UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

Algoritmos y estructura de datos

Practica 10. Método de ordenamiento: Quicksort

Alumno: Caudillo Sánchez Diego

Matricula: 1249199

Grupo: 551

Docente: Alma Leticia Palacios Guerrero

Fecha de entrega: 24/Mayo/2019

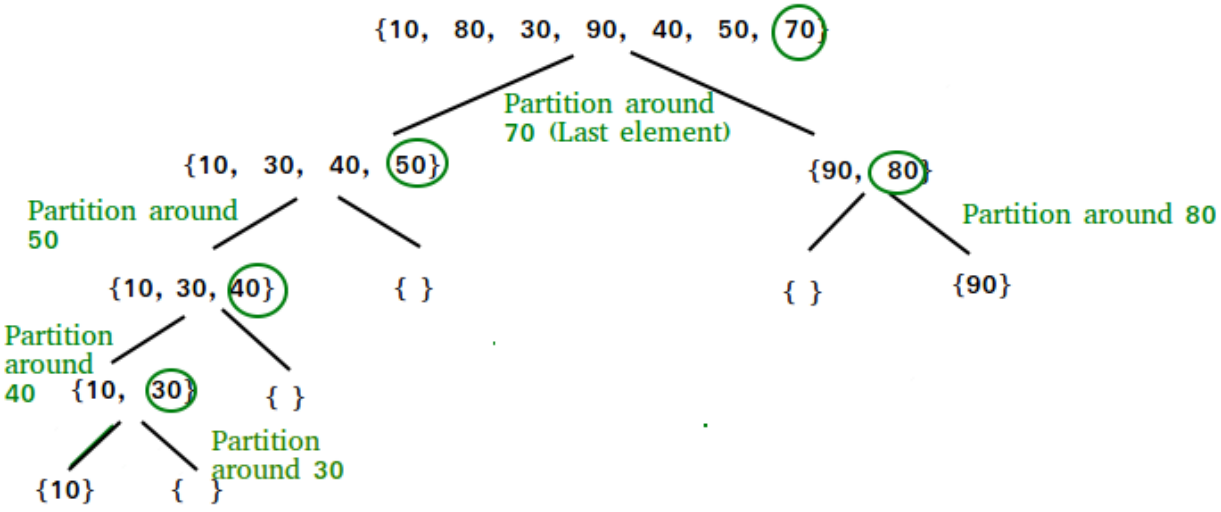
Introducción

Es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo proporcional a n log n.

El algoritmo consta de los siguientes pasos:

1. Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.
2. Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
3. La lista queda separada en dos *sublistas*, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
4. Repetir este proceso de forma recursiva para cada *sublista* mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

La eficiencia del algoritmo depende de la posición en la que termine el pivote elegido. En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos *sublistas* de igual tamaño. En este caso, el orden de complejidad del algoritmo es O(n•log n). En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de $O(n^2)$. El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas. Pero principalmente depende del pivote, si por ejemplo el algoritmo implementado toma como pivote siempre el primer elemento del array, y el array que le pasamos está ordenado, siempre va a generar a su izquierda un array vacío, lo que es ineficiente. En el caso promedio, el orden es O(n•log n). Y no es extraño, pues, que la mayoría de optimizaciones que se aplican al algoritmo se centren en la elección del pivote.



Competencia
Determinar la eficiencia de un algoritmo de ordenación según su desempeño en escenarios de prueba con distintos parámetros, para ser considerada como criterio en la selección de un algoritmo para resolver un problema.

Problema
Suponga que los datos de los estudiantes de una escuela se almacenan en una estructura que tiene los siguientes campos:

- Nombre
- Carrera
- Promedio
- Créditos cursados

Utilizando un arreglo de 10 elementos la estructura mencionada, ordene en forma ascendente (a-z) los datos por nombre y elabore una tabla con los siguientes datos:
Para los pasos a-d utilice el pivote que prefiera.

a) El tiempo y la cantidad de iteraciones para el peor de los casos
b) El tiempo y la cantidad de iteraciones para el mejor de los casos
c) El tiempo y la cantidad de iteraciones para cualquier otro caso
d) ¿Qué pasa si todos los datos son iguales?
   En este caso el único dato que tiene importancia con respecto al tiempo, es el nombre ya que de este depende la rapidez con la que el programa ejecuta las instrucciones. Los demás datos como la carrera, el promedio o los créditos cursados no son de importancia ya que no se utilizan, mas que para fines informativos, esto a menos que se quiera ordenar por calificaciones o por carrera.

