SISTEMAS OPERATIVOS

Asignatura: Sistemas Operativos

Grupo: 01

Número de práctica: 3

Nombre 1: Haytam Abaran Said

Nombre 2: Pablo Martínez Onofre

MEMORIA DE LA PRÁCTICA 3

GESTIÓN DE MEMORIA

Esta práctica la hemos realizado en el lenguaje C++.
Las librerías usadas para el programa cliente son:
- include <iostream> : es la biblioteca estándar en C++ para poder tener acceso a los dispositivos estándar de entrada y/o salida.
- include <vector> : es una plantilla de clase para contenedores de secuencia.
- include <stdlib.h> : es una biblioteca que contiene los prototipos de funciones de C para gestión de memoria dinámica, control de procesos y sirve para convertir un entero a cadena de caracteres.
- include <string.h> : es una biblioteca que contiene un conjunto de funciones para manipular cadenas: copiar, cambiar caracteres, comparar cadenas, etc.
- include <fstream> : es una biblioteca para poder usar los "flujos de datos de ficheros" (cuyo manejo será muy parecido al de la consola, en la que escribíamos con "cout" y leíamos con "cin". Un fichero de escritura será un "ofstream", abreviatura de "output file stream".
- include <sstream> : es una biblioteca que se utiliza para la inserción y extracción de datos hacia / desde los objetos de cadena. Actúa como una secuencia para el objeto de cadena.

Llamada *swap,* recibe dos parámetros e intercambia el valor del uno por el otro.

En esta práctica se pedía realizar un gestor de memoria sin importar el lenguaje de programación. Hemos utilizado C++ ya que es uno de los lenguajes que más hemos practicado.

Lo primero que hemos hecho ha sido crearnos dos structs que es un tipo de estructura en el que podemos encapsular distintos tipos de variables como puede ser enteros, strings.

En el primero que nos hemos creado, ha sido para guardar la información de los procesos, es por eso que necesitamos toda la información que vamos a tener que leer del fichero “entrada.txt” como puede ser el nombre del proceso, el momento de llegada, la memoria requerida, etc.

El segundo struct que nos hemos creado llamado ‘Processor’ en esta segunda estructura hemos metido dos vectores, el primero lo utilizaremos para ver y meter el conjunto de procesos que están encolados, y el segundo vector es para el conjunto de procesos que están siendo ejecutados. Además, tenemos un entero en el que vamos a tener el total de memoria que corresponde con el tamaño de 2000 bytes que nos expone el enunciado.
Lo primero que vamos a tener que hacer es tener que leer del fichero, que lo utilizaremos en ambos algoritmos tanto en el del peor hueco como en el mejor hueco.

Para leer del fichero lo primero que tenemos que saber es el nombre del fichero de entrada, para ello le pasamos como argumento ‘fichero’ y además todos los procesos que vamos a ir guardando los vamos a tener que ir metiendo en el struct ‘Processor’, porque si no a la hora de utilizar el algoritmo correspondiente no tendremos los procesos necesarios cargados en memoria.

Por otra parte, para leer del fichero primero tenemos que abrir el fichero, si no se abre bien dará error. En caso contrario, tendremos que leer línea a línea del fichero mediante el stringstream e ir guardando los en una variable auxiliar que es una instancia del struct ‘Processor’, una vez hemos leído un proceso y cumple con la restricción de ser mayor que 100 (memoria mínima) y menor o igual que 2000 (memoria máxima) tenemos que hacer push back que lo que hace es añadirlo al vector.

El siguiente método que hemos implementado es el ‘ampliarProcessor’ en este método tenemos una posición y a partir de esa posición vamos a tener que añadir huecos con un tamaño de ‘memoriaRestante’ y todo lo demás a 0.

También necesitaremos un método en el que podamos eliminar los procesos según se vayan ejecutando, por eso tendremos que ir decrementando la variable del vector del procesador que se está ejecutando.

Seguidamente, nos hemos tenido que crear un método para los procesos que se están ejecutando en especial los que tienen nombre de ‘hueco’, en este caso lo que tenemos que hacer es sumarlos y dejarlos de forma contigua y la unión de estos dará un único proceso que será también un hueco.

Por consiguiente, tenemos un método llamado ‘queueToProcessor’, en el que tenemos un struct ‘Processor’ y un booleano en el que nos indica si lo tenemos que borrar el proceso que se está ejecutando o no.

El método ‘ProcessToProcesador’, lo primero que se hace en este método es que se mira a ver si hay un hueco bueno para el 1er Process de la queue. Si el primer elemento del vector Process cabe en un hueco. Si cabe justo simplemente se sustituyen los valores (cambiamos el que se está ejecutando con el que está en la cola). Sino hay que crear un nuevo hueco con la memoria restante, en el que le tenemos que desplazar todos los procesos una posición a la derecha y hacemos el hueco.

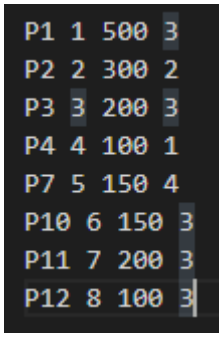
Según el algoritmo que toque en cada instante del proceso tenemos que mostrar una gráfica, para ello creamos un método que recibe un ‘Processor’ y según el tamaño del procesador que se está ejecutando ponemos el nombre del procesador y el tamaño de la memoria del procesador, y vamos poniendo el siguiente instante leyendo cualquier caracter (intro) mediante el cin.get().

Las particiones para cada proceso se van creando a medida que son asignadas al procesador. Tiene como ventaja principal que evitamos el desperdicio de memoria dentro de cada bloque ya que cada uno está hecho a medida para el proceso que contiene.

En el método de ‘MejorHueco’ consiste en asignarle al proceso el hueco con menor desperdicio interno, el hueco el cual al serle asignado el proceso deja menos espacio sin utilizar.

