EJERCICIOS BUSQUEDA BINARIA

PROBLEMA 1 — “Antenas Solares en el Desierto”

Planteamiento

En un desierto se encuentran **N estaciones de investigación**, ubicadas en posiciones enteras (ordenadas no necesariamente).
La empresa energética colocará **M antenas solares**, también con posiciones enteras conocidas.

Cada antena provee energía a estaciones que estén a distancia **≤ R**.

Objetivo:

Encontrar el **valor mínimo de R** para que **todas** las estaciones reciban energía de **al menos una antena**.

Razonamiento (Búsqueda Binaria sobre R)

- Si R es pequeño → varias estaciones quedan sin cobertura.
- Si R es muy grande → todas quedan cubiertas.
- Luego, existe un R mínimo que funciona.
- Se usa búsqueda binaria sobre R y una función `check(R)`
  que verifica si todas las estaciones tienen alguna antena a distancia ≤ R.

CODIGO

```
#include <iostream>
#include <algorithm>
using namespace std;

int N, M;

bool check(int *est, int *ant, int R) {
   int j = 0;
   for (int i = 0; i < N; i++) {
      while (j < M && ant[j] + R < est[i]) j++;
      if (j == M) return false;
      if (abs(ant[j] - est[i]) > R) return false;
   }
   return true;
}

int main() {
   cin >> N >> M;

   int *est = new int[N];
   int *ant = new int[M];

   for (int i = 0; i < N; i++) cin >> est[i];
   for (int i = 0; i < M; i++) cin >> ant[i];
```

```
    sort(est, est + N);
    sort(ant, ant + M);

    int low = 0, high = 1000000000;

    while (low < high) {
        int mid = (low + high) / 2;
        if (check(est, ant, mid)) high = mid;
        else low = mid + 1;
    }

    cout << low;

    delete[] est;
    delete[] ant;
    return 0;
}
```

PROBLEMA 2 — “Distribución Óptima de Tanques de Agua”

Planteamiento

Una zona agrícola tiene **K pozos de agua**, ubicados en posiciones enteras ordenadas. Los agricultores desean instalar **N tanques de riego**, cada uno en una posición entera (que tú eliges), pero **dentro del rango total** de los pozos.

Quieren colocar los N tanques de forma que la **distancia mínima entre cualquier par** sea lo más grande posible.

Devuelve la distancia más grande posible.

Razonamiento (Búsqueda Binaria sobre la distancia mínima `D`)

- Si eliges una distancia mínima D muy grande → no puedes colocar N tanques.
- Si eliges D muy pequeña → sí puedes.
- Hay un D máximo posible → lo buscamos con binary search.
- La función `check(D)` intenta colocar tanques greedy:
    - El primero en la posición del primer pozo,
    - El siguiente lo más cerca posible pero cumpliendo distancia ≥ D.

CODIGO

```
#include <iostream>
#include <algorithm>
using namespace std;

int K, N;
```

```
bool check(int *pozos, int D) {
   int grupos = 1;
   int prev = pozos[0];

   for (int i = 1; i < K; i++) {
      if (pozos[i] - prev >= D) {
         grupos++;
         prev = pozos[i];
         if (grupos == N) return true;
      }
   }
   return false;
}

int main() {
   cin >> K >> N;
   int *pozos = new int[K];

   for (int i = 0; i < K; i++) cin >> pozos[i];
   sort(pozos, pozos + K);

   int low = 0;
   int high = pozos[K - 1] - pozos[0];

   while (low < high) {
      int mid = (low + high + 1) / 2;

      if (check(pozos, mid))
         low = mid;
      else
         high = mid - 1;
   }

   cout << low;

   delete[] pozos;
   return 0;
}
```

PROBLEMA 3 — “Carga Mínima de Drones Mensajeros”

Planteamiento

Una compañía tiene **N paquetes**, cada uno con un peso entero.
Poseen **M drones**, y cada drone solo puede cargar paquetes **consecutivos** (no se puede saltar).

Cada drone tiene una **capacidad máxima C**, y se desea encontrar **el menor valor de C** tal que los M drones puedan llevar todos los paquetes sin exceder su capacidad.

Razonamiento (Binary Search sobre C)

- Si C < peso máximo de un paquete → imposible.
- Si C muy grande → todos los paquetes pueden ir en 1 drone → válido.
- Se busca el mínimo C que permita usar ≤ M drones.
- La función `check(C)` simula:
  - Suma paquetes hasta que no pueda más, y crea un nuevo drone.
  - Si se usan más de M drones → C es muy pequeño.

CODIGO

```
#include <iostream>
using namespace std;

int N, M;

bool check(int *p, int C) {
   int used = 1;
   int suma = 0;

   for (int i = 0; i < N; i++) {
      if (p[i] > C) return false;

      if (suma + p[i] > C) {
         used++;
         suma = p[i];
         if (used > M) return false;
      } else {
         suma += p[i];
      }
   }
   return true;
}

int main() {
   cin >> N >> M;
   int *p = new int[N];

   int low = 0, high = 0;

   for (int i = 0; i < N; i++) {
      cin >> p[i];
      if (p[i] > low) low = p[i];
      high += p[i];
   }

   while (low < high) {
      int mid = (low + high) / 2;

      if (check(p, mid))
         high = mid;
      else
```

```
            low = mid + 1;
    }

    cout << low;

    delete[] p;
    return 0;
}
```