Tiempo: 100 Minutos (de 17:20 a 19:00), todas las preguntas tiene igual puntaje.

1. a) Escriba una función en C que permita evaluar en forma eficiente polinomios de un orden dado. El prototipo de la función es: double eval(double ***c***[], int ***orden***, double ***x***); El arreglo ***c*** contiene los coeficientes del polinomio, ***orden*** representa el orden del polinomio y ***x*** el punto a evaluar.

$$"eval(c,n,x)" = c[0]+c[1]x+.....+c[\text{n-1}]x^{n-1}+c[\text{n}]x^{n}$$

b) Determine el costo asintótico de su algoritmo en función del **orden** del polinomio.

a)

$$"eval(c,n,x)" = c[0]+\text{x}(c[1]+x(c[2]+\text{x}(.....+x(c[\text{n-1}]+x(c[\text{n}]+0))...)))$$

```
double eval (double c[], int orden, double x) {
        double sum;
        int i;
        sum=0;
        for (i=orden; i>=0; i--) {
                sum=c[i]+x*sum;
        }
        return sum;
}
```

6 pts por algoritmo eficiente + 6 por algoritmo que funcione.

b) Costo asintótico:

double eval (double c[], int orden, double x) {	**c1**
double sum;	
int i;	
sum=0;	**c2**
for (i=orden; i>=0; i++) {	**c3+c4(orden+1)+c5*orden**
sum=c[i]+x*sum;	**c6*orden**
}	
return sum;	**c7**
}	

T(n)=c1+c2+c3+n(c4+c5+c6) +c4+c7= K1+K2*n=θ(n).

8 por costo

Otra solución aceptable:

```
double eval (double c[], int orden, double x) {
        double sum;
        int i;
        double f=1;
        sum=0;
        for (i=0; i<=orden; i++) {
                sum=f*c[i]+sum;
                f=f*x;
        }
        return sum;
}
```

2. Una estrategia usada para mejorar el comportamiento promedio de Quicksort es tomar la mediana entre tres valores elegidos aleatoriamente y luego usarla como pivote.
a) Escriba en C el algoritmo que implementa esta estrategia.
b) Muestre un escenario que conduzca al costo de peor caso.

```
int  pivote (int a[], int le, int ri);
void miQuicksort(int a[], int le, int ri) {
        int tmp, q;
        if (le>=ri) return;
        q=pivote(a,le,ri);
        tmp=a[q];
        a[q]=a[le];
        a[le]=tmp;
        q=partition(a,le,ri); /* el mismo visto en clases se puede copiar de formulario */
        miQuicksort(a,le,q);
        miQuicksort(a,q+1,ri);
}
int  pivote (int a[], int le, int ri) {
        int ran, med, max;

        max = rand()%(ri-le) + le;
        ran=rand()%(ri-le) + le;
        if(a[ran]<a [max])
                mediana=ran;
        else {
                mediana=max;
                max=ran;
        }
        ran=rand()%(ri-le) + le;
        if (a[max] <= a[ran]) mediana=max;
        else if (a[mediana]<a[ran]) mediana=ran;
        return mediana;
}
```

4 por buen ensamble con quicksort

10 por buen cálculo de pivote

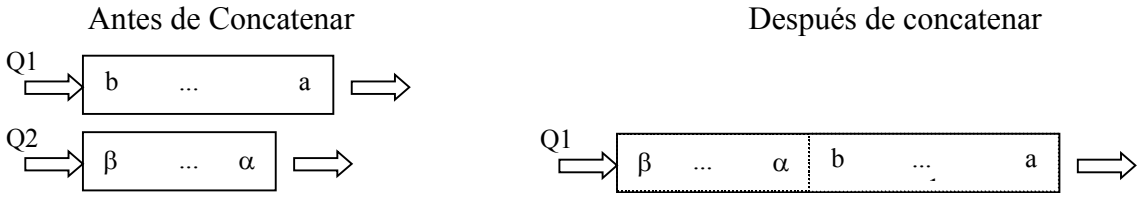
b) Dependiendo de la implementación de partition, un escenario de peor caso es aquel cuando el arreglo tiene todos sus valores iguales.
Si los números son distintos, un peor caso se da cuando cada vez la mediana corresponde a la estadística de orden 2 (segundo menor).
Hay otras respuestas posibles.

