Universidad Simón Bolívar
Departamento de Computación
y Tecnología de la Información
Laboratorio de Computación II
Ene-Mar 2019

> 311 *Lo que se llama alma.*—La suma de los movimientos interiores que le resultan fáciles al hombre, y que, por consiguiente, ejecuta de buen grado y con gracia , es lo que se llama alma; se dice de un hombre que carece de alma cuando deja traslucir que los movimientos del alma se le hacen penosos y duros.
>
> Aurora, Federico Nietzsche.

Lab. 3—Sem. 4—Algoritmos de Ordenamiento.

En este laboratorio definiremos y probaremos algunos de los algoritmos de ordenamiento vistos en clase y aprenderemos cómo se le pasa un función como parámetro a una función en el lenguaje C. Además mostraremos cómo usar la función (procedimiento) *qsort* de la librería standard *stdlib* para ordenar un arreglo de enteros y uno de cadenas de caracteres. Además si el tiempo lo permite usaremos la función *bserch* de la librería *stdlib* para hacer varias búsquedas en un arreglo previamente ordenado por *qsort*.

Tareas:

1. *Codificación de algoritmos de ordenamiento.*

   En un proyecto Nuevo de nombre **Lab3a** copiar el código que se muestran a continuación. El encargado de su Laboratorio debe explicarle como funciona la función *call* la misma es un ejemplo de una función que tiene una función como parámetro y cuya única función, en este caso, es que llame a un método particular de ordenamiento. Además observe como se invoca desde el *main()*.

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

void escribeArreglo(int a[], char name[], int n){
   int k = 0;
   printf("\n%s = [", name);
   while (k < n-1) {
      printf("%d, ", a[k]);
      k++;
   }
   printf("%d].", a[k]);
}

void swap(int *a, int *b){
   int t = *a; *a = *b; *b = t;
}

/*   Versiones del algoritmo de inserción  */

void insertSort(int a[], int n){
   int k = 1, j;
   while (k<n){
      j = k;
      while (j > 1 && a[j-1] > a[j]) {swap(&a[j-1], &a[j]); j--;}
      if (a[0] > a[1]) swap(&a[0], &a[1]);
      k++;
   }
}

void call(int a[], int n, void (*ord)(int *, int)){
   ord(a,n);
}
```

```
int main(){
   printf("\n Probando Ordenar:\n");
   int a[] = {12, 7, 4, 9, 2, 6};
   escribeArreglo(a, "a", 6);
   call(a, 6, insertSort);
   escribeArreglo(a, "a", 6);
   return 0;
}
```

2. *Variantes del Método de Inserción.*

   a) Agregue una modificación del algoritmo anterior que llamará insertSort2 en la cual eliminará *if* a la salida del ciclo y cambiará la condición de parada del ciclo. Para Probarla sólo debe cambiar el nombre de la función con que se invoca a *call*.

   b) Agregue la siguiente versión del método de inserción y Pruebe que funciona adecuadamente.

```
//Mejora de la versión anterior
void insertSortM(int a[], int n){
   int k = 1, j, aux;
   while (k<n){
      j = k; aux = a[k];
      while (j>=1 && a[j-1]>aux) {a[j] = a[j-1]; j--;}
      a[j] = aux;
      k++;
   }
}
```

3. *Probando el Método de Selección.* Agregue los siguientes dos algoritmos y pruébelas usando la función call.

```
///version de Wirth
void selectionSort(int a[], int n){
   int i, j, pm, aux;
   for (i = 0; i < n-1; i++){
      pm = i; aux = a[i];
      for(j = i+1; j<n; j++)
      if (a[j] < aux){pm = j; aux = a[pm];}
      {a[pm] = a[i]; a[i] = aux;}
   }
}

///version de Wirth-mod-vy
void selectionSortY(int a[], int n){
   int i, j, pm, aux;
   for (i = 0; i < n-1; i++){
      pm = i;
      for(j = i+1; j<n; j++) if (a[j] < a[pm]) pm = j;
      if (i != pm) {aux = a[pm]; a[pm] = a[i]; a[i] = aux;}
   }
}
```

4. *Método de Intercambio.* El siguiente ejercicio es para la casa!!! Agregue a su proyecto la siguientes funciones y pruébelas cambiando la invocación a *call* en el *main()*.

