Estructura de Datos y Algoritmos
SOLUCIONES del EXAMEN del 21/12/2010

Ejercicio 1)

Orden del tiempo de ejecución:

```
sum=0;
for (i=1;i<n;i++)
    for (j=0;j<n-1;j++)
        sum++;
```

$\Sigma_{i=1}^{n-1}\Sigma_{0}^{n-2}1 =$
$\Sigma_{i=1}^{n-1}(n-1) =$
$n\Sigma_{i=1}^{n-1}1 - \Sigma_{i=1}^{n-1}1 =$
$n.(n-1) - n + 1 \Longrightarrow O(n^2)$

```
sum=0;
for (i=1;i<n;i++)
    for (j=0;j<i;j++)
        sum++;
```

$\Sigma_{i=1}^{n-1}\Sigma_{0}^{i-1}1 =$
$\Sigma_{i=1}^{n-1} i =$
$(n-1).n/2 \Longrightarrow O(n^2)$

Ejercicio 2)

```
void invierto (char a [],int n)
{    int i,j;
     char c;
     i = 0;
     j=n-1;
     while (i<j)
     {    c=a[i];
          a[i]=a[j];
          a[j]=c;
          i++;
          j - - ;
     }
}
```

Ejercicio 3)

```
//Estructura de datos
struct nodo {
       int valor;
       nodo * sig;
};
typedef nodo * Lista;

//La siguiente es una variante posible de implementacion, puede haber otras.
//La idea es recorrer la lista parametro y para cada valor almacenado en un nodo
//llamo a eliminar ese valor del resto de la lista.

void PURGE(Lista & l){
     //En esta implementacion se elimina los duplicados modificando la lista parametro.
    Lista l_act = l;
    int val;
    while (l_act != NULL){
       val = l_act -> valor;
       eliminar(val,l_act->sig);
       //elimino del resto de la lista, el valor que se encuentra en el nodo actual
       l_act = l_act->sig;
    }
}


//Implementacion de la operacion eliminar:
//Se presentan dos posibles implementaciones una iterativa y otra recursiva.
//En el examen alcanza con implementar una de ellas.

//Implementacion iterativa

void eliminar(int valPrm, Lista & l){
   Lista l_act = l;
   Lista l_ant = NULL;

   while (l_act != NULL){
         if (l_act -> valor == valPrm){
                   if (l_ant==NULL){
                      //el que voy a eliminar es el primero de la lista
                      l = l->sig;
                      delete l_act;
                      l_act = l;
                   }
                   else {
                      //el que voy a eliminar NO es el primero de la lista
                      l_ant->sig = l_act->sig;
                      delete l_act;
                      l_act = l_ant->sig;
```

```
                    }
         }
         else {
               l_ant=l_act;
               l_act = l_act -> sig;
         }
   }//while
}


//Implementacion recursiva

void eliminarRec(int valPrm, Lista & l){
     Lista l_aux = NULL;

     if (l!=NULL){
        if (l->valor == valPrm){
              l_aux = l;

              //como la lista se pasa por referencia no necesito tener un puntero al nodo
              //anterior al que voy a eliminar, al saltear el nodo a eliminar la lista
              //sigue quedando correctamente encadenada
              l=l->sig;

              delete l_aux;
              eliminarRec(valPrm,l);
        }
        else eliminarRec(valPrm,l->sig);
     }
}
```

Ejercicio 4)

Parte a)

```
struct nodo{
       nodo *izq;
       nodo *der;
       char car;
       int prio;//cantidad de veces que el caracter car aparece en el texto
};
typedef nodo * ABBC;
```

Parte b)

```
#define MAX_ELEMS 10000 //10000 es un valor a modo de ejemplo

typedef struct arrTope {
       int valores[MAX_ELEMS];
       int tope;//cantidad de elementos en el arreglo
} ARR_INT;


void recorrerPostOrden(ABBC arbol, ARR_INT & arr_par, ARR_INT & arr_impar){

     if (arbol!=NULL){
        recorrerPostOrden(arbol->izq,arr_par,arr_impar);
        recorrerPostOrden(arbol->der,arr_par,arr_impar);

        if ((arbol->prio%2)==0){
           //tiene prioridad par
           arr_par.valores[arr_par.tope]=arbol->car;
           arr_par.tope++;
        }
        else {
           //tiene prioridad impar
           arr_impar.valores[arr_impar.tope]=arbol->car;
           arr_impar.tope++;
        }

     }
}
```

Parte c)

```
//Extraido del teorico 6, punto 1.2
//Comenzamos definiendo una funcion hallo pivote. Si hallo pivote no encuentra
//dos valores distintos retorna 0, sino retorna el indice del mayor de los
//primeros dos elementos distintos.
```

```
int hallo_pivote(int A [],int i,int j){
    int clave1,k;
    clave1=A[i];
    for (k=i+1;k<=j;k++){
        if (A[k]>clave1)
           return k;
        else if (A[k]<clave1)
                return i;
    }
    return 0;
}

void intercambio (int A[],int i,int j){
     int tmp = A[i];
     A[i] = A[j];
     A[j] = tmp;
}

//Recorro el arreglo desde la izquierda y desde la derecha al mismo tiempo
//dejando a la izquierda los elementos menores que el pivote
//y a la derecha los mayores que el pivote, realizando los intercambios necesarios
int particion (int A[],int i,int j,int pivote){
    int izq,der;
    izq=i;
    der=j;
    do
    {
      intercambio(A,izq,der);
      while(A[izq] < pivote) izq++;
      while(A[der] >= pivote) der--;
    }
    while (izq <= der);
    return izq;
}
//el intercambio inicial no interesa ya que no asumimos ningun orden en la
//entrada al comienzo del programa.

//El programa final es el siguiente:
void sort_rapido (ARR_INT A, int i, int j){
     int pivote,indice,k;
     indice=hallo_pivote(A.valores,i,j);
     if (indice!=0){
        pivote=A.valores[indice];
        k=particion(A.valores,i,j,pivote);
        sort_rapido(A,i,k-1);
        sort_rapido(A,k,j);
     }
}
```