Started on	Wednesday, 13 March 2024, 7:18 PM
State	Finished
Completed on	Wednesday, 13 March 2024, 8:09 PM
Time taken	50 mins 46 secs
Marks	20.00/20.00
Grade	**10.00** out of 10.00 (**100**%)

Question 1

Correct

Mark 10.00 out of 10.00

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

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

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 4^{th} query deletes 4 from the heap, and the 5^{th} 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

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

// Function to decrease the key of an element in the heap
void heapDecreaseKey(vector<int>& arr, int index, int key) {
    arr[index] = key;
    while (arr[index / 2] > key && index > 0) {
        swap(arr[index / 2], arr[index]);
        index = index / 2;
    }
}

// Function to insert an element into the heap
void insert(vector<int>& arr, int key) {
    arr.push_back(numeric_limits<int>::infinity()); // Push positive infinity
    heapDecreaseKey(arr, arr.size() - 1, key);   // Decrease the key of the newly inserted element
}

// Function to perform min-heapify operation
void minHeapify(vector<int>& arr, int index) {
    int len = arr.size();
    int lowest = index;
    int l = 2 * index;
    int r = 2 * index + 1;
    // Check if the left child is smaller than the current node
    if (l < len && arr[index] > arr[l]) {
        lowest = l;
    }
    // Check if the right child is smaller than the current node and the left child
    if (r < len && arr[lowest] > arr[r]) {
        lowest = r;
    }
    // If the smallest element is not the current node, swap with the smallest child and continue min-heapi
    if (index != lowest) {
        swap(arr[index], arr[lowest]);
        minHeapify(arr, lowest);
    }
}

// Function to build a min-heap from an array
void buildMinHeap(vector<int>& arr) {
    int len = arr.size();
    // Start from the last non-leaf node and perform min-heapify on each node
    for (int i = len / 2; i >= 0; i--) {
        minHeapify(arr, i);
    }
}
```

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

	Input	Expected	Got	
✔	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! ✔

▸ Show/hide question author's solution (Cpp)

Correct

Marks for this submission: 10.00/10.00.

Question 2

Correct

Mark 10.00 out of 10.00

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 %)

Reset answer

```cpp
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'cookies' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER k
 *  2. INTEGER_ARRAY A
 */

int cookies(int k, vector<int> A) {
    // Create a min-heap priority queue from the input vector A
    priority_queue<int, vector<int>, greater<int>> min_heap(A.begin(), A.end());

    // Initialize the number of operations performed to 0
    int ops = 0;

    // Continue loop until top cookie sweetness >= k and at least 2 cookies in heap
    while (min_heap.top() < k && min_heap.size() >= 2) {
        // Pop sweetness of least and second least sweet cookies
        int least = min_heap.top();
        min_heap.pop();
        int second_least = min_heap.top();
        min_heap.pop();

        // Calculate sweetness of combined cookie and push it into heap
        min_heap.push(least + 2 * second_least);

        // Increment operation count
        ops++;
    }

    // If top cookie sweetness is still < k or achieving k is impossible, return -1
    if (min_heap.top() < k){
       return -1;
    }
    // Otherwise, return number of operations performed
    else{
        return ops;
    }
}


int main()
{
```

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

	Input	Expected	Got	
✔	8 10 2 6 8 10 6 6 7 6	4	4	✔

Passed all tests! ✔

▸ **Show/hide question author's solution (Cpp)**

Correct

Marks for this submission: 10.00/10.00.