Guía 6

Rodrigo Estay Muñoz

Problema 1:

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// Codigo hecho solo.

int esPalindromo(char* palabra, int tam);

// main es para probar el funcionamiento de la funcion.
int main(){
    char pal[10000];
    scanf("%s", pal);
    printf("%s.\n", (esPalindromo(pal,strlen(pal)))?"Si":"No");
    return 0;
}

// Comparamos el primer elemento de la palabra con el ultimo, si son iguales puede tratarse
// de un palindromo, asi que quitamos el primer y ultimo elemento de la palabra y lo mandamos
// a la funcion. Asi sigue hasta que el tamaño de la funcion sea de dos o uno, o que no se
// cumpla que es palindromo.
int esPalindromo(char* palabra, int tam){
    if(tam==1){
        return 1;
    }
    if(palabra[0]!=palabra[tam-1]){ // Si estos elementos son distintos, no es palindromo.
        return 0;
    }
    // Como este else depende del if anterior, solo se ejecutara cuando queden dos elementos
    // que son iguales.
    else if(tam==2){
        return 1;
    }
    char temp[tam-1];
    strcpy(temp, (palabra+1)); // Eliminamos primera letra.
    temp[tam-2]='\0'; // Eliminamos la ultima letra.
    esPalindromo(temp,tam-2);
}
```

problema 2:

```c
#include <stdlib.h>
#include <stdio.h>

// Programa hecho sin ayuda.

int elementoMayorRecursivo(int* lista, int tam);

// main para probar la funcion.
int main(){
    int n;
    scanf("%d", &n);
    int i, lista[n];
    for(i=0;i<n;++i){
        scanf("%d", &lista[i]);
    }
    printf("%d\n", elementoMayorRecursivo(lista, n));
    return 0;
}

```

```
//  Simplemente comparamos el elemento mayor de la cola con el de la cabeza, entonces mandamos la
// cola como otra lista a la misma funcion, se sigue de esta manera hasta que hayan 2 elementos o
// menos, si son 2, se retorna el mayor de estos, si es uno, significa que el tamaño inicial de la
// lista era de 1, asi que retornamos el unico elemento.
int elementoMayorRecursivo(int* lista, int tam){
    if(tam<=2){
        if(tam==1){
            return *lista;
        }
        if(*lista>*(lista+1)){
            return *lista;
        }
        else{
            return *(lista+1);
        }
    }
    int mayorCola=elementoMayorRecursivo(lista+1, tam-1);
    if(mayorCola>*lista){
        return mayorCola;
    }
    else{
        return *lista;
    }
}
```

Problema 3:

```
#include <stdlib.h>
#include <stdio.h>

// Codigo hecho sin ayuda.

int busquedaBinaria(int* v, int elem, int limInferior, int limSuperior);

// Main para probar la funcion.
int main(){
    int n;
    scanf("%d", &n);
    int v[n], i, lim, e;
    for(i=0;i<n;++i){
        scanf("%d", &v[i]);
    }
    scanf("%d%d", &lim, &e);
    printf("%d\n", busquedaBinaria(v, e, lim, n));
    return 0;
}

//  Simplemente avanzamos por el vector desde el limite inferior, hasta que el limite
// inferior sea igual al limite superior, si se llega a esta situacion significa que
// no existe el elemento pedido en el rango dado.
//  Cabe notar que los limites y las posiciones de los elementos se toman como que empiezan
// desde el 1, no desde el 0, es decir que el primer elemento esta en la posicion 1 (no 0).
int busquedaBinaria(int* v, int elem, int limInferior, int limSuperior){
    if(limInferior==limSuperior){
        return -1;
    }
    if(*(v+limInferior-1)==elem){
        return limInferior;
    }
    return busquedaBinaria(v, elem, limInferior+1, limSuperior);
}
```

Problema 4:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Programa hecho sin ayuda

void mergeSort(int *v, int tam);
void merge(int* v1, int* v2, int tam1, int tam2);

//  Este main es para probar las funciones, lo que se hace es que se le pide al usuario un numero n, luego
// se generan n vectores de tamanos aleatorios entre 1 y 50, luego se llenan con valores aleatorios entre
// 0 y 149, se imprime el vector, luego se ordena con la funcio mergeSort, una vez ordenado se vuelve a
// imprimir el vector, pero si es que se encuentra que no esta ordenado de menor a mayor se imprime un grito.
int main(){
    int n, i, tam;
    srand(time(NULL));
    scanf("%d", &n);
    while(n--){
        tam=rand()%50+1;
        int *arr=(int*)malloc(tam*sizeof(int));
        for(i=0;i<tam;++i){
            *(arr+i)=rand()%150;
        }
        for(i=0;i<tam;++i){
            printf("%d ", *(arr+i));
        }
        printf("\n");
        mergeSort(arr, tam);
        for(i=0;i<tam;++i){
            if(i){
                if(*(arr+i-1) > *(arr+i)){
                    printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAHHHHHHHHHHHHHHHHHHHHHH\n");
                }
            }
            printf("%d ", *(arr+i));
        }
        printf("\n\n");
        printf("\n\n");
        free(arr);
    }
    return 0;
}

//  Esta funcion divide el vector en 2, y luego se envian sus dos partes a esta misma
// funcion, luego se combinan con merge.
void mergeSort(int *v, int tam){
    if(tam==1){
        return;
    }
    mergeSort(v, tam-tam/2);
    mergeSort(v+(tam-tam/2), tam/2);
    merge(v, v+(tam-tam/2), tam-tam/2, tam/2);
}

//  Esta funcion combina los vectores v1 y v2 de manera que se mantenga el orden de menor
// a mayor. Hay que notar que esta funcion considera que los vectores v1 y v2 ya estaban
// ordenados de menor a mayor.
void merge(int* v1, int* v2, int tam1, int tam2){
    int n=tam1, m=tam2, i;
    int *v=(int*)malloc((n+m)*sizeof(int));
    //  Vamos comparando los primeros valores de v1 y v2, y el menor de estos
    // lo colocamos en v.
    while(n && m){
        if(*(v1+(tam1-n)) > *(v2+(tam2-m))){
            *(v+(tam2+tam1-n-m))=*(v2+(tam2-m));
            --m;
        }
        else{
            *(v+(tam2+tam1-n-m))=*(v1+(tam1-n));
            --n;
        }
    }
    //  Se implementan los valores restantes de v1 en v
    // (este ciclo solo se inicia si se acabaron los valores de v2).
```

```
71      }
72      //  Se implementan los valores restantes de v1 en v
73      // (este ciclo solo se inicia si se acabaron los valores de v2).
74      while(n){
75          *(v+(tam2+tam1-n-m))=*(v1+(tam1-n));
76          --n;
77      }
78      //  Se implementan los valores restantes de v2 en v
79      // (este ciclo solo se inicia si se acabaron los valores de v1).
80      while(m){
81          *(v+(tam2+tam1-n-m))=*(v2+(tam2-m));
82          --m;
83      }
84      //  Como sabemos que v1 y v2 componen las mitades de un vector original de tal manera
85      // que la primera mitad y la segunda son v1 y v2 respectivamente, entonces igualamos los
86      // valores de v1 a la primera mitad del vector con los valores ordenados (v), y la segunda
87      // mitad a v2, de esta manera se ordena el vector original.
88      for(i=0;i<tam1;++i){
89          *(v1+i)=*(v+i);
90      }
91      for(i=0;i<tam2;++i){
92          *(v2+i)=*(v+tam1+i);
93      }
94      free(v);
95  }
```