Peor de los casos	Mejor de los casos	Otro caso	Datos iguales
0 segundos	0 segundos	0 segundos	0 segundos

Utilizando el mismo conjunto de datos para los siguientes pasos determine los puntos e-h:

e) ¿Cantidad de pasadas si el pivote es el elemento de la izquierda?
f) ¿Cantidad de pasadas si el pivote es el elemento de la derecha?
g) ¿Cantidad de pasadas si el pivote es el elemento del centro del arreglo?
h) ¿Cantidad de pasadas si el pivote es un elemento aleatorio?

Pivote	Cantidad	Tiempo
Izquierda	8	0.006 segundos
Derecha	6	0.001 segundos
Centro	6	0.002 segundos
Aleatorio	5	0.001

Repita los pasos e-h con otro conjunto de datos y responda.

¿La selección del pivote afectó de alguna manera la cantidad de pasadas que hace el algoritmo?

El pivote es importante saber elegirlo, la mayor parte del tiempo se recomienda que sea el ultimo elemento del arreglo para que tenga mayor eficacia y se pudo comprobar que es cierto. Pues que cuando se cambiaba de pivote el tiempo y la cantidad de iteraciones aumentaba.

¿Hubo cambios al utilizar otros datos o se comportó igual que con el primero?

Si hubo cambios, y fueron desfavorables. El algoritmo no es optimo cuando el pivote no es el último elemento del arreglo.

¿Cuál forma de elegir el pivote resultó más eficiente?

Que se inicie con el último elemento del arreglo resultó ser óptimo.

