ICOM4015 Primavera 2001 Exámen II 5/10/01

Nombre: _____ **Sección:** _____

¡Anota tu nombre y número de sección en <u>todas las hojas</u> del examen AHORA!

Tienes 2 horas para completar tres problemas. Lee cuidadosamente todo el examen antes de empezar a trabajar. Muestra todo el trabajo conducente a tu contestación. Podrás recibir crédito parcial por contestaciones parciales siempre y cuando muestres tu trabajo por escrito. Usa tu tiempo inteligentemente. Suerte.

ICOM 4015 Staff

1	35
2	35
3	35
Total	105

ICOM4015 Primavera 2001 Exámen II 5/10/01

Nombre: _____ **Sección:** _____

Problema 1. Análisis de Algoritmos - Arrays (35 puntos)

(a) **(5 puntos)** Reordene la siguiente lista de mayor a menor grado de complejidad:

- O($n \log_2 n$)
- O ($n + n^2 + n^3$)
- O (2^n)
- O (n)

1. 2^n
2. $n + n^2 + n^3$
3. $n \log_2 n$
4. n

(b) **(5 puntos)** Indique cual es el tiempo de ejecución del siguiente código, en *Big O notation*.

```
for (int i=0; i<n; i++) {
  for (int j=0; j<n; ++j) {
    x = y % z;
  }
  for (int k=0; k<n; ++k) {
    cout << x - y;
  }
}
```

Big O Notation

$O(n^2)$

(c) **(8 puntos)** Escriba una función llamada ***palíndroma()***, que determine si una palabra es o no un palíndromo. Palíndroma es aquella palabra que se puede leer de igual forma al derecho y al revés. La función debe recibir la palabra en un arreglo de **char** y el largo en un **int**.

```
bool palindrome(char items[], int len) {


  bool temp = true;
  for (i=0; i == (len/2 - 1); i++) {
    if items[i] != items[len-(1+i)] {
      temp = false;
      break;
    }
  }
  return temp;



}
```

¡OJO! Problema 1 continúa en la próxima página ...

Nombre: _____ **Sección:** _____

(d) **(9 puntos)** La función ***alphacomp()***, tiene como argumentos a dos arreglos de letras. La función debe responder si es cierto o no que el primer arreglo esta léxicamente antes que el segundo arreglo, es decir, si alfabéticamente hablando se encuentra primero. Los arreglos pueden ser de distinto tamaño.

```
bool alphacomp(char firstarray[], int lfirst, char secondarray[], int lsecond)
{
  bool temp = true;
  int minlength = lfirst;

  if (lfirst > lsecond) minlength = lsecond;

  for (i=0; i < minlength; i++) {
    if (firstarray[i] > secondarray[i]) {
      temp = false;
      break;
    }
  }
  return temp;
}
```

(e) **(8 puntos)** Escriba una función llamada ***addlet()***, que inserte una letra en un arreglo de letras. El arreglo ha sido previamente ordenado alfabéticamente. La nueva letra se debe insertar en la posición que le correspondería alfabéticamente dentro del arreglo. Asuma que el arreglo es lo suficientemente grande para acomodar la nueva letra.

```
void addlet(char alphArray[], int longArray, char newLet) {


  int i = 0;
  int newPos;

  while ((alphArray[i] < newLet) && (i < longArray))
    i++;

  newPos = i;

  for (i = longArray; i >= newPos; i--) {
    alphArray[i+1] = alphArray[i];

  alphArray[newPos] = newLet;
}
```

Nombre: _____ **Sección:** _____

Problema 2. Algoritmos de Ordenamiento (35 puntos)

(a) **(18 puntos)** Escriba un algoritmo de ordenamiento mejorado de Bubblesort, que ordene descendentemente. El algoritmo debe terminar tan pronto el “array” quede ordenado y no continúe las iteraciones. Por ejemplo si el "array" ya esta ordenado solo será necesario recorrer el "array" una vez.

```
template <class T>
void bubblesort(T list[], int numElements)
{
```

¡OJO! Problema 2 continúa en la próxima página ...

Nombre: _____ **Sección:** _____

(b) **(17 puntos)** En la clase vimos el algoritmo de ordenamiento de Quicksort, que contiene la función de "partition" utilizando el primer elemento del "array" como el pivote para buscar el lugar donde se hace la partición del "array".

```
template <class TYPE>
static long partition(TYPE list[], long numElements, int low, int high)
{
  TYPE pivot = findPivot(list, numElements, low, high);
  int i = low;
  int j = high;
  while (i<j) {
    while ((list[i] < pivot) && (i <=high)) i++;
    while ((list[j] >= pivot) && (j >= low)) j--;
    if (i < j) {
      swap(list[i], list[j]);
    }
  }
  return ((list[i] < pivot)? i+1: i);
}
```

Escriba una función **“findPivot”** que devuelva el valor del pivote a ser utilizado en "partition" donde el pivote será calculado como el promedio de 2 valores seleccionados al azar del “array”. Recuerda que la funcion rand() devuelve un int en el intervalo [0,RAND_MAX].

```
#include<cstdlib.h>

template <class T>
T findPivot(T list [], long numElements, long low, long high) {

    Const int numSamples = 2;  // Numero de valores para calcular promedio









}
```

Nombre: _____ **Sección:** _____

Problema 3. Pointers a granel (30 puntos)

Esta sección contiene varios pedazos de programas que contienen errores de uso de memoria dinámica. Para cada problema, identifica el error con una explicación breve y provee una solución al mismo. Cada problema es independiente del próximo.

Parte 3.1. (10 puntos)

```
int f()
{
     char *a = new char;
     return 0;
}
```

Error	Corregido

Parte 3.2. (10 puntos)

```
void g()
{
    int* p;
    *p = 5;
    int* q = p;
    delete p;
    cout << “El valor de q es: “ << *q << endl;
}
```

Error	Corregido

¡OJO! Problema 3 continúa en la próxima página ...

Nombre: _____ **Sección:** _____

Parte 3.3. (10 puntos)

```
int g() {
    int * l[10];
    for(int i = 0; i<10; i++) {
        l[i] = new int(i);
    }
    delete [] l;
}
```

Error	Corregido