Tiempo: 100 Minutos (de 8:05 a 9:45), todas las preguntas tiene igual puntaje.

1. Considere los siguientes algoritmos de búsqueda:

```
int search_A(int a[], int v, int le, int ri) {
      int i;
      for (i=le; i<=ri; i++)
            if (v == a[i]) return i;
      return –1;
}
int search_B(int a[], int v, int le, int ri) {
      int m;
      while(ri >=le) {
            m=(le+ri)/2;
            if (v == a[m]) return m;
            if (v < a[m])
                  ri=m-1;
            else
                  le=m+1;
      }
      return –1;
}
```

a) Indique cuál es el mejor caso y el peor caso de la entrada para la función search_A. ¿Cuál es el costo en tiempo para estas situaciones?.
b) Indique cuál es el mejor caso y el peor caso de la entrada para la función search_B. ¿Cuál es el costo en tiempo para estas situaciones?.

Exprese sus costos usando notación asintótica.

Suposición indicada durante el certamen: El arreglo a está ordenado al iniciar la búsqueda en ambos casos.

a) El mejor caso se produce cunado el elemento buscado se encuantra en la primera posición del arreglo. El peor caso se produce cuando el elemento buscado no se encuentra en el arreglo.

Los costos son: Mejor caso tiene costo Θ(1). El segundo caso tiene costo es Θ(n). Esto debido a que se hace una busqueda lineal y se comparan todos los valores.

b) El mejor caso se produce cunado el elemento buscado se encuentra en la posición (le+ri)/2, donde le y ri son los extremos al iniciar la búsqueda.

El peor caso se produce cuando el elemento no es encontrado en el arreglo. El mejor caso tiene costo Θ(1) y el peor caso tiene costo Θ(log n) donde n es el núemor de elementos del arreglo inicial. Esto se explica porque en cada iteración el espacio de búsqueda es reducido a la mitad.

2. Para los datos almacenados en el arreglo de enteros mostrado se desea extraer uno a uno el mayor de los datos restantes en el arreglo. Para ello se propone usar un heap.

a) Muestre el estado del arreglo luego de construir el heap.
b) Muestre el estado del arreglo luego de extraer el mayor,
c) Repita b) hasta extraer los 3 enteros mayores del arreglo original.

12	4	1	7	23	14	9	5	7	3	2	6	15	10	8	11	25

a) La construcción del heap conduce a:

25	23	15	11	12	14	10	7	7	3	2	6	1	9	8	4	5

b) Luego de extraer el mayor y para poder seguir extrayendo datos el heap queda:

23	12	15	11	5	14	10	7	7	3	2	6	1	9	8	4

c1) Luego de extraer el siguiente mayor el heap queda:

15	12	14	11	5	6	10	7	7	3	2	4	1	9	8

c2) Luego de extraer el siguiente mayor el heap queda:

14	12	10	11	5	6	9	7	7	3	2	4	1	8

3. Codifique en C las operaciones Enqueue y Dequeue de una cola (queue) implementada con la estructura de datos lista doblemente enlazada con centinela. Considere la siguiente estructura para cada nodo:

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

```
void Enqueue(NODO_LISTA * head, NODO_LISTA * x) {
/* head apunta al centinela y x apunta al nodo que se desea
incorporar en la cola*/
      x->next = head->next;
      head->next->prev = x;
      x->prev = head;
      head->next=x;
}

NODO_LISTA * Dequeue(NODO_LISTA * head) {
/* head apunta al centinela, Dequeue retorna el más antiguo de la
cola*/
      NODE_LISTA * x; /* nodo a eliminar y retornar */
      x = head->prev;
      head->prev = x->prev;
      x->prev->next=head;
      return x;
}
```

4. Proponga un algoritmo codificado en C o pseudo lenguaje que reciba como entrada un puntero a la raíz de un árbol binario y retorne su altura. (Ayuda: observe que la altura de un nodo es uno más que la altura mayor de sus hijos)

Sea la siguiente estructura para cada nodo del árbol:

```
typedef struct nodo_arbol {
      struct nodo_arbol * p; /* puntero al padre */
      struct nodo_arbol * left; /* hijo izquierdo */
      struct nodo_arbol * right; /* hijo derecho*/
```

```
     ELEMENTO elemento;
} NODO_ARBOL;
int Altura (NODO_ARBOL * T) {
     int le, ri;
     if (T==NULL) return -1; /* en realidad no está definida la
altura en este caso (por observación de Manuel Jander)*/
     le = Altura(T->left);
     ri = Altura(T->right);
     if (le > ri) return(le+1);
     else return (ri+1);
}
```

5. Para el arreglo mostrado en la pregunta ~~3~~, **2** muestre las sucesivas particiones creadas tras la búsqueda de la mediana cuando se ejecuta Randomized_select. Asuma que siempre se selecciona el primer elemento como pivote para crear la partición.

12	**4**	**1**	**7**	**23**	**14**	**9**	**5**	**7**	**3**	**2**	**6**	**15**	**10**	**8**	**11**	**25**
11	**4**	**1**	**7**	**8**	**10**	**9**	**5**	**7**	**3**	**2**	**6**	**15**	**14**	**23**	**12**	**25**
11	**4**	**1**	**7**	**8**	**10**	**9**	**5**	**7**	**3**	**2**	**6**					
6	**4**	**1**	**7**	**8**	**10**	**9**	**5**	**7**	**3**	**2**	**11**					
6	**4**	**1**	**7**	**8**	**10**	**9**	**5**	**7**	**3**	**2**						
2	**4**	**1**	**3**	**5**	**10**	**9**	**8**	**7**	**7**	**6**						
					10	**9**	**8**	**7**	**7**	**6**						
					6	**9**	**8**	**7**	**7**	**10**						
					6	**9**	**8**	**7**	**7**							
						9	**8**	**7**	**7**							
						7	**8**	**7**	**9**							
						7	**8**	**7**								
							8	**7**								
							7	**8**								
								8								

Retorna 8 como la mediana.