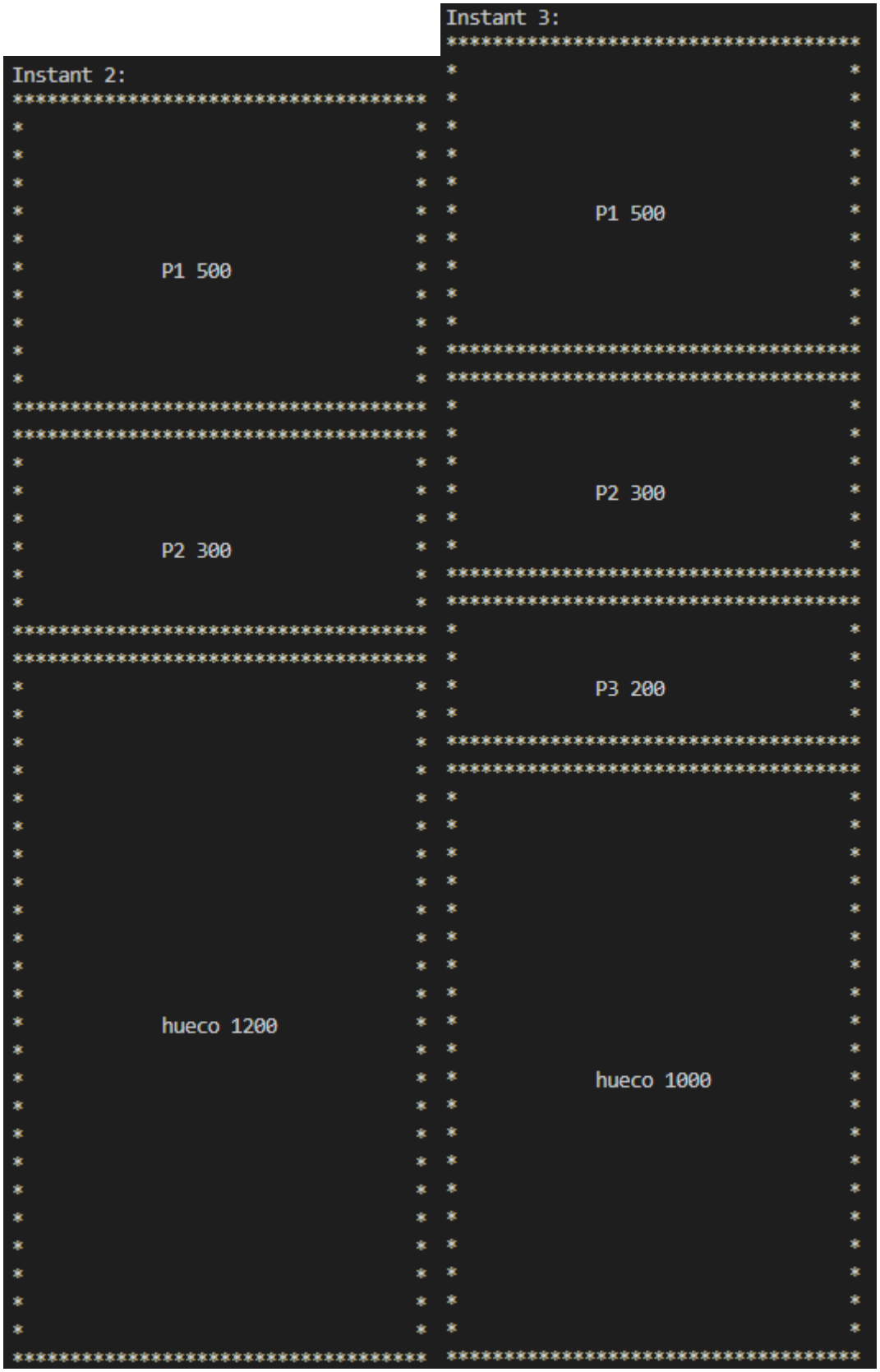
En el método de ‘PeorHueco’ al contrario que el criterio anterior, se busca el hueco con mayor desperdicio interno, el hueco el cual al serle asignado el proceso deja más espacio sin utilizar, y se corta de él el trozo necesario.

