Solución - práctico 10

Complejidad

Ejercicio 1

a) Una implementación del algoritmo es la siguiente:

```
/* Procedimiento que se encarga de realizar la unión de los dos conjuntos
 * ordenados der1 e izq1 son los límites del primer conjunto y der2 e izq2
 * los del segundo */
void Intercala (int *a, int *tmp, int izq1, int izq2, int der2)
{
    int der1 = izq2 - 1;
    int i = izq1;
    int numElementos = der2 - izq1 + 1;
    while ((izq1 <= der1) && ( izq2 <= der2))
    {
           if (a[izq1] <= a[izq2])
                  tmp[i] = a[izq1++];
           else
                  tmp[i] = a[izq2++];
           i++;
    }
    while (izq1 <= der1)
           tmp[i++] = a[izq1++];

    while (izq2 <= der2)
           tmp[i++] = a[izq2++];

    for (i = 1; i <= numElementos; i++)
    {
           a[der2] = tmp[der2];
           der2--;
    }
}

/* Procedimiento encargado de partir el conjunto inicial en cada paso de
 * la recursión */
void Intercal_Ord (int *a, int *tmp, int izq, int der)
{
    if (izq < der)
    {
           int centro = (izq + der) / 2;
           Intercal_Ord (a, tmp, izq, centro);
           Intercal_Ord (a, tmp, centro + 1, der);
           Intercala (a, tmp, izq, centro + 1, der);
    }
}
```

```
/* Programa principal, recibe un vector a con los datos y su tope
 * el arreglo a ordenar va desde 1 hasta tope, la casilla 0 es auxiliar */
void Merge_Sort (int *a, int tope)
{
    int *tmp = new int[tope+1];
    Intercal_Ord (a, tmp, 1, tope);
    delete [] tmp;
}
```

b) Dado que el tamaño del vector de datos es potencia de 2, se puede iniciar el análisis de este algoritmo para el peor caso de la siguiente manera:

$$T(n) = \begin{cases} 2.T(n/2) + n - 1 & \text{si n} > 1 \\ 0 & \text{si n} = 1 \end{cases}$$

Primer ecuación: para ordenar un arreglo de n elementos utilizando *MergeSort* se tiene que partir éste en dos mitades iguales y ordenarlas (primer término de la ecuación) y luego se deben intercalar esas dos mitades, lo cual es lineal en el largo del arreglo (segundo término).

Segunda ecuación: ordenar un vector de tamaño uno lleva un tiempo constante e igual a cero.

Recordar que: $\sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1}$

Desarrollando con n = 2^k se tiene:

$$T(2^k) = 2T(2^k/2) + 2^k - 1$$
$$2T(2^{k-1}) = 2^2 T(2^{k-1}/2) + 2(2^{k-1} - 1)$$
.
.
.
$$2^i T(2^{k-i}) = 2^{i+1} T(2^{k-i}/2) + 2^i (2^{k-i} - 1)$$
.
.
.
$$2^{k-1} T(2^1) = 2^k T(2^1/2) + 2^{k-1}(2^1 - 1)$$
$$2^k T(2^0) = 2^k T(1) = 0$$

Sumando:

$2T(2^1) = \sum_{i=0}^{k-1}(2^k - 2^i) = k.2^k - 2^k + 1$, y como $k = \log_2(n)$

Resulta:

$$T(n) = \begin{cases} n.\log(n) - n + 1 & \text{si n} > 1 \\ 0 & \text{si n} = 1 \end{cases}$$

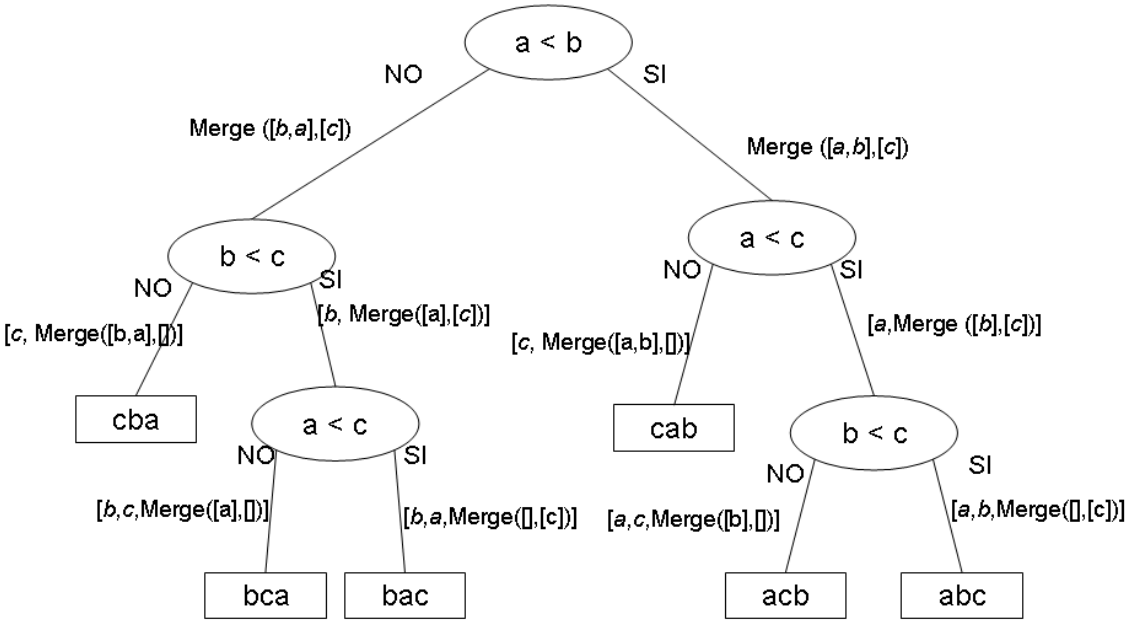
Al considerar el comportamiento asintótico, el término n+1 pierde relevancia y obtenemos: $T(n) = n.\log(n)$

