PROPOSITO: Programar los algoritmos de ordenación rápidos: Mergesort, Quicksort, Heapsort. Gráfica de comparación de los mismos.

Práctica de Algoritmos de ordenación 2

Sergio Casado López y Sandra Gómez Gálvez

Ingeniería del Software y Matemáticas

ÍNDICE

[Algoritmo Mergesort 2](#_Toc4708449)

[Algoritmo Quicksort 4](#_Toc4708450)

[Algoritmo Heapsort 5](#_Toc4708451)

[Gráfica de tiempos 7](#_Toc4708452)

# Algoritmo Mergesort

El algoritmo Mergesort ordena por mezcla.

El algoritmo es programado mediante el método Divide y Vencerás. Tendrá un caso base y casos recursivos.

Dónde dado el caso base devuelve el array, y si no:

1. crea una variable llamada centro con la longitud del array entre 2
2. En la variable a1 llama recursivamente a la función mergesort con un array de tamaño centro.
3. En la variable a2 llama recursivamente a la función mergesort con un array de tamaño que va de centro+1 a la longitud del array
4. Finalmente llama a la función merge(a1,a2) donde se mezclaran las variables.
5. Dentro a la función merge se mezclaran las variables de tal forma que se vayan introduciendo en el array r1 de menor a mayor orden.

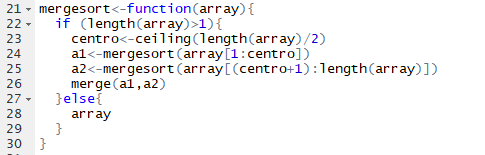


Ilustración 1 - Función mergesort()

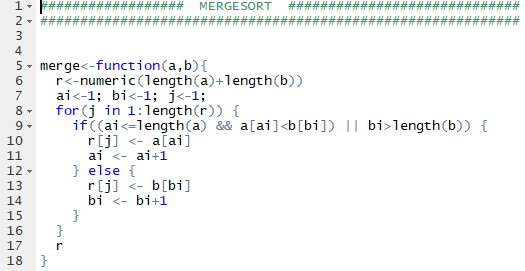


Ilustración 2 - Función merge()

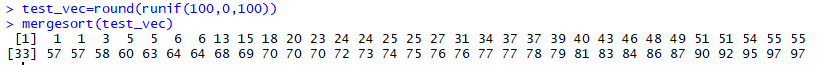
Introducimos un caso de prueba:



1 Prueba

Donde observamos que funciona.

Para un caso de prueba más grande:



2 Prueba

También funciona.

Por tanto, el algoritmo mergesort está ordenando correctamente.

# Algoritmo Quicksort

El algoritmo Quicksort es un algoritmo de ordenación rápida.

Ordena organizando sus elementos en 3 secciones: izquierda, Pivote y derecha.

La función quickSort funciona de la siguiente manera:

1. Elige un número aleatorio que será el pivote.
2. Crea 2 vectores izquierda y derecha.
3. Estos 2 vectores los irá rellenando de manera recursiva y así irá ordenando.
4. Finalmente concatena el vector izquierdo, el pivote y el derecho para crear el array final ordenado.

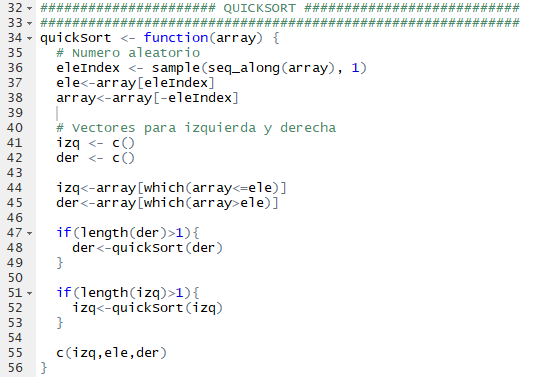


Ilustración 3- función quickSort()

Para probarlo:

Introducimos un caso de prueba:



3 Prueba

Donde observamos que funciona.

Para un caso de prueba más grande:



4 Prueba

También funciona.

Por tanto, el algoritmo quicksort está ordenando correctamente.

# Algoritmo Heapsort

El algoritmo Heapsort es un algoritmo de ordenación por heap.

Heapsort almacena los elementos del array en un heap y después extrae la cima hasta obtener el conjunto ordenado. Para ello, después de cada extracción, recoloca en la cima la última hoja por la derecha del último nivel (de un árbol). Continúa descendiendo el número insertado que se elige el mayor de sus dos hijos, y con él se intercambia. Al descender el nodo acaba hundido en el árbol. Este proceso se realiza sucesivamente hasta que quede completamente ordenado.

