Lomo de queso

Luís trabaja en un bar preparando bocadillos calientes. Desde hamburguesas hasta bocadillos de tortilla, nada escapa a su legendaria habilidad. Sin embargo, su especialidad es el *lomo queso*, un bocadillo que consta de dos filetes de lomo acompañados de un par de lonchas de queso. Puesto que el bar es muy transcurrido y poco organizado, n personas han pedido un *lomo queso* a la vez, y Luís quiere preparar todos los bocadillos en el menor tiempo posible, para no tener que soportar los gritos de su jefe.

Luís dispone de una plancha para cocinar los filetes, donde tiene espacio suficiente para cocer hasta cuatro filetes de lomo simultáneamente. Cada cara del filete tarda exactamente 20 segundos en estar lista. Dado que Luís tiene una extensa experiencia en la profesión, puedes asumir que una vez ha cocinado el lomo por ambas caras, finaliza y sirve el bocadillo inmediatamente con el queso frio. Calcula cuanto tiempo tardará Luís en satisfacer su demanda.

Input Format

La entrada consta de varios casos, un caso por línea. Cada caso lo forma un único número natural n, el número de personas que han pedido un *lomo queso*.

Constraints

$0 \leq n \leq 10^8$

Output Format

Para cada caso, debe imprimirse una única línea con un único número natural: el tiempo (en segundos) que Luís tardará en acabar de preparar los bocadillos.

Sample Input 0

```
1
4
```

Sample Output 0

```
40
80
```

Solución:

```cpp
#include <iostream>

using namespace std;

int main(){
    int n;
    while (cin >> n) {
        if (n == 1) cout << 40 << endl;
        else cout << n*20 << endl;
    }
}
```

Kaitenzushi

En Japón, kaitenzushi es la versión del sushi en comida rápida. Los platos se colocan sobre un transportador de banda que atraviesa el restaurante y pasa por cada mesa y silla de los comensales.

Por delante de tu mesa pasan N platos, numerados del 0 al $i-1$, uno detrás del otro. Una vez ha passado un plato ya no vuelve a pasar. El plato i tiene un precio de p_i yenes.

Tienes Y yenes, y quieres comer platos de manera que te gastes el mayor número de yenes, sin pasarte de Y. Además, como no quieres estar mucho rato esperando, has decidido que solo vas a tomar una secuencia consecutiva de platos, es decir, que vas a elegir dos enteros i, j y vas a tomar todos los platos numerados entre el i y el j.

¿Cuántos yenes puedes gastar como máximo?

Input Format

La entrada consistirá en múltiples casos de prueba. Cada uno de ellos consistirá de dos enteros N, Y, seguidos de N enteros $p_0, \ldots p_{N-1}$. En todos los casos se cumple $1 \leq N \leq 50000$, $1 \leq Y \leq 10^9$, $1 \leq p_i \leq 10^9$.

Constraints

8 puntos: Todos los precios son iguales.

17 puntos: $1 \leq N \leq 200$

25 puntos: $1 \leq N \leq 3000$

50 puntos: Todo tipo de casos.

Output Format

Una sola línea por caso, que contiene la máxima cantidad de yenes que te puedes gastar cumpliendo las restricciones del problema.

Sample Input 0

```
1 120
120
2 160
200 180
3 350
210 170 140
4 390
100 100 100 100
5 1500
100 150 200 250 300
6 950
250 330 180 420 360 270
```

Sample Output 0

```
120
0
310
300
1000
930
```

Explanation 0

En el primer caso, podemos tomar el único plato disponible.

En el segundo caso, no podemos permitirnos ninguno de los dos platos.

En el tercer caso, cogemos los dos últimos platos. Aunque cogiendo el primer y el tercer plato gastamos exactamente 350 yenes, no podemos cogerlos porque no son consecutivos.

Solución:

```cpp
#include <iostream>

using namespace std;

typedef long long ll;

int main() {
    int n;
    ll y;
    while (cin >> n >> y) {
        ll p[n];
        for (int i = 0; i < n; ++i) cin >> p[i];
        ll m = 0;
        ll s = 0;
        int l = 0, r = -1;
        while (r < n) {
           if (s < y) {
               ++r;
               if (r >= n) break;
               s += p[r];
           } else {
               s -= p[l];
               ++l;
           }
            if (s <= y) m = max(s,m);
        }
        cout << m << endl;
    }
}
```

Circuito de carreras

Las masas están enfurecidas. No solo la ciudad no tiene ni una mísera carretera asfaltada, sino que, y probablemente a causa de lo anterior, los ciudadanos aún no pueden disfrutar de un buen circuito de carreras. El alcalde, pretendiendo amansar las turbas ansiosas, decide solventar los dos problemas a la vez construyendo una red de carreteras entre las casas de la ciudad en la que exista un circuito cerrado y que este se pueda alcanzar desde cualquier otra casa. Sin embargo, solo se pueden construir carreteras entre ciertos pares de casas, y cada una a cierto precio. El alcalde pide a sus mejores ingenieros que encuentren el coste mínimo para construir una red que cumpla los requisitos anteriores.

Input Format

La entrada consistirá en varios casos. La primera línea contendrá un entero t, el número de casos a resolver.

Para cada caso:

-La primera línea contendrá dos enteros n, m, el número de casas y el número de carreteras posibles.

-La siguientes m líneas contendrán tres enteros u_k, v_k, c_k cada una, la ciudad de salida, la de llegada, y el coste de construir esa carretera. Las carreteras son bidireccionales y no hay dos carreteras distintas que conecten los mismos puntos.

Se garantiza que existe una red de carreteras que cumple los requisitos necesarios.

Constraints

$3 \leq n \leq 100000$

$n \leq m \leq 200000$

$1 \leq u_k, v_k \leq n,\ u_k \neq v_k$

$1 \leq c_k \leq 20000$

Output Format

La salida debe ser un entero para cada caso, el mínimo coste de construir una red adecuada.

Sample Input 0

```
2
3 3
1 2 1
2 3 2
3 1 3
4 6
1 2 4
1 3 6
1 4 1
2 3 8
2 4 2
3 4 12
```

Sample Output 0

```
6
13
```

Sample Input 1

```
1
8 16
8 2 7
7 1 4
7 6 10
8 7 3
5 8 4
8 1 10
5 7 6
1 3 5
2 1 9
6 4 7
3 5 7
6 5 9
4 3 8
7 4 2
6 8 1
6 1 7
```

Sample Output 1

```
32
```

Solución:

```
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long ll;
typedef pair<int,int> pi;
typedef pair<ll,pi> ppi;
typedef vector<ppi> vppi;
typedef vector<int> vi;

vi P;

int padre(int x) {
    if (P[x] == -1) return x;
    return P[x] = padre(P[x]);
}
```

```
int main() {
    int cas;
    cin >> cas;
    while (cas--) {
        int n, m;
        cin >> n >> m;
        vppi V(m);
        P = vi(n+1,-1);
        for (int i = 0; i < m; ++i)
            cin >> V[i].second.second >> V[i].second.first >>
                                                    V[i].first;
        sort(V.begin(),V.end());
        ll res = 0;
        bool primer = true;
        for (int i = 0; i < m; ++i) {
            int a1 = padre(V[i].second.first);
            int a2 = padre(V[i].second.second);
            if (a1 != a2) {
                P[padre(a1)] = padre(a2);
                res += V[i].first;
            } else if (primer) {
                primer = false;
                res += V[i].first;
            }
        }
        cout << res << endl;
    }
    return 0;
}
```