Problema 1

Se usa ordenamiento por selección.

Problema 2

Se usa ordenamiento por selección.

Problema 3

Se pueden tener dos arreglos, uno con las alturas y otro con los nombres. Para destruir una montaña, podemos hallar el índice correspondiente a la montaña de mayor altura iterando sobre todos los elementos. Al encontrar este índice, podemos simular el "destruir" la montaña haciendo la altura correspondiente igual a cero e imprimimos el nombre de dicha montaña. Se repite este proceso N veces.

Problema 4

Este problema simplemente se trata de simular el paso de "mezcla" del ordenamiento por mezcla.

Problema 5

Se hace un análisis matemático en base a la suma de estos valores mínimos y máximos en el árbol de subproblemas. El mínimo es $\frac{k}{2} \times \log_2 k$ y el máximo $k \times \log_2 k - (k-1)$.

Problema 6

Podemos re-escribir el ordenamiento mezcla para reforzar los conceptos aunque también se puede usar la función sort de la librería algorithm directamente.

Problema 7

En este problema debemos realizar un torneo binario por cada extracción. Solamente es necesario imprimir lo que sucede en la extracción número K. Para implementar el torneo binario de una extracción, podemos guardar las posiciones de las L cabezas y avanzarlas cada vez que sea necesario durante el transcurso del torneo binario.

Problema 8

Se modifica ligeramente el paso de mezcla en el algoritmo de ordenamiento por mezcla de tal forma que sean los elementos de uno de los arreglos en el orden inverso.

Problema 9

Usamos la función sort de la librería algorithm directamente.

Problema 10

Ordenamos las alturas de los postes usando la función sort. Luego iteramos guardando la mínima diferencia comparando la diferencia entre postes con distancia k.

Problema 11

Para poder guardar el número y su frecuencia, podemos hacer uso de un struct (colección de datos) o también usar un std::pair. En caso de usar std::pair no es necesaria una función de comparación, pues std::sort ordena en base al primer elemento. Sino, hay que usar una función de comparación que compare la cantidad de veces que aparezca un número (frecuencia). Luego, debemos contar cuántas veces aparece cada número y guardar dicha cantidad en un mapa on en un arreglo, de struct o std::pair (como se prefiera). Usamos la función std::sort y tenemos nuestro resultado.

Problema 12

Para este problema se puede crear una struct que representa una persona y una función de comparación que compare dos personas en base a su icm. Se leen los datos y luego se usa la función sort con la función de comparación como tercer argumento. Al final se itera sobre el arreglo y se imprime la salida.