La función heapsort funciona de la siguiente manera:

1. Primero ordenamos el array en forma de árbol (la función modHeap modifica el árbol a partir de un nodo concreto)
2. Luego creamos un array ordenado vacío
3. Añadimos el primer elemento del árbol al ordenado, ponemos el último elemento del árbol en la primera posición y eliminamos la última posición.
4. Hacemos este proceso hasta que el árbol quede vacío, para finalmente nos resulte el array ordenado.

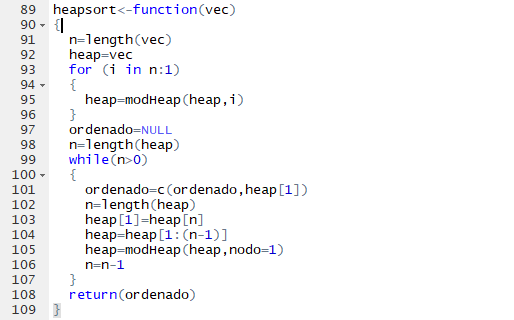


Ilustración 4 - función heapsort()

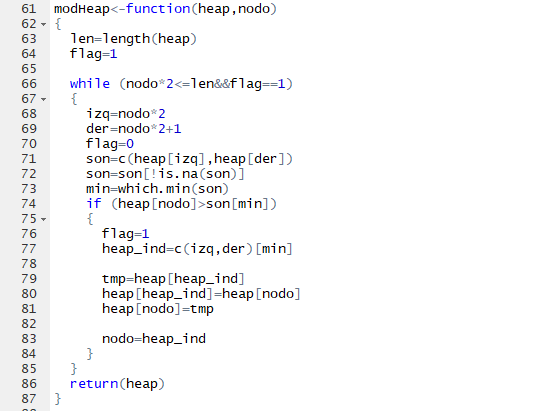


Ilustración 5 - funcion modHeap()

Para probarlo:

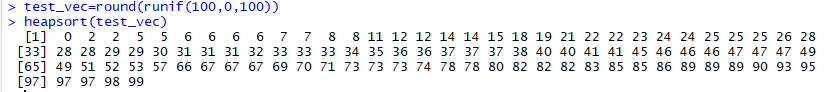
Introducimos un caso de prueba:



5 Prueba

Donde observamos que funciona.

Para un caso de prueba más grande:



6 Prueba

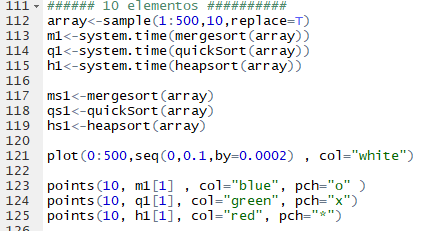
También funciona.

Por tanto, el algoritmo heapsort está ordenando correctamente.

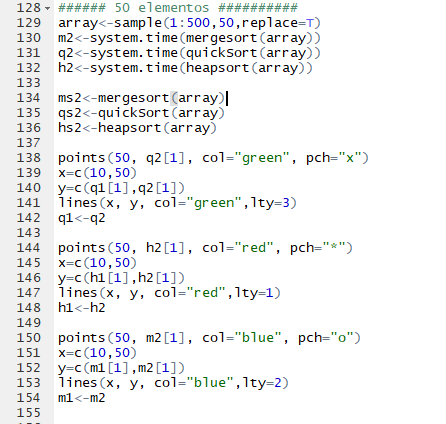
# Gráfica de tiempos

Crearemos una gráfica de tiempos comparando las diferentes funciones mergesort(), quickSort() , y heapsort().

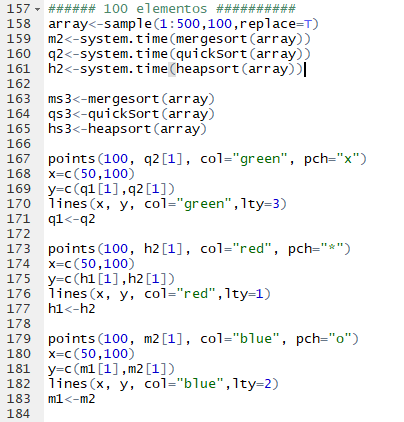
Cambiaremos los tamaños de los vectores a ordenar para así poder comparar los tiempos entre ambos algoritmos y saber cuál es más eficiente.

