C. Mr. Kitayuta, the Treasure Hunter

time limit per test 1 second
memory limit per test 256 megabytes
input standard input
output standard output

The Shuseki Islands are an archipelago of 30001 small islands in the Yutampo Sea. The islands are evenly spaced along a line, numbered from 0 to 30000 from the west to the east. These islands are known to contain many treasures. There are n gems in the Shuseki Islands in total, and the i-th gem is located on island p_i.

Mr. Kitayuta has just arrived at island 0. With his great jumping ability, he will repeatedly perform jumps between islands to the east according to the following process:

- First, he will jump from island 0 to island d.
- After that, he will continue jumping according to the following rule. Let l be the length of the previous jump, that is, if his previous jump was from island $prev$ to island cur, let $l = cur - prev$. He will perform a jump of length $l - 1$, l or $l + 1$ to the east. That is, he will jump to island $(cur + l - 1)$, $(cur + l)$ or $(cur + l + 1)$ (if they exist). The length of a jump must be positive, that is, he cannot perform a jump of length 0 when $l = 1$. If there is no valid destination, he will stop jumping.

Mr. Kitayuta will collect the gems on the islands visited during the process. Find the maximum number of gems that he can collect.

Input

The first line of the input contains two space-separated integers n and d ($1 \le n, d \le 30000$), denoting the number of the gems in the Shuseki Islands and the length of the Mr. Kitayuta's first jump, respectively.

The next n lines describe the location of the gems. The i-th of them ($1 \le i \le n$) contains a integer p_i ($d \le p_1 \le p_2 \le \ldots \le p_n \le 30000$), denoting the number of the island that contains the i-th gem.

Output

Print the maximum number of gems that Mr. Kitayuta can collect.

Examples

input Copy

```
4 10
10
21
27
27
```

output Copy

```
3
```

input Copy

```
8 8
9
19
28
36
45
55
66
78
```

output Copy

```
6
```

input Copy

```
13 7
8
8
9
16
17
17
18
21
23
24
24
26
30
```

output Copy

```
4
```

Note

In the first sample, the optimal route is 0 $\rightarrow$ 10 (+1 gem) $\rightarrow$ 19 $\rightarrow$ 27 (+2 gems) $\rightarrow$ ...

In the second sample, the optimal route is 0 $\rightarrow$ 8 $\rightarrow$ 15 $\rightarrow$ 21 $\rightarrow$ 28 (+1 gem) $\rightarrow$ 36 (+1 gem) $\rightarrow$ 45 (+1 gem) $\rightarrow$ 55 (+1 gem) $\rightarrow$ 66 (+1 gem) $\rightarrow$ 78 (+1 gem) $\rightarrow$ ...

In the third sample, the optimal route is 0 $\rightarrow$ 7 $\rightarrow$ 13 $\rightarrow$ 18 (+1 gem) $\rightarrow$ 24 (+2 gems) $\rightarrow$ 30 (+1 gem) $\rightarrow$ ...

Let m be the number of the islands (that is, 30001). First, let us describe a solution with time and memory complexity of $O(m^2)$.

We will apply Dynamic Programming. let $dp[i][j]$ be the number of the gems that Mr. Kitayuta can collect after he jumps to island i, when the length of his previous jump is j (let us assume that he have not collect the gems on island i). Then, you can calculate the values of the table dp by the following:

- $dp[i][j] = 0$, if $i \geq m$
  (actually these islands do not exist, but we can suppose that they exist and when Mr. Kitayuta jumps to these islands, he stops jumping)
- $dp[i][j] =$ (the number of the gems on island i) $+ max(dp[i+j][j], dp[i+j+1][j+1])$, if $i < m, j = 1$
  (he cannot perform a jump of length 0)
- $dp[i][j] =$ (the number of the gems on island i) $+ max(dp[i+j-1][j-1], dp[i+j][j], dp[i+j+1][j+1])$, if $i < m, j \geq 2$

This solution is unfeasible in terms of both time and memory. However, the following observation makes it an Accepted solution: there are only 491 values of j that we have to consider, which are $d-245, d-244, d-243, ..., d+244$ and $d+245$.