Las comparaciones en el algoritmo de MergeSort se llevan a cabo en el procedimiento Merge, que toma dos listas ordenadas y las combina en una tercera.

c) En este ejercicio se pide la implementación del algoritmo *MergeSort*, para resolver este algoritmo se utiliza la técnica de diseño de algoritmos: *Divide & Conquer*, más específicamente la técnica es denominada *Easy to split – Hard to join* ya que la dificultad algorítmica radica en la unión de las soluciones intermedias.

d) Tomando la entrada de largo 3, sea a = [*a,b,c*] la entrada del algoritmo MergeSort, Dado que el algoritmo divide la secuencia original en las sub secuencias a_1 = [*a,b*]; a_2 = [*c*], la primer comparación se lleva a cabo en la lista a_1.

A continuación se muestra la figura resultante del árbol de decisión para MergeSort con tres elementos.



Ejercicio 2

a) Una posible implementación del algoritmo es la siguiente:

```
void Swap(int &a, int&b){
    int tmp = a;
    a = b;
    b = tmp;
}

int elegirPivote(int *a, int izq, int der){
    int medio = (der + izq) / 2;

    if (a[izq] > a[medio])
           Swap (a[izq], a[medio]);
    if (a[izq] > a[der])
           Swap (a[izq], a[der]);
    if (a[medio] > a[der])
           Swap (a[medio], a[der]);
    return medio;
}

/* Procedimiento recursivo de ordenacion */
void Q_Sort (int *a, int izq, int der)
{
    if (izq + 1 < der)
    {
           int posPiv = elegirPivote(a, izq, der);
           int p = a[posPiv];
           Swap (a[posPiv], a[der - 1]);

           int i = izq + 1;
           int j = der - 2;
           do
           {
                  while (a[i] < p)
                         i++;
                  while (a[j] > p)
                         j--;
                  Swap (a[i], a[j]);
           }
           while (i < j);

           Swap (a[i], a[j]);
           Swap (a[i], a[der - 1]); // Restaura el Pivote
           Q_Sort (a, izq, i-1);
           Q_Sort (a, i+1, der);
    }
    else if (a[izq] > a[der])
           Swap (a[izq], a[der]);
}

/* Programa principal, recibe el vector a con los datos y su tope
 * el arreglo a ordenar va desde 1 hasta tope, la casilla 0 es auxiliar */
void Quick_Sort (int *a, int tope)
{
    Q_Sort (a, 1, tope);
}
```

Esta implementación selecciona el pivote calculando la mediana entre el primer elemento, el último y el que se encuentra en el medio del vector en cada paso de la recursión. Esta forma de calcular el pivote es la que ha dado mejores resultados en la práctica. Además, permite eliminar ciertos casos de borde en la solución.

b) En este ejercicio se pide la implementación del algoritmo *QuickSort*, este algoritmo utiliza la técnica de diseño: *Divide & Conquer*, más específicamente la técnica se denomina *Hard to split – Easy to join* ya que la dificultad algorítmica radica en la división del problema a resolver.

Anexo

Se adjunta un programa principal para probar el funcionamiento de los algoritmos de ordenación:

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <limits.h>
#include <time.h>
#include <math.h>

int main ()
{
	const float MAX = 20000;
	const int N = 10000;
	int a[N + 1], i, valor = INT_MIN;

	srand (time(0));
	// asigna la semilla para la funcion que obtiene números aleatorios

	for (i = 1; i <= N; i++){
		a[i] = (int) ( (((double)rand() )/RAND_MAX) * MAX + 1.0);
		/*
		 * se asigna a la posición i del vector a un valor aleatorio
		 * entre 1 y MAX.
		 */
		cout << a[i];
	}
	cout << endl << endl;

	Algoritmo_Sort(a, N);

	for (i = 1; i <= N; i++){
		assert (valor <= a[i]);
		valor = a[i];
		cout << a[i];
	}
	cout << endl << endl;

	return 0;
}
```