Pseudocódigo realizado

```
INICIO Worker0

  Definir PUERTO = 8080
  Definir PUERTO_WORKER1 = 8081
  Definir PUERTO_CLIENTE = 9090
  Definir clienteIP

  Iniciar servidor en PUERTO para escuchar conexiones

  IMPRIMIR "Worker0 iniciado en puerto " + PUERTO
  IMPRIMIR "IP del Worker0: " + Obtener IP local

  MIENTRAS siempre (bucle infinito para aceptar clientes):

    Aceptar conexión de cliente

    Obtener IP del cliente
    IMPRIMIR "Cliente conectado desde: " + clienteIP

    Llamar a manejarCliente(clienteSocket, True)

  FIN mientras

FIN Worker0
```

PROCEDIMIENTO manejarCliente(socket, esDirectoDesdeCliente):

Crear flujo de entrada (ObjectInputStream) desde socket

Crear flujo de salida (ObjectOutputStream) si no esDirectoDesdeCliente

Leer el vector de enteros

Leer el método de ordenamiento

Leer el tiempo límite

Leer la IP del cliente

IMPRIMIR "[Worker0] Iniciando ordenamiento..."

Llamar a procesarOrdenamiento(vector, metodoOrdenamiento, tiempoLimite, salida, PUERTO_WORKER1)

FIN manejarCliente

PROCEDIMIENTO procesarOrdenamiento(vector, metodoOrdenamiento, tiempoLimite, salida, puertoDestino):

Crear objeto de Ordenamientos

Crear hilo de ordenamiento

Iniciar hilo de ordenamiento

Esperar que el hilo termine hasta el tiempo límite (tiempoLimite * 1000 milisegundos)

SI el hilo sigue vivo (tiempo agotado):

IMPRIMIR "[Worker0] Tiempo agotado. Reenviando a Worker1..."

Interrumpir hilo de ordenamiento

Llamar a reenviarAWorker(vector, metodoOrdenamiento, tiempoLimite, "localhost", puertoDestino)

SINO (ordenamiento completado):

IMPRIMIR "[Worker0] Ordenamiento completado. Enviando al cliente..."

Llamar a enviarAlCliente(vector, clienteIP)

FIN procesarOrdenamiento

PROCEDIMIENTO reenviarAWorker(vector, metodoOrdenamiento, tiempoLimite, host, puerto):

Crear socket a "host" y "puerto"

Crear flujo de salida (ObjectOutputStream)

Enviar vector, metodoOrdenamiento, tiempoLimite y clienteIP

FIN reenviarAWorker

PROCEDIMIENTO enviarAlCliente(vector, clienteIP):

Crear socket a clienteIP en PUERTO_CLIENTE

Crear flujo de salida (ObjectOutputStream)

Enviar el vector al cliente

FIN enviarAlCliente

```
INICIO Worker1

  Definir PUERTO = 8081
  Definir PUERTO_WORKER0 = 8080
  Definir PUERTO_CLIENTE = 9090
  Definir clienteIP

  Iniciar servidor en PUERTO para escuchar conexiones

  IMPRIMIR "Worker1 iniciado en puerto " + PUERTO
  IMPRIMIR "IP del Worker1: " + Obtener IP local

  MIENTRAS siempre (bucle infinito para aceptar conexiones):

    Aceptar conexión de Worker0

    IMPRIMIR "Conexion recibida desde: " + dirección de la conexión

    Llamar a manejarCliente(workerSocket)

  FIN mientras

FIN Worker1

PROCEDIMIENTO manejarCliente(socket):
  Crear flujo de entrada (ObjectInputStream) desde socket
```

Leer el vector de enteros

Leer el método de ordenamiento

Leer el tiempo límite

Leer la IP del cliente

IMPRIMIR "[Worker1] Iniciando ordenamiento..."

Llamar a procesarOrdenamiento(vector, metodoOrdenamiento, tiempoLimite, PUERTO_WORKER0)

FIN manejarCliente

PROCEDIMIENTO procesarOrdenamiento(vector, metodoOrdenamiento, tiempoLimite, puertoDestino):

Crear objeto de Ordenamientos

Crear hilo de ordenamiento

Iniciar hilo de ordenamiento

Esperar que el hilo termine hasta el tiempo límite (tiempoLimite * 1000 milisegundos)

SI el hilo sigue vivo (tiempo agotado):

IMPRIMIR "[Worker1] Tiempo agotado. Reenviando a Worker0..."

Interrumpir hilo de ordenamiento

Llamar a reenviarAWorker(vector, metodoOrdenamiento, tiempoLimite, "localhost", puertoDestino)

SINO (ordenamiento completado):

IMPRIMIR "[Worker1] Ordenamiento completado. Enviando al cliente..."