Con el siguiente código podemos dibujar una grafica para vectores de tamaño 10, 50, 100, 200, 300, 400, y 500. Primero generamos un vector de dicho tamaño con números aleatorios, calculamos su ordenación y medimos su tiempo. Vamos dibujando los tiempos en función del tamaño del vector para ambos algoritmos de ordenación, para poder ir observando la evolución y diferencia entre ambos.

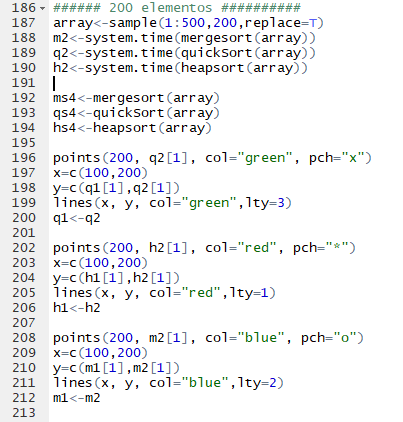
10 elementos



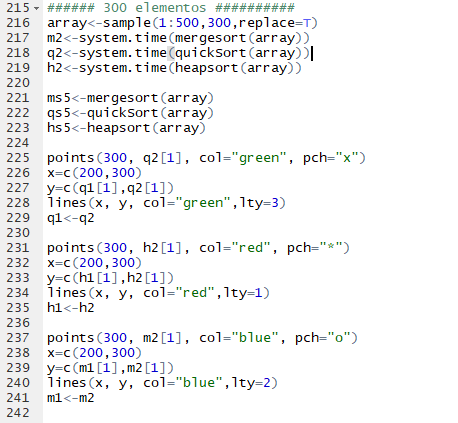
50 elementos



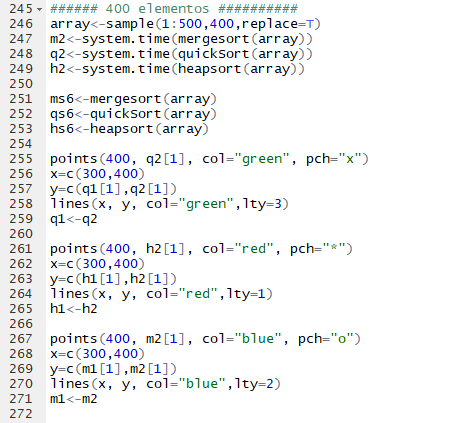
100 elementos



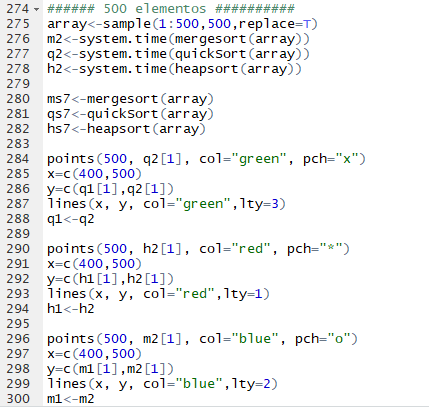
200 elementos



300 elementos

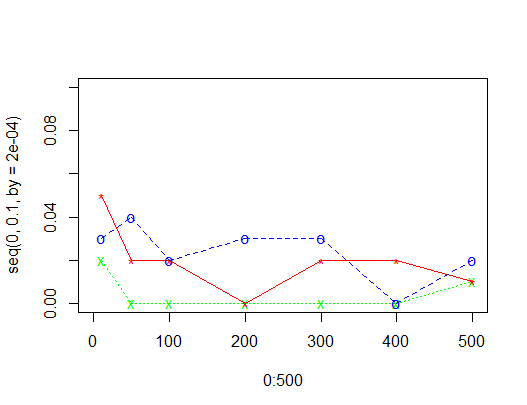


400 elementos



500 elementos

Donde nos muestra la gráfica:



Red: heapsort | Blue: mergesort | Green: quicksort

En el eje de las *x* encontraremos el número de elementos del array, y en el de las *y* el tiempo que tarda en ejecutarse el algoritmo.

Podemos observar como el algoritmo de la función mergesort() (azul) es el que se mantiene más tiempo en la parte superior por tanto es el qué más tarda de todos los algoritmos, el segundo más lento es el algoritmo de la función heapsort() (rojo), y el código más rápido es el código de la función Quicksort () (verde).