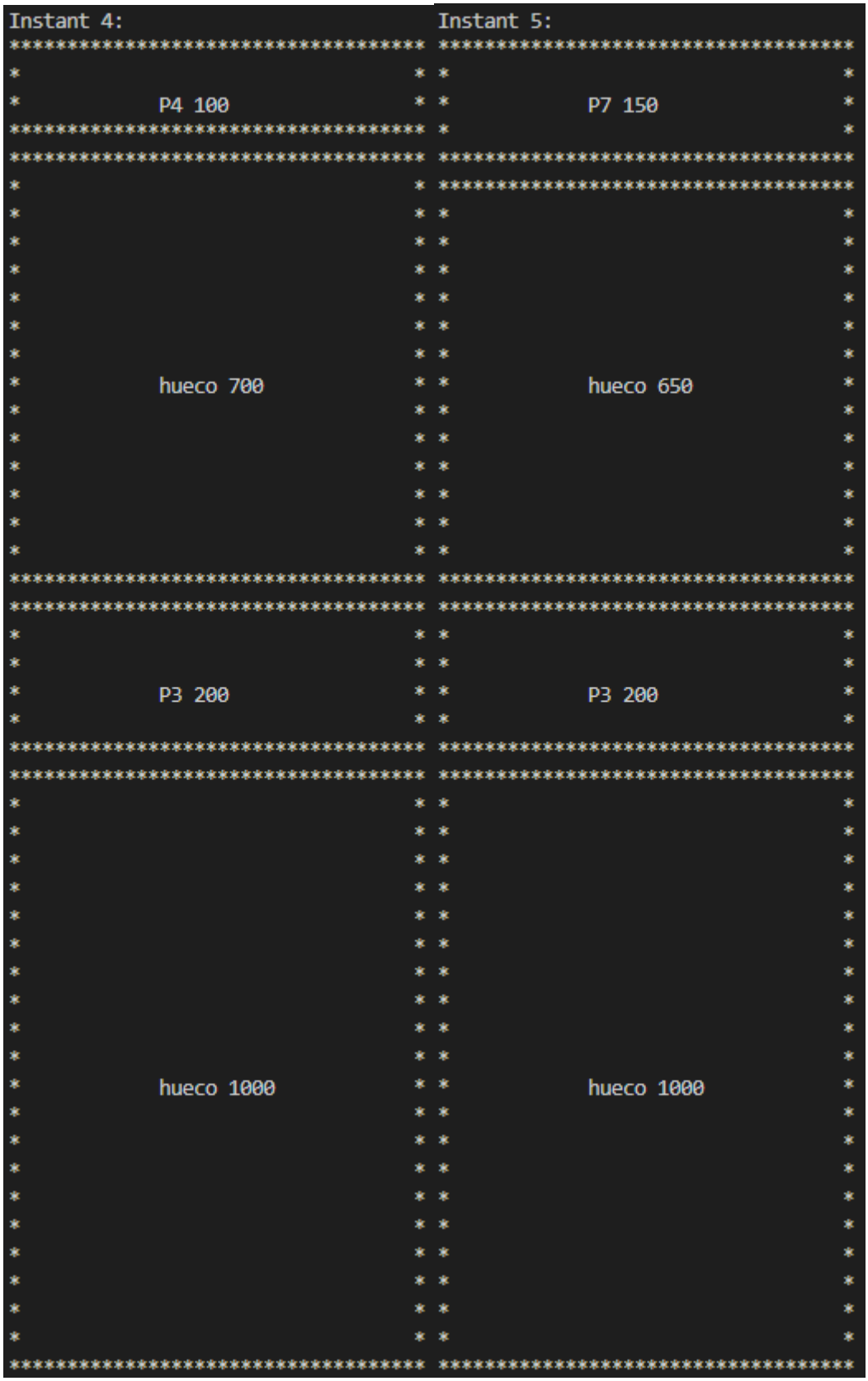
ARCHIVO DE ENTRADA.TXT

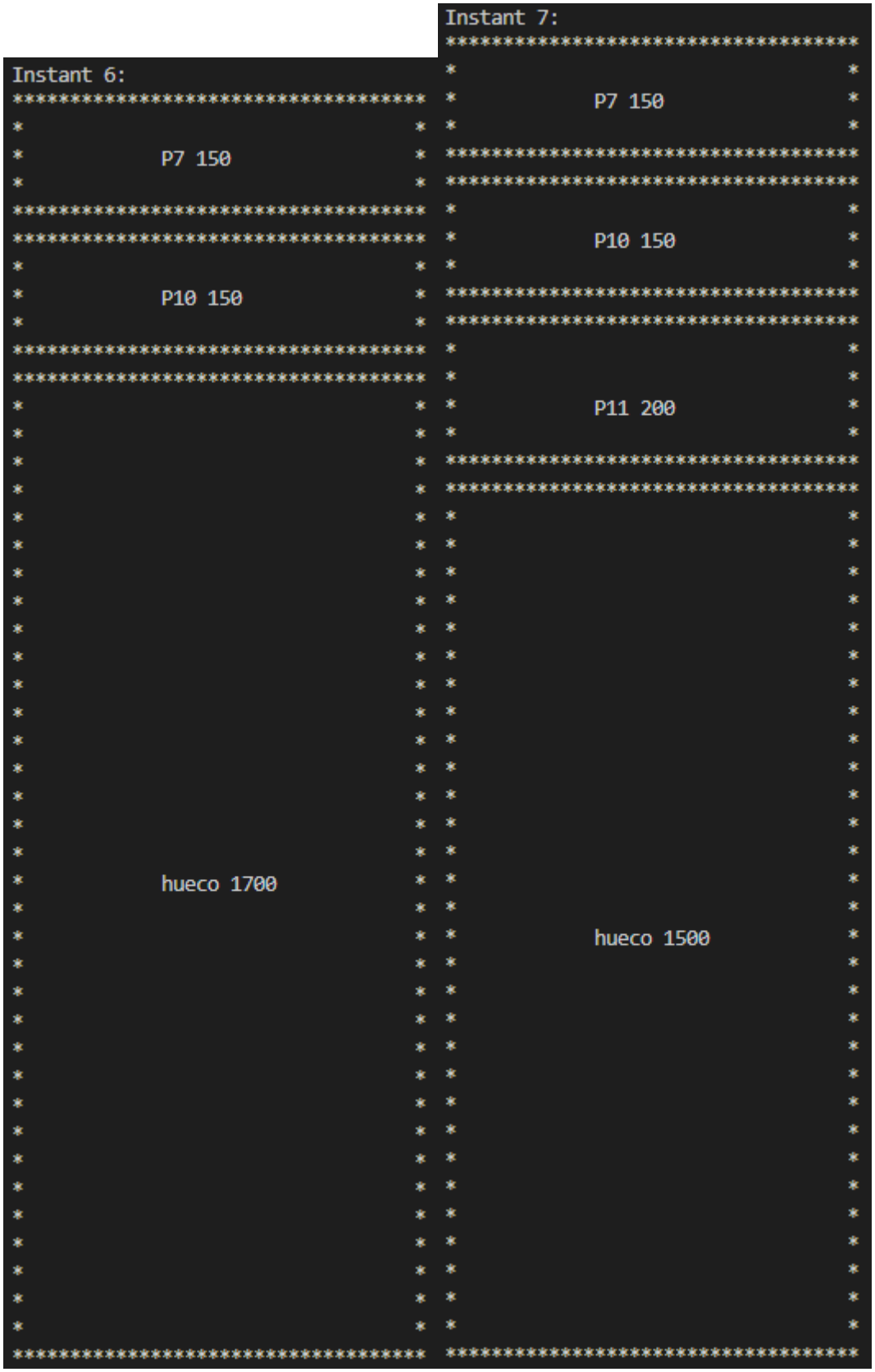


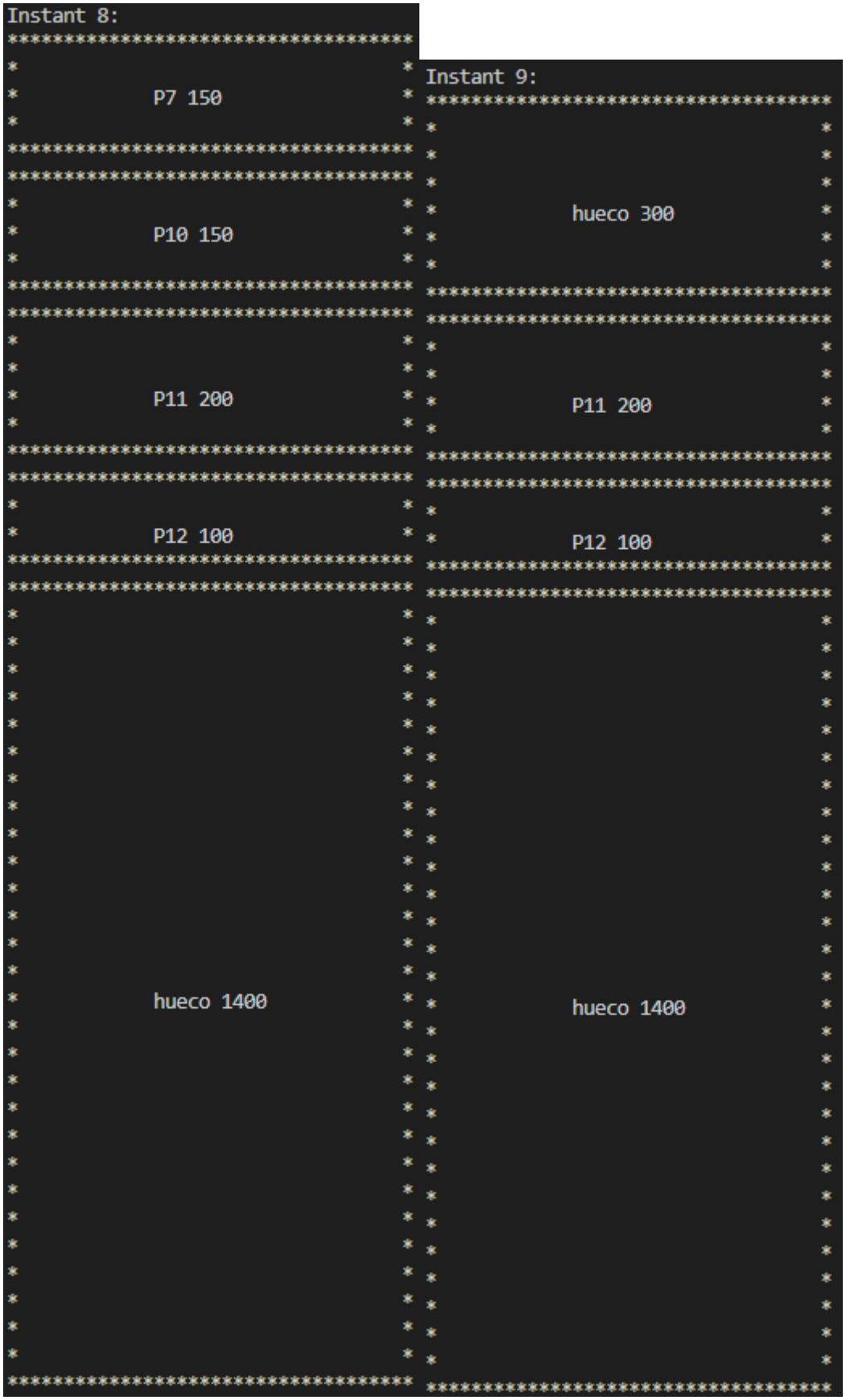
ALGORITMO DE MEJOR HUECO

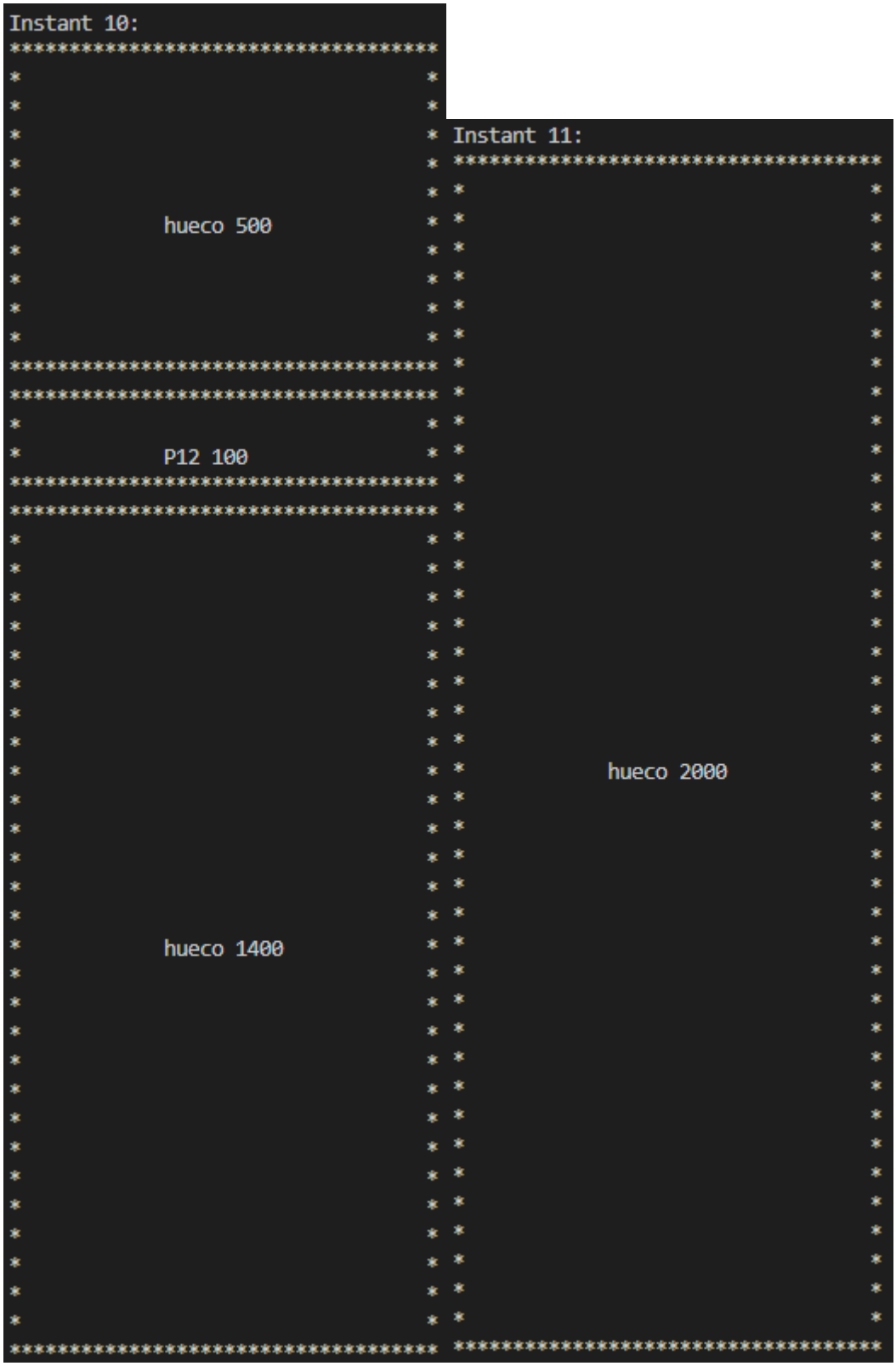
```
PS C:\Users\hayta\Documents\MATERIAS UA> g++ -Wall -o prog gestionmemoria.cc
PS C:\Users\hayta\Documents\MATERIAS UA> .\prog entrada.txt
Chosen input file: entrada.txt
Chosen dump file: Particiones.txt
Maximum processor capacity: 2000
=======================================================
What algorithm do you want to use?
1) Press 1 to use the best gap algorithm
2) Press 2 to use the worst gap algorithm
Enter the memory allocation model: 1

You have selected the best gap algorithm
Instant 1:
**********************************
*                                *
*                                *
*                                *
*                                *
*                                *
*           P1 500               *
*                                *
*                                *
*                                *
*                                *
**********************************
**********************************
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*           hueco 1500           *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
*                                *
```