Código

```c
/**
 * Practica 10. Método de ordenamiento Quicksort
 * Alumno: Caudillo Sánchez
 * Grupo: 551
 * Asignatura: Algoritmos y estructura de datos
 * Docente: Alma Leticia Guerrero Palacios.

 * Descripción de la practica
    Utilizando un arreglo de 10 elementos la estructura mencionada, ordene en forma ascendente (a-z) los
    datos por nombre.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

typedef struct
{
    char name[15] ;
    char carrera[20];
    float promedio;
    int creditos;
}alumno_t;

//Funciones prototipo
void menu(alumno_t lista[10]);
void fillList(alumno_t lista[10]);
void quickSort(alumno_t lista[10], int inicio, int final);
int partition (alumno_t lista[10], int inicio, int final);
void swap(alumno_t *alumno, int i, int j);
void printData(alumno_t lista[10], int index);

int main(int argc, char const *argv[])
{
    alumno_t lista[10]; //declaración de una lista de 10 alumnos con sus atributos.
    menu(lista);
    return 0;
}

/*
```

```
    DESCRIPCION:
    PARAMETROS
*/
void menu(alumno_t lista[10])
{
    int inicio = 0, index = 9, opc;
    do
    {
        system("cls");
        puts("[1] Llenar lista");
        puts("[2] Ordernar datos ascendetemente");
        puts("[3] Mostrar datos");
        puts("[4] Salir");
        printf("Opci%cn: ",162); scanf("%d", &opc);
        setbuf(stdin, NULL);
        switch (opc)
        {
        case 1: fillList(lista);
            break;
        case 2: t = clock();

            quickSort(lista, inicio, index-1);
            t = clock() - t;
            double time_taken = ((double)t)/CLOCKS_PER_SEC;
            printf("quickSort() took %lf seconds to execute \n", time_taken);
        case 3: printData(lista, index);
            getchar();
            break;
        }
    } while (opc != 4);
}
/*
    DESCRIPCION: función que llena una lista con los datos de un alumno.
    PARAMETROS
       -> alumno: arreglo del tipo alumno_t el cual contiene espacio para
       10 alumnos.
*/
void fillList(alumno_t alummo[10])
{
    srand(time(NULL));

    char name[10][15] = {"Miguel","Eduardo",
    "Luis","Carlos","Ana","Daniel",
    "David","Jorge","Ricardo", "Alejandro"};
    char carrera[] = "computacion";
```

```
    for (int i = 0; i < 9; i++)
    {
       strcpy(alummo[i].name, name[i]); //copia el nombre del listado
       strcpy(alummo[i].carrera, carrera);
       //rand() % (max_number + 1 - minimum_number) + minimum_number
       alummo[i].promedio = (float)(rand() % (100+1 - 60) + 60);
       alummo[i].creditos = (rand() % (280+1 - 100) + 100);
    }
}
/*
    DESCRIPCION:
        Función de ordenamiento, el cual ordena una lista de
        alumnos por nombre de manera ascendente. El ordena-
        miento lo maneja de manera recursiva.

    PARAMETROS
        -> lista: arreglo del tipo alumno_t que contiene
        la información de 10 alumnos y de la cual se
        -> inicio: indica la posición inicial del arreglo
        -> final: indica la posición final del arreglo.
*/
void quickSort(alumno_t lista[10], int inicio, int final)
{
    if (inicio < final)
    {
        int index = partition(lista, inicio, final);
        quickSort(lista, inicio, index-1);
        quickSort(lista, index+1, final);
    }
}

/*
    DESCRIPCION:
        Esta función toma el último elemento como pivote,
        colocándolo en la posición correcta en el arreglo
        y posición todos los elementos menores a la
        izquierda del pivote y los mayores a la derecha.

    PARAMETROS
        -> lista: arreglo del tipo alumno_t que contiene
        la información de 10 alumnos y de la cual se
        va acomodar por nombres de manera ascendente.
        -> inicio: indica la posición inicial del arreglo
        -> final: indica la posición final del arreglo.
*/
```

```c
int partition (alumno_t lista[10], int inicio, int final)
{
    alumno_t *pivote = &(lista[final]);
    int i =  inicio - 1;
    for (int j = inicio; j <= final - 1; j++)
    {
        if(strcmp(lista[j].name, pivote->name) <= 0)
        {
            i++;
            swap(lista, i,j);
        }
    }swap(lista, i+1, final);
    return(i+1);
}


/*
    DESCRIPCION:
        Función que intercambia el contenido de dos estructuras
        del tipo alumno_t. La función no retorna un valor.

    PARAMETROS
        -> alumno: arreglo del tipo alumno_t de la cual se
        elegirá las posiciones a cambiar dadas por i y j.
        -> i: primer índice de valor a intercambiar
        -> j: segundo índice de valor a intercambiar
*/
void swap(alumno_t alumno[10], int i, int j)
{
    alumno_t tmp = alumno[i]; // se guarda en la posición i
    alumno[i] = alumno[j];  // intercambio del primer índice
    alumno[j] = tmp; // intercambio del segundo índice
}

/*
    DESCRIPCION:
        Función que imprime los datos de una lista de alumnos.
        Los datos que imprime son los siguientes:
        -> Nombre del alumno
        -> Carrera a la que pertenece
        -> Promedio del alumno
        -> Créditos cursados
        La función no retorna un valor.

    PARAMETROS
```

```
        -> lista: arreglo del tipo alumno_t que contiene los datos de los
alumnos.
        -> index: cantidad de alumnos en la lista. Es un índice que indica
cuantos
        alumnos va a mostrar en pantalla.
*/
void printData(alumno_t lista[10], int index)
{
    printf("Nombre\t Carrera\tPromedio\tCr%cditos\n\n",130);
    for(int i = 0; i < index; i++ )
    {
        printf("%s\t" ,lista[i].name);
        printf(" %s\t" ,lista[i].carrera);
        printf("  %.2f\t" ,lista[i].promedio);
        printf("\t%d\n", lista[i].creditos);
    }
}
```