6 puntos

3. Se tiene dos colas Q1 y Q2 las cuales han sido implementadas usando listas circulares doblemente enlazadas con centinela. La estructura de cada nodo de la lista es:

```
typedef struct nodo_lista {
      struct nodo_lista * prev;
      struct nodo_lista * next;
      int elemento;
} NODO_LISTA;
```

Implemente la función **void concatenar(NODO_LISTA * Q1, NODO_LISTA * Q2)**, la cual pone al final de la cola Q1 todos los nodos de la cola Q2 manteniendo el orden de llegada. Q1 es la referencia a la nueva cola.




```
void concatenar(NODO_LISTA * Q1, NODO_LISTA *Q2) {
        if (Q2->next==Q2->prev)  return;   /*Lista Q2 vacía, nada que concatenar */
        Q1->next->prev=Q2->prev;
        Q2->prev->next=Q1->next;
        Q1->next=Q2->next;
        Q2->next->prev=Q1;
}
```

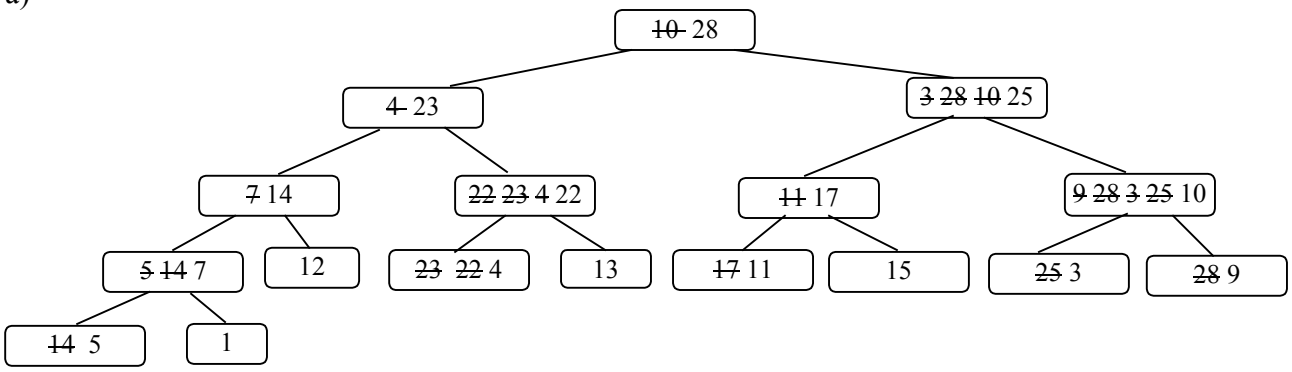
4. Considere el arreglo mostrado. Usando las operaciones vistas para un heap responda:

10	4	3	7	22	11	9	5	12	23	13	17	15	25	28	14	1

a) Muestre como queda el arreglo luego que se construye un heap aplicando builtHeap.
b) Muestre como queda luego de insertar 16 a la derecha del arreglo y restablecemos el heap.
c) Muestre como queda al extraer el mayor y restablecer el heap.

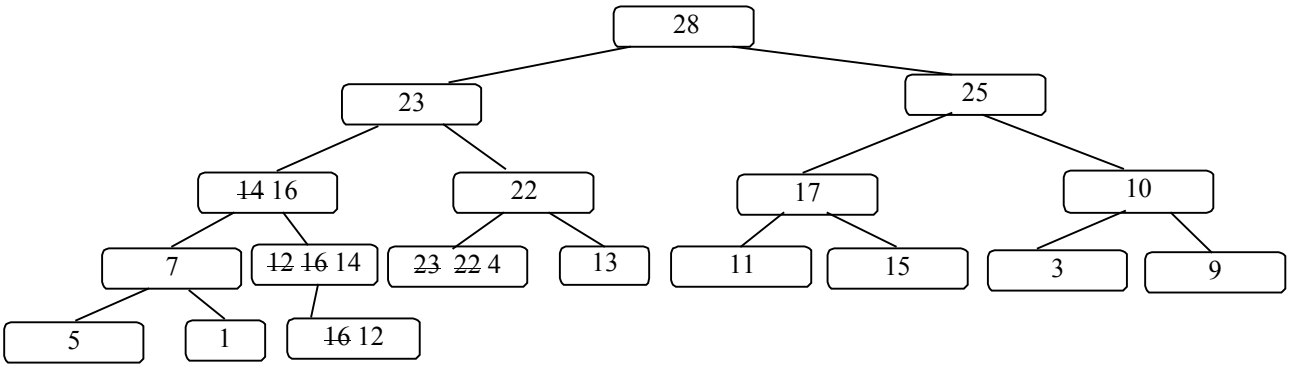
Se explicó en la prueba que las preguntas eran en secuencia, es decir hacer b) sobre el resultado de b) y c) sobre el resultado de b).