Instant 2:
P1 500
P2 300
hueco 1200
Instant 3:
P1 500
P2 300
P3 200
hueco 1000


Instant 4:
P4 100
hueco 700
P3 200
hueco 1000
Instant 5:
P7 150
hueco 650
P3 200
hueco 1000


Instant 6:
P7 150
P10 150
hueco 1700
Instant 7:
P7 150
P10 150
P11 200
hueco 1500


Instant 8:
P7 150
P10 150
P11 200
P12 100
hueco 1400
Instant 9:
hueco 300
P11 200
P12 100
hueco 1400

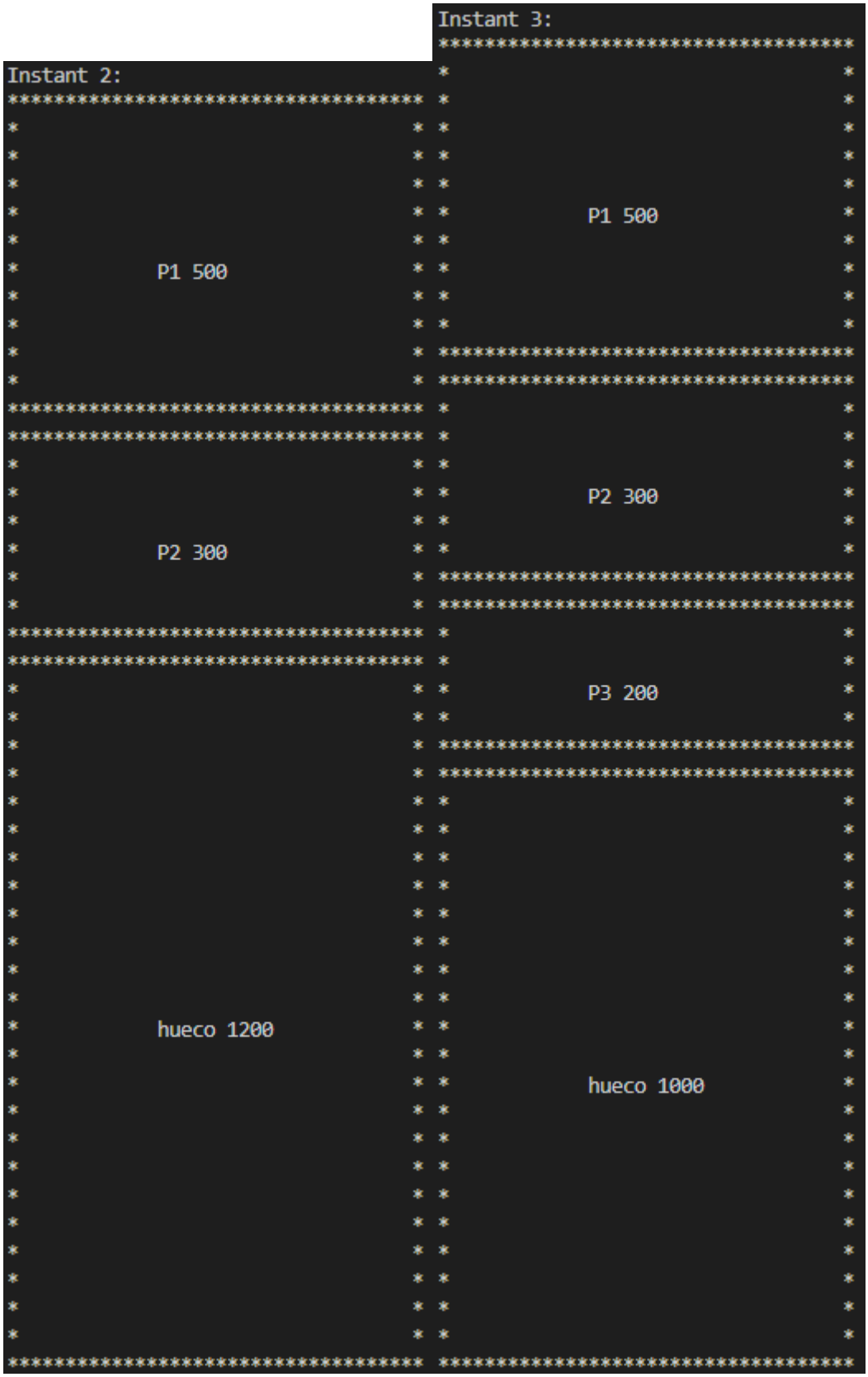

Instant 10:
hueco 500
P12 100
hueco 1400
Instant 11:
hueco 2000

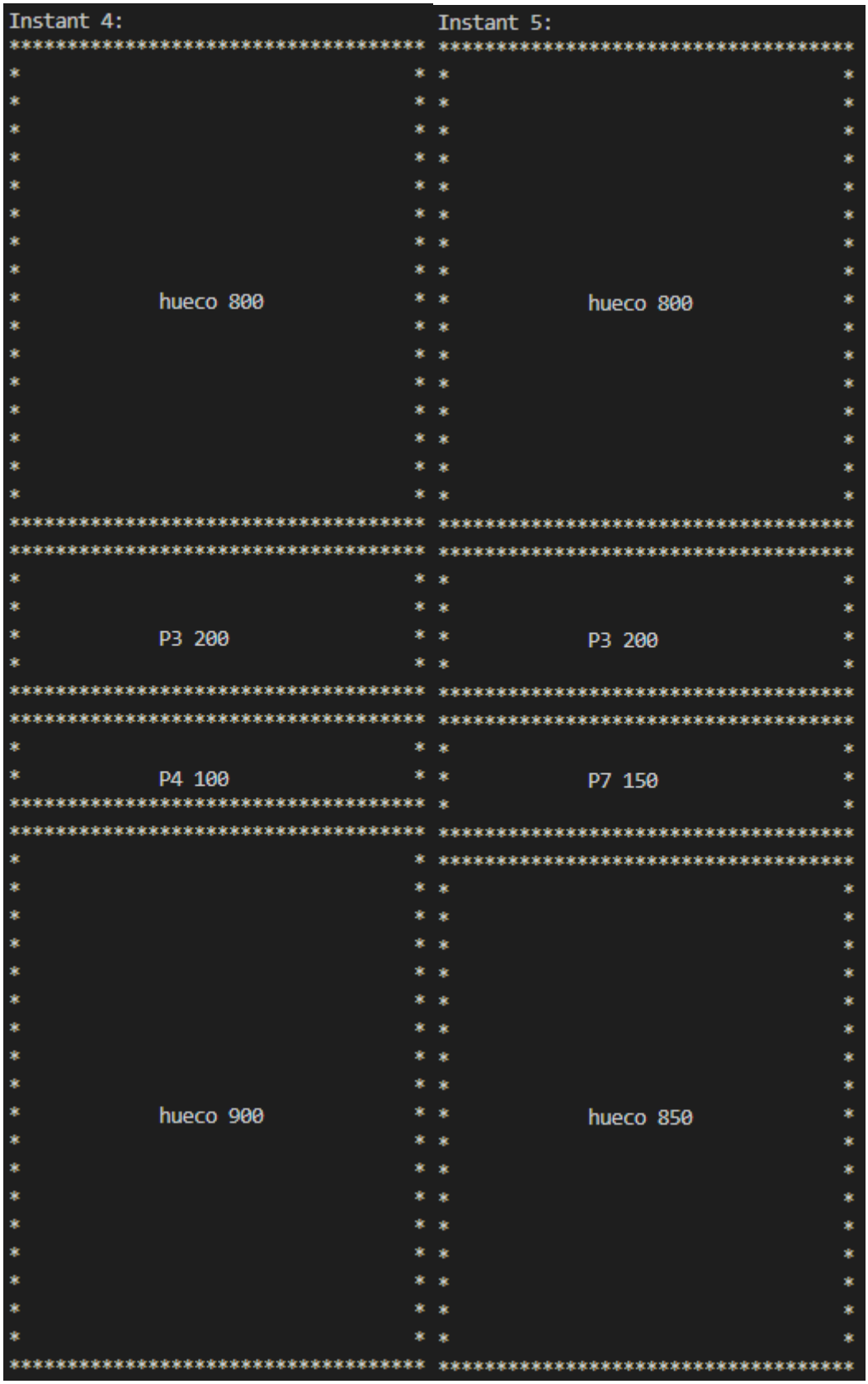
RESULTADO EN PARTICIONES.TXT

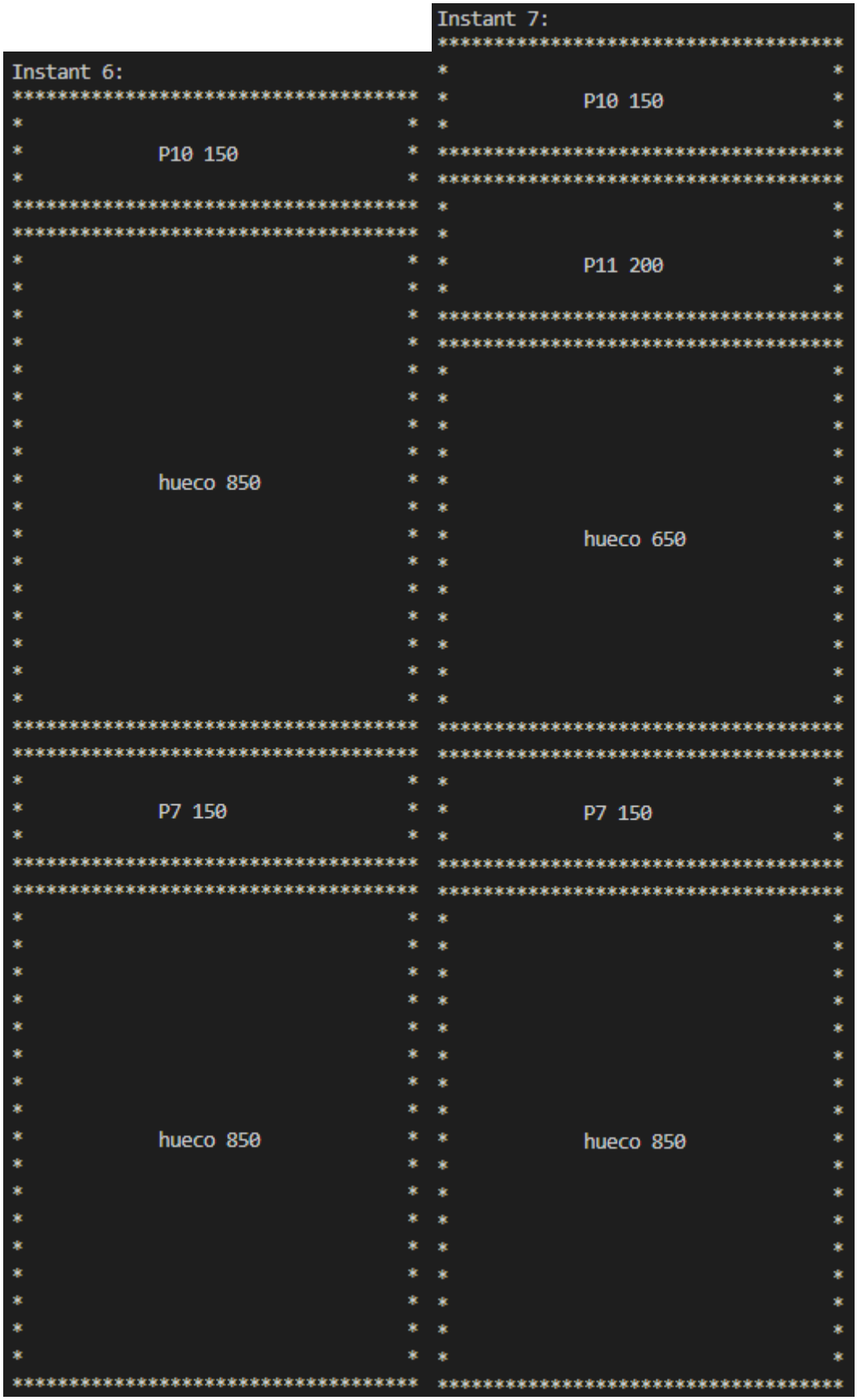
```
1 [0 P1 500] [500 hueco 1500]
2 [0 P1 500] [500 P2 300] [800 hueco 1200]
3 [0 P1 500] [500 P2 300] [800 P3 200] [1000 hueco 1000]
4 [0 P4 100] [100 hueco 700] [800 P3 200] [1000 hueco 1000]
5 [0 P7 150] [150 hueco 650] [800 P3 200] [1000 hueco 1000]
6 [0 P7 150] [150 P10 150] [300 hueco 1700]
7 [0 P7 150] [150 P10 150] [300 P11 200] [500 hueco 1500]
8 [0 P7 150] [150 P10 150] [300 P11 200] [500 P12 100] [600 hueco 1400]
9 [0 hueco 300] [300 P11 200] [500 P12 100] [600 hueco 1400]
10 [0 hueco 500] [500 P12 100] [600 hueco 1400]
11 [0 hueco 2000]
```

