         if (xs[i] < xs[n])
51         {
52             std::swap(xs[i], xs[n]);
53             i = n;
54             n = left(i) > N ? 0 : left(i);
55         } else
56             n = 0;
57     }
58
59     return x;
60 }
61 };
```


Solução AC com complexidade $O(N \log N)$

```
63 int main()
64 {
65     unsigned int N;
66     scanf("%u", &N);
67
68     Heap pq;
69
70     for (size_t i = 0; i < N; ++i)
71     {
72         unsigned int x;
73         scanf("%u", &x);
74
75         pq.insert(x);
76
77         if (pq.size() > N/2 + 1)
78             pq.extract_max();
79     }
```

Solução AC com complexidade $O(N \log N)$

```
81  double ans = pq.extract_max();
82
83  if (N % 2 == 0)
84  {
85      ans += pq.extract_max();
86      ans /= 2.0;
87  }
88
89  printf("%.1f\n", ans);
90
91  return 0;
92 }
```