Why? First, let us find the upper bound of j. Suppose Mr. Kitayuta always performs the "$l+1$" jump (l: the length of the previous jump). Then, he will reach the end of the islands before he performs a jump of length $d+246$, because
$d+(d+1)+(d+2)+...+(d+245) \geq 1+2+...+245 = 245 \cdot (245+1)/2 = 30135 > 30000$. Thus, he will never be able to perform a jump of length $d+246$ or longer.

Next, let us consider the lower bound of j in a similar way. If $d \leq 246$, then obviously he will not be able to perform a jump of length $d-246$ or shorter, because the length of a jump must be positive. Suppose Mr. Kitayuta always performs the "$l-1$" jump, where $d \geq 247$. Then, again he will reach the end of the islands before he performs a jump of length $d-246$, because
$d+(d-1)+(d-2)+...+(d-245) \geq 245+244+...+1 = 245 \cdot (245+1)/2 = 30135 > 30000$. Thus, he will never be able to perform a jump of length $d-246$ or shorter.

Therefore, we have obtained a working solution: similar to the $O(m^2)$ one, but we will only consider the value of j between $d-245$ and $d+245$. The time and memory complexity of this solution will be $O(m^{1.5})$, since the value "245" is slightly larger than $\sqrt{2m}$.

This solution can be implemented by, for example, using a "normal" two dimensional array with a offset like this: `dp[i][j - offset]` . The time limit is set tight in order to fail most of naive solutions with search using std::map or something, so using hash maps (unordered_map) will be risky although the complexity will be the same as the described solution.

```cpp
http://codeforces.com/contest/505/problem/C
int n,d,offshift,cnt[30005],dp[30005][2005];
int go(int cur,int jump){
    if(cur>30000)return 0;
    if(dp[cur][jump]!=-1)return dp[cur][jump];
    int ret = 0;
    if(cur+jump-1>cur)ret = max(ret, cnt[cur] + go(cur+jump-1,jump-1));
    ret = max(ret, cnt[cur] + go(cur+jump,jump));
    ret = max(ret, cnt[cur] + go(cur+jump+1,jump+1));
    return dp[cur][jump] = ret;
}
int gogo(int cur,int jump){
    if(cur>30000)return 0;
    int ret = 0;
    if(cur+jump-1>cur)ret = max(ret, cnt[cur] + gogo(cur+jump-1,jump-1));
    ret = max(ret, cnt[cur] + gogo(cur+jump,jump));
    ret = max(ret, cnt[cur] + gogo(cur+jump+1,jump+1));
    return ret;
}
int main(){
    scanf("%d%d",&n,&d);
    for(int i=1; i<=n; i++){
        int x; scanf("%d",&x); cnt[x]++;
    }
    if(d<=2000){
        memset(dp,-1,sizeof(dp));
        int ans = go(d,d);
        printf("%d\n",ans);
    }else{
        int ans = gogo(d,d);
        printf("%d\n",ans);
    }
    return 0;
}
/* To make the dp feasible in this problem is to use brute force
when the first jump is very large. Define very large to be 2000.
Then the brute force would have maximum depth of 16, and about 3^16(43046721) operations.
When less than 2000, dp[30001][2000] would fit into time and memory.*/

/*Yes!! I did this in the same way you have explained! But In this way, Among addition
of 3 length, l-1,l,l+1 one shouldn't call the function of length l-1 first.
I have got many accepted code failed on a test case I have made!
The program stop executing because of stack capacity since function calling with
the length l-1 don't bring any solution.
My function have called above 16000 times without return any value in any call
and have stopped for stack capacity! It was indeed a tricky problem!!!)*/

int n,d,offshift,cnt[30005], dp[30005][505];
int fun(int cur,int jump){
    if(cur>30000)return 0;
    if(dp[cur][250+jump]!=-1)return dp[cur][250+jump];
    int ret = 0;
    if(d+jump-1>0)ret = max(ret, cnt[cur] + fun(cur+d+jump-1,jump-1));
    ret = max(ret, cnt[cur] + fun(cur+d+jump,jump));
    ret = max(ret, cnt[cur] + fun(cur+d+jump+1,jump+1));
    return dp[cur][250+jump] = ret;
}
int main(){
    scanf("%d%d",&n,&d);
    for(int i=1; i<=n; i++){
        int x; scanf("%d",&x); cnt[x]++;
    }
    memset(dp,-1,sizeof(dp));
    int ans = fun(d,0);
    printf("%d\n",ans);
    return 0;
}
```