Medidas estadísticas
====================

1) Estructura de datos a utilizar:

   -- Qué tiene que almacenar?: secuencia de números (puede haber repetidos)

   -- Qué tiene que hacer?:
          * consultar el mínimo
          * borrar el mínimo (uno de ellos si está repetido)
          * consultar máximo y average (suma de todos y número de elementos)

   -- alternativas:
          * cola de prioridad (>): el mínimo es top(), número de elementos es size()
            máximo (int)
            suma_todos (int)

          * set:
              + de enteros: NO (hay repetidos)
              + de pares {número, apariciones}:
                  -> el mínimo es begin(), el máximo en --end()
                  -> pero sus valores no se pueden modificar (simular borrar e insertar)
                  -> se complica la gestión de las operaciones
            suma_todos (int)
            num_elems (int)

          * diccionario {número, apariciones}: el mínimo es begin(), el máximo en --end()
            suma_todos (int)
            num_elems (int)

2) Cola de prioridad ordenada según > (por defecto es <):

   -- priority_queue<int, vector<int>, greater<int>> pq;

3) Ojo con la división:
   -- suma/pq.size()   : es un valor entero
   -- suma/double(pq.size()) : es un valor real

4) Tiempo ejecución con n operaciones: O(n log n)

Filling the bag
===============

1) Estructura de datos a utilizar:

    -- Qué tiene que almacenar?: secuencia de números (sin repetidos)

    -- Qué tiene que hacer?:
         * saber la suma de los k números más grandes
         * borrar un valor dado cualquiera

    -- alternativas:
         * cola de prioridad: NO (no se puede borrar cualquier número)
         * set
         * diccionario {número, ?}: no hace falta asociarle ningún valor a la clave
              -> estructura demasiado compleja para lo que necesitamos

2) Algoritmos:

    -- un único set que representa todas las joyas que hay:
              -> insertar joya: logarítmico
              -> quitar joya: logarítmico
              -> saber suma de los k primeros números (recorrer k): lineal

    -- dos sets: uno con las joyas de Ali y otro con el resto + mantener suma en joyas Ali
              -> insertar joya: logarítmico
              -> quitar joya: logarítmico
              -> saber suma de los k primeros números: constante (se mantiene calculada)

3) Set ordenado según > (por defecto es <):

```
    -- insertar el valor numérico negado (el primer elemento será el máximo)
    -- set<int, greater<int>> s;  (solución generalizable a otros tipos de datos)

    -- set<string, decltype(&cmp)> s(&cmp);   // bool cmp(string a, string b)
                                              //     con cualquier orden
```

4) Cómo representar enteros muy grandes:

```
    -- int no puede representar cualquier entero
    -- alternativas:
          -> long (long int)
          -> long long (long long int)
    -- saber cuántos bytes tiene un tipo de datos:
          -> cout << sizeof(int) << endl;
```

Casino
==========

1) Estructura de datos a utilizar:

```
    -- Qué tiene que almacenar?: pares {nombre, ganancia} (el par representa a un jugador)

    -- Qué tiene que hacer?:
         * acceder por nombre a la info del jugador
         * insertar nuevo jugador con ganancia 0 (saber si ya estaba)
         * borrar jugador y saber su ganancia    (saber si no estaba)
         * modificar ganancia de un jugador      (saber si está)
         * listar jugadores ordenados por nombre

    -- alternativas:
         * set de pares {nombre, ganancia}: NO (no tiene acceso sólo por nombre)

         * diccionario {nombre, ganancia}
              -> sus operaciones permiten hacer todo lo necesario
```

2) Listar jugadores:

```
    for (auto jugador : dict) {
        cout << jugador.first << " is winning " << jugador.second << endl;
    }

    -- donde el tipo de datos de jugador es:

    for (pair<string, int> jugador : dict) {
        cout << jugador.first << " is winning " << jugador.second << endl;
    }
```