```
void intercambio(int a[], int n){
   int i, j;
   for (i = 0; i < n-1; i++)
      for (j = i+1; j < n; j++)
         if (a[j] < a[i]) swap(&a[i], &a[j]);
}

void bubbleSort(int a[], int n){
   int i, j, x;
   for (i = 1; i < n; i++)
      for(j = n-1; j >= i; j--)
         if(a[j-1] > a[j]){
            x = a[j-1]; a[j-1] = a[j]; a[j] = x;
         }
}

void shakeSort(int a[], int n){
   int j,k, L, R;
   L = 1, R = n-1, k = n-1;
   do{
      for (j= R; j>= L; j--)
      if(a[j-1] > a[j]){swap(&a[j-1], &a[j]); k=j;}
      L = k+1;
      for (j= L; j<= R; j++)
      if(a[j-1] > a[j]){swap(&a[j-1], &a[j]); k=j;}
      R = k-1;
   }while (L <= R);
}
```

5. *Cómo usar* qsort *y* bsearch. En un **nuevo proyecto** copie y pruebe el siguiente código. El mismo ordena el arreglo de forma no-creciente. Para invertir el orden basta eliminar el menos (-) en de *cmpString*. Haga la prueba.

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

int cmpString(const void* a, const void* b){
   return -strcmp(*(char* const*)a, *(char* const*)b);
}

int main(){
   int k; char *key = "amarguras";
   char * s[5] = {"zixia", "calor", "amargura", "amor", "amar"};
   qsort(s,5,sizeof(char*), cmpString);
   for (k = 0; k< 5; k++) printf("%s, ", s[k]);
   char **i = (char**)bsearch(&key, s, 5, sizeof(char*), cmpString);
   if (i != NULL) printf("\n%s se encontro en el arreglo", key);
   else printf("\n%s NO se encontro", key);
   return 0;
}
```

La función **bserch** y la función **qsort** tienen las siguientes interfaces:

```
void *bsearch(const void *key, const void *base, size_t nitems, size_t size,
              int (*comparator)(const void *, const void *));

void qsort(void *base, size_t num, size_t size,
               int (*comparator)(const void *, const void *));
```

6. *Cómo usar* qsort *y* bsearch. Copie y pruebe el siguiente código. El mismo ordena el arreglo de enteros de forma no-decreciente.

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

int a[] = { 88, 36, 98, 6, 18, 12 };

int cmpInt (const void * a, const void * b) {
   return ( *(int*)a - *(int*)b );
}

int main () {
   int k;
   printf("\nAntes de ordenar el arreglo: \n");
   for( k = 0 ; k < 6; k++ ) printf("%d ", a[k]);
   qsort(a, 6, sizeof(int), cmpInt);
   printf("\nDespues de ordenar el arreglo: \n");
   for( k = 0 ; k < 6; k++ ) printf("%d ", a[k]);
   return(0);
}
```

7. *Ordenar registros por notas y nombres.* A continuación se ilustra cómo se puede usar la función *qsort* para ordenar los elementos de un arreglo de registros usando un orden léxico-gráfico.

```
typedef struct est{
   int nota; char* name;
} Est;

void printArregloR(Est a[], int n){ // o char**a
   int k = 0;
   printf("\n");
   while(k < n-1){
      printf("%2d %s\n", a[k].nota, a[k].name);
      k++;
   }
   printf("%2d %s\n", a[k].nota, a[k].name);
}

int cmpReg2(const void * a, const void * b){
   return ((Est*)a)->nota - ((Est*)b)->nota != 0 ?
           ((Est*)a)->nota - ((Est*)b)->nota :
           strcmp(((Est*)a)->name, ((Est*)b)->name) ;
}

int cmpReg(const void * a, const void * b){
   Est x = *(Est*)a, y = *(Est*)b;
   if(x.nota == y. nota) return strcmp(x.name, y.name);
   else return x.nota- y.nota;
```

```
}
```

Escriba un main() y haga la llamadas necesarias para probar estás dos rutinas.

8. Escriba una función que permita ordenar los elementos de un arreglo de enteros de tal manera que los números primos aparezcan de primero ordenados de menor a mayor y los no primos aparezcan después ordenados de mayor a menor.

9. Escriba una función que permita ordenar los elementos de un arreglo de cadenas de caracteres de tal manera que las cadenas aparezcan de primero ordenadas de menor a mayor longitud y las de la misma longitud aparezcan ordenadas en el orden léxico-gráfico inverso.

10. Use la función *bsearch* para llevar a cabo varias búsquedas en usando el arreglo que ya ordenó en el ejercicio anterior. Hágalo en el mismo archivo anterior.

Nota: Se recomienda seriamente que no se hagan *cut and paste*. Debe copiar su código para que lo entienda mejor!!!