a)



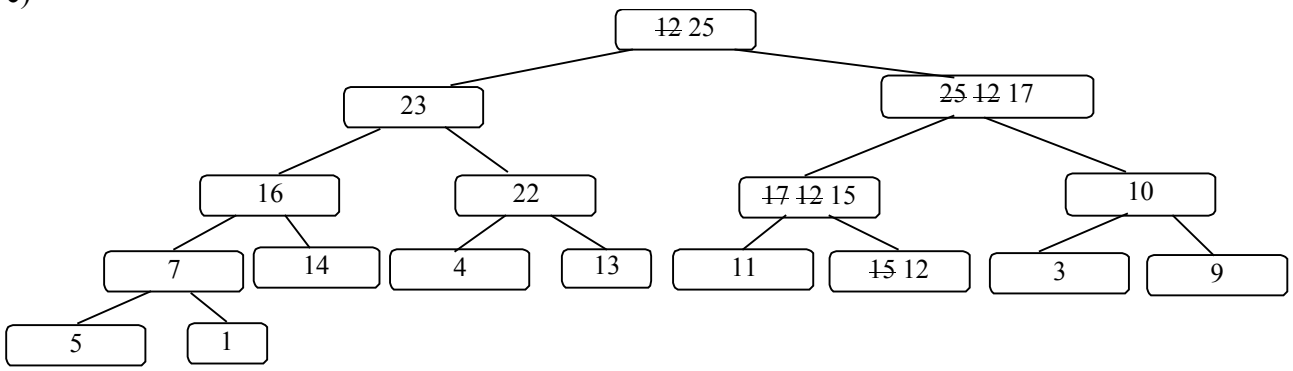
28	23	25	14	22	17	10	7	12	4	13	11	15	3	9	5	1

b)



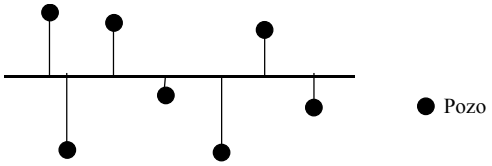
28	23	25	16	22	17	10	7	14	4	13	11	15	3	9	5	1	12

c)



25	23	17	16	22	15	10	7	14	4	13	11	12	3	9	5	1

5.- Usted está apoyando a una empresa petrolera que instalará una tubería de este a oeste a través de un campo de n pozos petroleros. Desde cada pozo saldrá una tubería a la recolectora siguiendo la ruta más directa (ya sea norte o sur). Teniendo como entrada las coordenadas (x,y) de cada pozo, ¿cómo decide usted la posición del colector para minimizar el total de tuberías?



Basta con tomar el valor de las ordenadas *y* todos los pozos, y luego buscar la mediana entre ellas. Si el número de pozos es impar, la media es única y el colector debe pasar por el pozo. Si el número de pozos es par, habrá dos medianas y el colector se pude ubicar en cualquier punto entre los pozos de las dos medianas.

Explicación más detallada. Si el colector está al sur de todos los pozos, cualquier desplazamiento de éste hacia el norte hará acortar todas la tuberías. Cuando el colector está al norte de sólo una, y son más de dos por el norte, nos convendrá correr el colector más al norte porque se alargará una tubería pero serán dos las que se acortarán. Podemos repetir esto hasta que llegaremos al punto donde el número de las tuberías que se acortan será igual al que se alarga. Esto ocurre en o

entre las medianas de las posiciones *y* de los pozos. En este punto de divide el número de pozos en partes iguales al sur y al norte del colector.