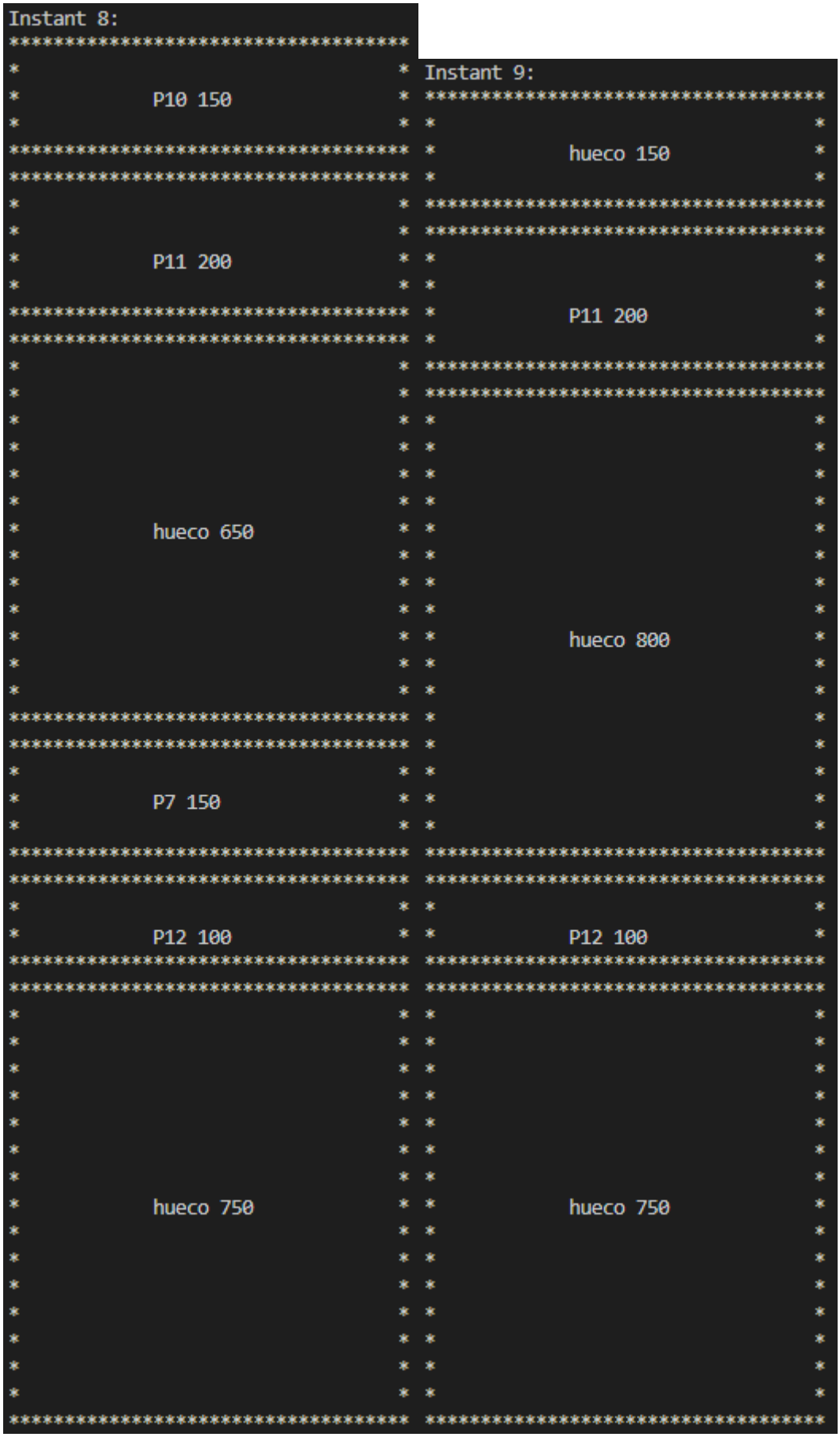
ALGORITMO DE PEOR HUECO

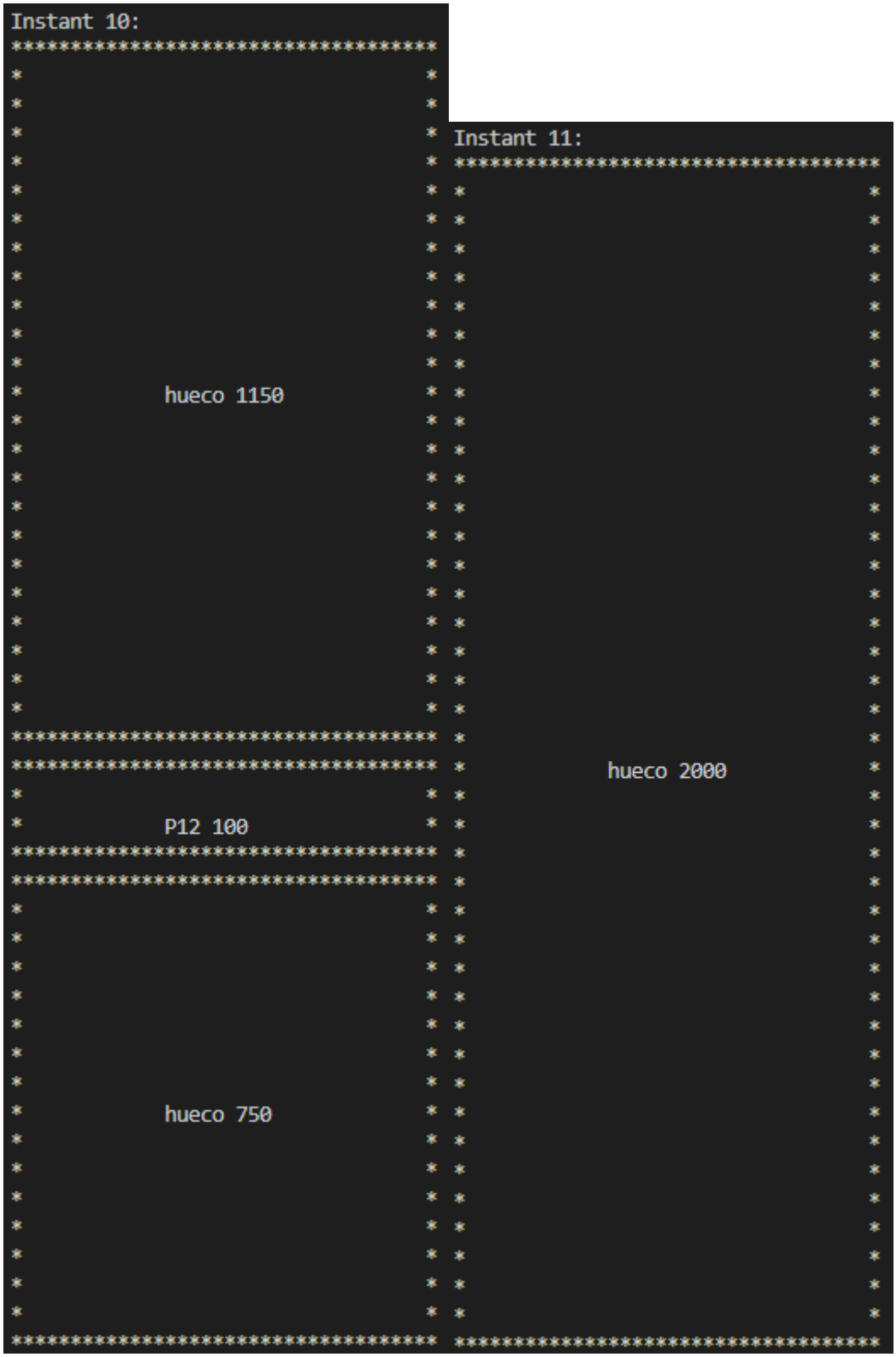
```
PS C:\Users\hayta\Documents\MATERIAS UA> .\prog entrada.txt
Chosen input file: entrada.txt
Chosen dump file: Particiones.txt
Maximum processor capacity: 2000
=========================================================
What algorithm do you want to use?
1) Press 1 to use the best gap algorithm
2) Press 2 to use the worst gap algorithm
Enter the memory allocation model: 2

You have selected the worst gap algorithm
Instant 1:
************************************
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*            P1 500                *
*                                  *
*                                  *
*                                  *
*                                  *
************************************
************************************
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*            hueco 1500            *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
*                                  *
```


Instant 2:
P1 500
P2 300
