Started on	Wednesday, 13 March 2024, 4:44 PM
State	Finished
Completed on	Wednesday, 13 March 2024, 5:24 PM
Time taken	40 mins 25 secs
Marks	20.00/20.00
Grade	**10.00** out of 10.00 (**100**%)

This question is designed to help you get a better understanding of *basic heap* operations.

There are **3** types of query:

- "**1** v" - Add an element v to the heap.
- "**2** v" - Delete the element v from the heap.
- "**3**" - Print the minimum of all the elements in the heap.

NOTE: It is guaranteed that the element to be deleted will be there in the heap. Also, at any instant, only distinct elements will be in the heap.

Input Format

The first line contains the number of queries, Q.
Each of the next Q lines contains one of the **3** types of query.

Constraints

$1 \le Q \le 10^5$
$-10^9 \le v \le 10^9$

Output Format

For each query of type **3**, print the minimum value on a single line.

Sample Input

```
STDIN       Function
-----       --------
5           Q = 5
1 4         insert 4
1 9         insert 9
3           print minimum
2 4         delete 4
3           print minimum
```

Sample Output

```
4
9
```

Explanation

After the first **2** queries, the heap contains $\{4, 9\}$. Printing the minimum gives **4** as the output. Then, the 4th query deletes **4** from the heap, and the 5th query gives **9** as the output.

For example:

Input	Result
5 1 4 1 9 3 2 4 3	4 9
10 1 10 1 4 1 3 3 2 4 1 5 2 3 3 1 0 3	3 5 0

Answer: (penalty regime: 0 %)

Reset answer

```
1  #include <cmath>
2  #include <cstdio>
3  #include <vector>
4  #include <iostream>
5  #include <algorithm>
```

```cpp
 6  using namespace std;
 7
 8  int main()
 9  {
10      long long int n;
11      long long int c;
12      vector<long int> v;
13
14      // Read the number of queries
15      cin >> n;
16
17      for (int i = 0; i < n; i++)
18      {
19          cin >> c;
20          if (c == 1)
21          {
22              // Add an element
23              cin >> c;
24              v.push_back(c);
25              push_heap(v.begin(), v.end(), greater<long long int>());
26          }
27          else if (c == 3)
28          {
29              // Print the minimum element
30              printf("%ld\n", v.front());
31          }
32          else if (c == 2)
33          {
34              // Delete an element
35              cin >> c;
36              vector<long int>::iterator it;
37              for (it = v.begin(); it != v.end(); it++)
38              {
39                  if (*it == c)
40                  {
41                      v.erase(it);
42                      break;
43                  }
44              }
45              make_heap(it, v.end(), greater<long long int>());
46          }
47      }
48
49      return 0;
50  }
51
```

	Input	Expected	Got	
✔	5 1 4 1 9 3 2 4 3	4 9	4 9	✔
✔	10 1 10 1 4 1 3 3 2 4 1 5 2 3 3 1 0 3	3 5 0	3 5 0	✔

Passed all tests! ✔

Jesse loves cookies and wants the sweetness of some cookies to be greater than value k. To do this, two cookies with the least sweetness are repeatedly mixed. This creates a special combined cookie with:

sweetness $= (1\times$ *Least sweet cookie* $+ 2\times$ *2nd least sweet cookie*).

This occurs until all the cookies have a sweetness $\geq k$.

Given the sweetness of a number of cookies, determine the minimum number of operations required. If it is not possible, return -1.

Example
$k = 9$
$A = [2, 7, 3, 6, 4, 6]$

The smallest values are $2, 3$.
Remove them then return $2 + 2 \times 3 = 8$ to the array. Now $A = [8, 7, 6, 4, 6]$.
Remove $4, 6$ and return $4 + 6 \times 2 = 16$ to the array. Now $A = [16, 8, 7, 6]$.
Remove $6, 7$, return $6 + 2 \times 7 = 20$ and $A = [20, 16, 8, 7]$.
Finally, remove $8, 7$ and return $7 + 2 \times 8 = 23$ to A. Now $A = [23, 20, 16]$.
All values are $\geq k = 9$ so the process stops after 4 iterations. Return 4.

Function Description
Complete the *cookies* function in the editor below.

cookies has the following parameters:

- *int k:* the threshold value
- *int A[n]:* an array of sweetness values

Returns

- *int:* the number of iterations required or -1

Input Format

The first line has two space-separated integers, n and k, the size of $A[]$ and the minimum required sweetness respectively.

The next line contains n space-separated integers, $A[i]$.

Constraints

$1 \leq n \leq 10^6$
$0 \leq k \leq 10^9$
$0 \leq A[i] \leq 10^6$

Sample Input

```
STDIN            Function
-----            --------
6 7              A[] size n = 6, k = 7
1 2 3 9 10 12    A = [1, 2, 3, 9, 10, 12]
```

Sample Output

```
2
```

Explanation

Combine the first two cookies to create a cookie with *sweetness* $= 1 \times 1 + 2 \times 2 = 5$
After this operation, the cookies are $3, 5, 9, 10, 12$.
Then, combine the cookies with sweetness 3 and sweetness 5, to create a cookie with resulting *sweetness* $= 1 \times 3 + 2 \times 5 = 13$
Now, the cookies are $9, 10, 12, 13$.
All the cookies have a sweetness ≥ 7.

Thus, 2 operations are required to increase the sweetness.

For example:

Input	Result
6 7 1 2 3 9 10 12	2
8 10 2 6 8 10 6 6 7 6	4

Answer: (penalty regime: 0 %)

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>

#include <queue>

using namespace std;

int main()
{
    /* Enter your code here. Read input from STDIN. Print output to STDOUT */

    int n, k, operations = 0;
    ;
    cin >> n;
    cin >> k;

    priority_queue<int> p;

    for (int a = 0; a < n; a++)
    {
        int cookie;
        cin >> cookie;
        p.push(cookie * -1);
    }

    while (p.top() > k * -1 && p.size() > 1)
    {
        int cookie1, cookie2, newCookie;

        cookie1 = p.top();
        p.pop();

        cookie2 = p.top();
        p.pop();

        newCookie = cookie1 + 2 * cookie2;
        p.push(newCookie);

        operations++;
    }

    if (p.top() > k * -1)
        cout << "-1";
    else
        cout << operations;

    return 0;
}
```

	Input	Expected	Got	
✔	6 7 1 2 3 9 10 12	2	2	✔
✔	8 10 2 6 8 10 6 6 7 6	4	4	✔

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.