Evidencia de ejecución

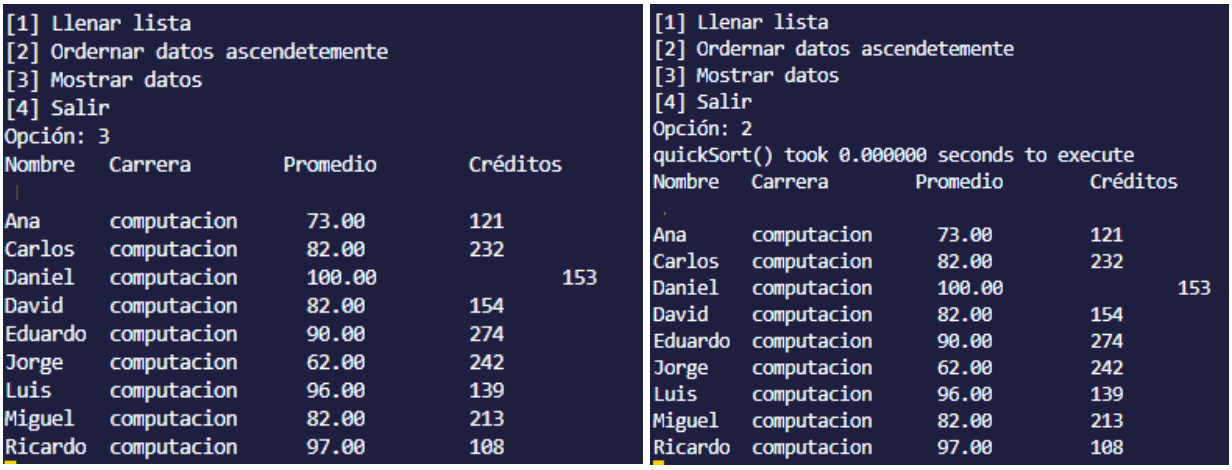
```
[1] Llenar lista
[2] Ordernar datos ascendetemente
[3] Mostrar datos
[4] Salir
Opción: 3
Nombre   Carrera        Promedio        Créditos

Miguel   computacion      61.00         263
Eduardo  computacion      80.00         141
Luis     computacion      78.00         150
Carlos   computacion      78.00         192
Ana      computacion      65.00         209
Daniel   computacion      97.00         225
David    computacion      76.00         145
Jorge    computacion      67.00         259
Ricardo  computacion      98.00         190
```

```
[1] Llenar lista
[2] Ordernar datos ascendetemente
[3] Mostrar datos
[4] Salir
Opción: 2
quickSort() took 0.000000 seconds to execute
Nombre   Carrera        Promedio        Créditos

Ana      computacion      65.00         209
Carlos   computacion      78.00         192
Daniel   computacion      97.00         225
David    computacion      76.00         145
Eduardo  computacion      80.00         141
Jorge    computacion      67.00         259
Luis     computacion      78.00         150
Miguel   computacion      61.00         263
Ricardo  computacion      98.00         190
```

Datos antes y despues de ordenarlos (datos desornedos)



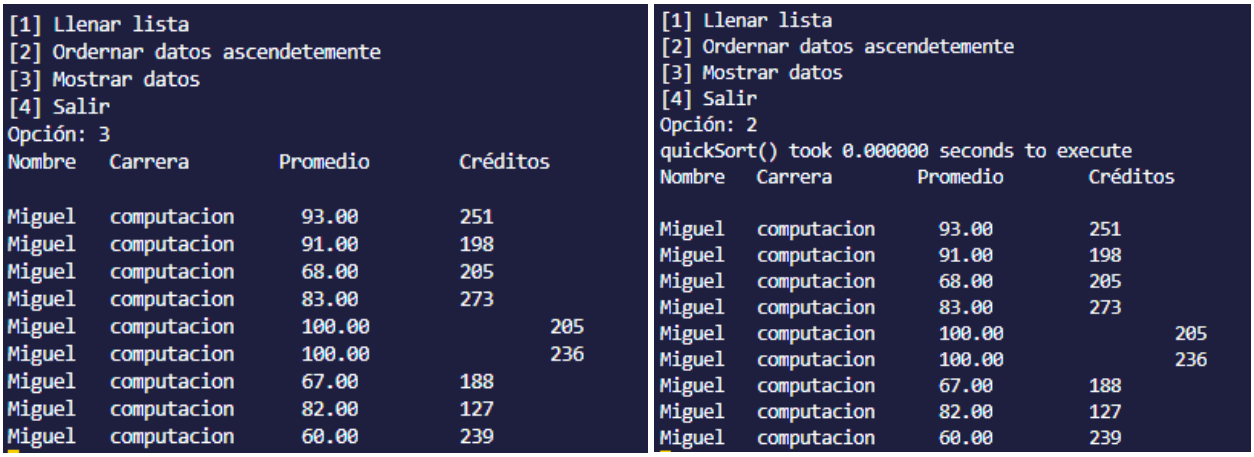
```
[1] Llenar lista
[2] Ordernar datos ascendetemente
[3] Mostrar datos
[4] Salir
Opción: 3
Nombre   Carrera        Promedio        Créditos

Ana      computacion      73.00         121
Carlos   computacion      82.00         232
Daniel   computacion      100.00                153
David    computacion      82.00         154
Eduardo  computacion      90.00         274
Jorge    computacion      62.00         242
Luis     computacion      96.00         139
Miguel   computacion      82.00         213
Ricardo  computacion      97.00         108
```

```
[1] Llenar lista
[2] Ordernar datos ascendetemente
[3] Mostrar datos
[4] Salir
Opción: 2
quickSort() took 0.000000 seconds to execute
Nombre   Carrera        Promedio        Créditos

Ana      computacion      73.00         121
Carlos   computacion      82.00         232
Daniel   computacion      100.00                153
David    computacion      82.00         154
Eduardo  computacion      90.00         274
Jorge    computacion      62.00         242
Luis     computacion      96.00         139
Miguel   computacion      82.00         213
Ricardo  computacion      97.00         108
```