hueco 1200
Instant 3:
P1 500
P2 300
P3 200
hueco 1000


Instant 4:
hueco 800
P3 200
P4 100
hueco 900
Instant 5:
hueco 800
P3 200
P7 150
hueco 850


Instant 6:
P10 150
hueco 850
P7 150
hueco 850
Instant 7:
P10 150
P11 200
hueco 650
P7 150
hueco 850


Instant 8:
P10 150
P11 200
hueco 650
P7 150
P12 100
hueco 750
Instant 9:
hueco 150
P11 200
hueco 800
P12 100
hueco 750


Instant 10:
hueco 1150
P12 100
hueco 750
Instant 11:
hueco 2000

RESULTADO EN PARTICIONES.TXT

```
1 [0 P1 500] [500 hueco 1500]
2 [0 P1 500] [500 P2 300] [800 hueco 1200]
3 [0 P1 500] [500 P2 300] [800 P3 200] [1000 hueco 1000]
4 [0 hueco 800] [800 P3 200] [1000 P4 100] [1100 hueco 900]
5 [0 hueco 800] [800 P3 200] [1000 P7 150] [1150 hueco 850]
6 [0 P10 150] [150 hueco 850] [1000 P7 150] [1150 hueco 850]
7 [0 P10 150] [150 P11 200] [350 hueco 650] [1000 P7 150] [1150 hueco 850]
8 [0 P10 150] [150 P11 200] [350 hueco 650] [1000 P7 150] [1150 P12 100] [1250 hueco 750]
9 [0 hueco 150] [150 P11 200] [350 hueco 800] [1150 P12 100] [1250 hueco 750]
10 [0 hueco 1150] [1150 P12 100] [1250 hueco 750]
11 [0 hueco 2000]
```