Llamar a enviarAlCliente(vector)

FIN procesarOrdenamiento

PROCEDIMIENTO reenviarAWorker(vector, metodoOrdenamiento, tiempoLimite, host, puerto):

Crear socket a "host" y "puerto"

Crear flujo de salida (ObjectOutputStream)

Enviar vector, metodoOrdenamiento, tiempoLimite y clienteIP

FIN reenviarAWorker

PROCEDIMIENTO enviarAlCliente(vector):

Crear socket a clienteIP en PUERTO_CLIENTE

Crear flujo de salida (ObjectOutputStream)

Enviar el vector al cliente

FIN enviarAlCliente

INICIO Cliente

Definir PUERTO_WORKER0 = 8080

Definir PUERTO_CLIENTE = 9090

Definir vector (Arreglo de enteros)

Definir metodoOrdenamiento (1=MergeSort, 2=QuickSort, 3=HeapSort)

Definir tiempoLimite (en segundos)

Definir clienteIP (IP del cliente)

IMPRIMIR "Conectando al Worker0 en puerto " + PUERTO_WORKER0

Crear socket y conectar a Worker0 en PUERTO_WORKER0

Crear flujo de salida (ObjectOutputStream) para enviar datos a Worker0

Crear flujo de entrada (ObjectInputStream) para recibir resultados

Enviar vector a Worker0

Enviar metodoOrdenamiento a Worker0

Enviar tiempoLimite a Worker0

Enviar clienteIP a Worker0

Esperar respuesta de Worker0

SI la respuesta es el vector ordenado:

  IMPRIMIR "Ordenamiento completado. Resultados recibidos."

SINO (en caso de reenvío a Worker1):

  IMPRIMIR "El tiempo se agotó. Recibiendo resultados de Worker1."

  Esperar respuesta de Worker1

```
  IMPRIMIR "Resultados finales: " + vectorOrdenado

FIN Cliente
INICIO Ordenamientos

PROCEDIMIENTO mergeSort(arr, left, right):
  SI left < right:
    Definir middle = (left + right) / 2
    Llamar a mergeSort(arr, left, middle)
    Llamar a mergeSort(arr, middle + 1, right)
    Llamar a merge(arr, left, middle, right)

FIN mergeSort

PROCEDIMIENTO merge(arr, left, middle, right):
  Definir n1 = middle - left + 1
  Definir n2 = right - middle

  Crear arreglo L de tamaño n1
  Crear arreglo R de tamaño n2

  Para i desde 0 hasta n1-1:
    L[i] = arr[left + i]

  Para j desde 0 hasta n2-1:
    R[j] = arr[middle + 1 + j]
```

```
  Definir i = 0, j = 0, k = left

  Mientras (i < n1 y j < n2):
    SI L[i] <= R[j]:
      arr[k] = L[i]
      Incrementar i
    SINO:
      arr[k] = R[j]
      Incrementar j
    Incrementar k

  Mientras (i < n1):
    arr[k] = L[i]
    Incrementar i
    Incrementar k

  Mientras (j < n2):
    arr[k] = R[j]
    Incrementar j
    Incrementar k

FIN merge

PROCEDIMIENTO quickSort(arr, low, high):
  SI low < high:
```

```
    Definir pi = partition(arr, low, high)
    Llamar a quickSort(arr, low, pi - 1)
    Llamar a quickSort(arr, pi + 1, high)

FIN quickSort

PROCEDIMIENTO partition(arr, low, high):
  Definir pivot = arr[high]
  Definir i = low - 1

  Para j desde low hasta high - 1:
    SI arr[j] < pivot:
      Incrementar i
      Intercambiar arr[i] con arr[j]

  Intercambiar arr[i + 1] con arr[high]
  Retornar i + 1

FIN partition

PROCEDIMIENTO heapSort(arr):
  Definir n = longitud de arr

  Para i desde n / 2 - 1 hasta 0 (índices de nodos internos):
    Llamar a heapify(arr, n, i)
```

```
  Para i desde n-1 hasta 1:
    Intercambiar arr[0] con arr[i]
    Llamar a heapify(arr, i, 0)

FIN heapSort

PROCEDIMIENTO heapify(arr, n, i):
  Definir largest = i
  Definir left = 2 * i + 1
  Definir right = 2 * i + 2

  SI left < n y arr[left] > arr[largest]:
    largest = left

  SI right < n y arr[right] > arr[largest]:
    largest = right

  SI largest != i:
    Intercambiar arr[i] con arr[largest]
    Llamar a heapify(arr, n, largest)

FIN heapify

FIN Ordenamientos
```