Datos antes y despues de ordenarlos (datos previamente ordenados)



```
[1] Llenar lista
[2] Ordernar datos ascendetemente
[3] Mostrar datos
[4] Salir
Opción: 3
Nombre   Carrera        Promedio        Créditos

Miguel   computacion      93.00         251
Miguel   computacion      91.00         198
Miguel   computacion      68.00         205
Miguel   computacion      83.00         273
Miguel   computacion      100.00                205
Miguel   computacion      100.00                236
Miguel   computacion      67.00         188
Miguel   computacion      82.00         127
Miguel   computacion      60.00         239
```

```
[1] Llenar lista
[2] Ordernar datos ascendetemente
[3] Mostrar datos
[4] Salir
Opción: 2
quickSort() took 0.000000 seconds to execute
Nombre   Carrera        Promedio        Créditos

Miguel   computacion      93.00         251
Miguel   computacion      91.00         198
Miguel   computacion      68.00         205
Miguel   computacion      83.00         273
Miguel   computacion      100.00                205
Miguel   computacion      100.00                236
Miguel   computacion      67.00         188
Miguel   computacion      82.00         127
Miguel   computacion      60.00         239
```

Datos antes y despues de ordenarlos (datos iguales)

Conclusión

Con la realización de esta practica podemos concluir que los elementos que se elijan para un algoritmo de ordenación son de suma importancia, ya que entre mayor sea el arreglo y con datos desordenados el programa toma una cantidad de tiempo mayor. A lo que se pudo observar en este caso, es que el pivote juega un papel importante, ya que este mismo el que ayuda a controlar el numero menos posible de iteraciones y por tanto un tiempo menor en la ejecución.

Quicksort como algoritmo de ordenamiento es fácil de entender e implementar, solo que tiene la desventaja de que en arreglos de mayor cantidad no es practico su uso, por lo que se tiene que recurrir a otros métodos de ordenamiento.

Bibliografía

Joyanes L. & Zahonero I. (2008). Colas y pilas. *En Metodología, algoritmos y estructura de datos. (*641-649). Madrid: McGraw Hill.

Anónimo. (s.f). *How to measure time taken by a function in C?* mayo 24, 2019, de GeeksforGeeks Sitio web: https://www.geeksforgeeks.org/how-to-measure-time-taken-by-a-program-in-c/

Anónimo. (s.f). *Quicksort* mayo 24, 2019, de GeeksforGeeks Sitio web: https://www.geeksforgeeks.org/quick-sort/

Peña M., Ricardo. *Diseño de programas. Formalismo y abstracción.* Prentice-Hall, 1998.

Weiss, M. A., *Estructuras de datos y algoritmos.* Addison-Wesley Iberoamericana, 1995.

Wirth, Niklaus. *Algoritmos y Estructuras de Datos*. Pentice Hall, 1987.