Pregunta 1

La siguiente es una solución incorrecta de un sistema para ordenar pizzas:

```
typedef struct {
  Ingredientes *ingr;
  Pizza *pizza;
  int fin;
} Orden;

Cola *cola;
pthread_t operador;

void initHorno() {
  cola= nuevaCola();
  pthread_create(&operador,
       NULL, pizzeria, NULL);
}

Orden *ordenar(Ingredientes
                 *ingr) {
  Orden *o= malloc(sizeof(*o));
  o->ingr= ingr;
  o->fin= 0;
  agregar(cola, o);
  return o;
}
```

```
void *pizzeria(void *ptr) {
 for (;;) {
   Pizza *pizzas[4];
   Orden *ords[4];
   while (vacia(cola))
     ;
   int i= 0, j;
   while (!vacia(cola) && i<4)
   {
     ords[i]= extraer(cola);
     pizzas[i]= pizzaCruda(
                  ords[i]->ingr);
     i++;
   }
   hornear(pizzas); /* lento */
   for (j= 0; j<i; j++) {
     ords[i]->fin= 1;
     ords[i]->pizza= pizzas[i];
}} }
Pizza *retirar(Orden *o) {
  while (!o->fin)
    ;
  return o->pizza;
}
```

Múltiples clientes representados por threads ordenan sus pizzas con los ingredientes deseados llamando a la función *ordenar*. La pizzería tiene un horno que es capaz de preparar hasta un máximo de 4 pizzas a la vez. Para ello llama a la función *hornear* (toma un tiempo considerable). En algún momento los clientes pasan a retirar su pizza llamando a *retirar*, pero deben esperar si la pizza no está aún lista.

(a) ¿Por qué esta solución puede entregar resultados incorrectos? ¿Por qué es ineficiente? (1 punto)

(b) Reprograme esta solución de manera correcta y eficiente. (3 puntos)

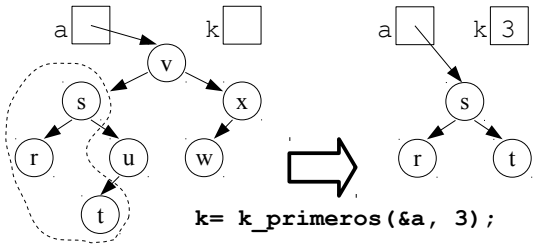
(c) Modifique la solución agregando la siguiente función: (2 puntos)

```
Pizza *ordenar_urgente(Ingredientes *ingr)
```

Esta función solicita una pizza urgentemente y espera hasta que esté terminada. Las pizzas urgentes tienen prioridad para ingresar al horno. *Hint:* Use una segunda cola para las pizzas urgentes. También deberá modificar las funciones *initHorno* y *pizzeria*.

Pregunta 2

i. Programe la función: `int k_primeros(Nodo **pa, int k)`
Esta función recibe un *árbol de búsqueda binaria* en ***pa* y lo poda dejando solo los primeros *k* nodos. El árbol resultante queda en ***pa* y debe ser un árbol de búsqueda binaria. Si el árbol tiene menos de *k* nodos, entonces no se modifica. La función retorna el número efectivo de nodos del árbol resultante. No cree nuevos nodos, reutilice los nodos existentes, no libere los nodos descartados. Ejemplo:



Hint: complete esta función

```
int k_primeros(Nodo **pa,
                 int k) {
  Nodo *a= *pa;
  if (k==0) {
    *pa= ...
  }
  else if (a==NULL) {
    k= ...
  }
  else {
    int kizq= k_primeros(
                &a->izq, k);
    if (kizq==k) {
      *pa= ...
    }
    else {
  ... etc. ...
```

ii. Programe la función: *`char *reemplazar(char *s, char c, char *pal)`*
Esta función entrega un nuevo string resultante de reemplazar en *s* todas las ocurrencias del carácter *c* por la palabra *pal*. Ejemplo:

```
char *res= reemplazar("hola que tal", 'a', "xyz");
```

El string resultante es "hol**xyz** que t**xyz**l".

Pregunta 3

A) Programe la siguiente función:

`unsigned int recortar(unsigned int x, int i, int j)`

Considere que *x* esta formado por los bits $x_{31}\ x_{30}\ ...\ x_1\ x_0$. La función *recortar* debe entregar el resultado de recortar los bits $x_i\ ...\ x_j$ de x. No puede usar los operadores de multiplicación, división o módulo. Use los operadores de bits. Ejemplo:

```
unsigned v= recortar(0x006af52c, 19, 8); /* queda: 0x0000062c */
```

B) Programe la siguiente función para una máquina dual-core:

```
int buscarx2(char **a, int n, char *s)
```

Esta función recibe un arreglo de strings *a*, su tamaño *n* (muy grande) y un string *s*. Debe retornar verdadero si el string *s* es igual a uno de los strings contenidos en el arreglo, o falso en caso contrario. ¡Cuidado! el arreglo no está ordenado. Para paralelizar Ud. debe usar 2 llamadas a *fork*. Si uno de los subprocesos encuentra el string buscado, entonces debe matar el otro subproceso con la señal SIGTERM (no espere a que termine).