Estructuras de Datos y Algoritmos
Ejemplo de Examen Parcial

Nota: En todos los casos justifique claramente su respuesta, caso contrario el ejercicio se considerará como no contestado. El parcial se aprueba con el 70%.

Se muestra aquí una posible solución a cada uno de los ejercicios. Hemos ampliado las justificaciones en cada caso, reforzando el porqué de cada justificación con los conceptos teóricos involucrados, para facilitarles la comprensión de la solución mostrada.

Ejercicio 1: un conjunto de N nuplas (siendo N par) se ha almacenado en una lista secuencial ordenada que mantiene ambos extremos móviles con recirculación y usa la siguiente estrategia de localización:

Se revisan secuencialmente las posiciones pares de la lista hasta encontrar el elemento (éxito), encontrar un elemento mayor al buscado o alcanzar la última posición de la lista, si el elemento no se encuentra en este recorrido y lo que se busca es menor que el elemento en esa posición, se revisan secuencialmente las posiciones impares hasta ese punto, teniendo éxito si se lo encuentra o fracasando en caso contrario.

Se pide:

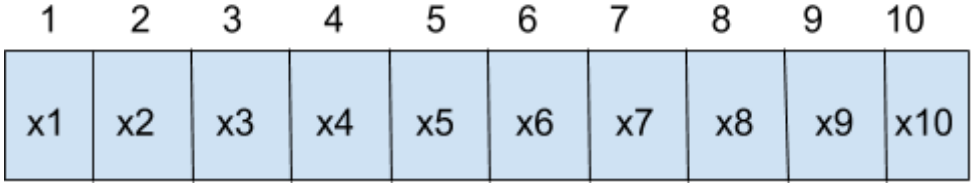
a) Obtener el vector de costos de la localización exitosa y que fracasa, aclarando la función de costo elegida.

Considerando como función de costo la cantidad de celdas consultadas, que el arreglo se inicia en la posición 1 y que el costo de la posición i del vector corresponde a lo que cuesta encontrar (o no) el elemento de la posición i en el arreglo; se obtienen los siguientes vectores.

$$\overline{c_{\text{éxito}}} = (2, 1,\ 4,\ 2,\ 6, 3, \ldots, 2((N/2)-1), (N/2)-1),\ N,\ N/2)$$

En el caso de localización exitosa, el vector de costo posee N componentes que corresponden a los costos de localizar con éxito cada uno de los N elementos presentes en la lista.

Como la forma de recorrer la lista secuencial es ordenada, para determinar que un elemento no se encuentra (fracaso en la localización) se deben revisar todas las celdas pares de la lista hasta alcanzar una mayor que el elemento buscado y luego revisar las celdas impares de la lista.
A modo de ejemplo, considérese tener la siguiente lista:



En una lista secuencial ordenada de N elementos hay $N + 1$ puntos de fracaso para la localización: los elementos menores que x1, los elementos mayores que x1 y menores que x2, los elementos mayores que x2 pero menores que x3, los elementos mayores que x3 pero menores que x4, y así siguiendo, hasta los elementos mayores que x9 y menores que x10 y los elementos mayores que x10.

En particular, por el método de búsqueda utilizado, las localizaciones sólo pueden fracasar al revisar una celda impar, salvo que se busque un elemento mayor que el último.

Si se busca por ejemplo un elemento entre x2 y x3, se revisarían las celdas en el siguiente orden: la de x2 (por ser la primera celda en posición par), como el elemento buscado es mayor que x2, se revisaría la celda de x4, pero como x4 es mayor que el elemento buscado, se detiene la revisión de las posiciones pares y se comienzan a revisar las celdas impares hasta ese punto. Así, se revisaría la celda de x1 y como el elemento buscado es mayor, se avanzaría a la celda de x3 en la que se detendría finalmente la búsqueda, porque x3 es mayor que el elemento buscado. Así, para un elemento entre x2 y x3 se revisaron 4 celdas. Se revisarían las mismas celdas si lo que se buscara estuviera entre x3 y x4.

Para las fronteras que rodean una celda en posición impar tendrán los mismos costos. Así, las fronteras de la celda de x1 deberán revisar 2 celdas (buscando algo menor a x1 y algo entre x1 y x2), las fronteras de la celda de x3 deberán revisar 4 celdas, las fronteras de la celda de x5 revisarán 6 y así siguiendo hasta que las fornteras de la celda x9 revisarán las 10 celdas. Sin embargo, si se busca un elemento mayor que el último presente la localización fracasaría al revisar la posición N de la lista, lo que implica revisar $N/2$ celdas. En el ejemplo sería que, si se buscara algo mayor que x10, sólo se deben revisar 5 celdas.

$$\overline{c}_{fracaso} = (2, 2, 4, 4, 6, 6, \ \ldots\ , 2((N/2) - 1),\ 2((N/2) - 1), N, N, N/2)$$

b) Obtener el vector de costos de inserción y de eliminación, aclarando la función de costo elegida.

Considerando como función de costo la cantidad de corrimientos de celda, por ser una lista secuencial ordenada y por tener ambos extremos móviles con recirculación.

Como la lista es de tamaño par y por tener ambos extremos móviles con recirculación, el peor caso de alta es incorporar un elemento en la frontera del medio; es decir algo entre el elemento en la posición $N/2$ y la posición $(N/2)$ +1, cuando se deben correr $N/2$ celdas (hacia cualquiera de los lados). Claramente, tal como el caso del vector de costos de localización que fracasa, el vector tiene N+1 componentes.

$$\overline{c}_{alta} = (0, 1, 2, 3, \ldots, (N/2) - 1, N/2,\ (N/2) - 1,\ (N/2) - 2, \ldots,\ 2, 1, 0)$$

En el caso de los costos de baja, como para poder realizar una baja exitosa el elemento a eliminar debe ser uno de los N que hacen que la localización sea exitosa, el vector de costos tendrá también N componentes:

$\overline{c_{baja}} = (0, 1, 2, 3, ..., (N/2) - 1, (N/2) - 1, (N/2) - 2, ..., 2, 1, 0)$

c) Obtenga los esfuerzos medios y máximos de localización exitosa, agregando las hipótesis que considere necesarias.

Esfuerzo Máximo de Localización exitosa: N

Esfuerzos Medio de Localización exitosa:

Se trabaja bajo la hipótesis que todos los elementos de la lista son igualmente probables de ser consultados. Probabilidad de cada costo $\frac{1}{N}$

$$\sum_{i=1}^{N/2} i * \frac{1}{N} + \sum_{i=1}^{N/2} 2i * \frac{1}{N} = \frac{1}{N} \sum_{i=1}^{N/2} (i + 2i) = \frac{1}{N} \sum_{i=1}^{N/2} 3i = \frac{3}{N} \sum_{i=1}^{N/2} i = \frac{3}{N} \frac{(N/2)((N/2)+1)}{2} = \frac{3}{4}((N/2) + 1) = \frac{3}{8}N + \frac{3}{4}$$

d) Expresar en notación asintótica el esfuerzo medio y máximo de localización exitosa.

Esfuerzos Máximos de Localización exitosa $\in O(N)$, porque claramente $N \in O(N)$, considerando $n_0 = 1$ y $c = 1$: $\forall n \geq 1,\ N \leq cN = N$.

Esfuerzos Medio de Localización exitosa $\in O(N)$, porque considerando $n_0 = 1$ y $c = 8$, se puede afirmar que $\forall n \geq 1,\ \frac{3}{8}N + \frac{3}{4} \leq 8N$.

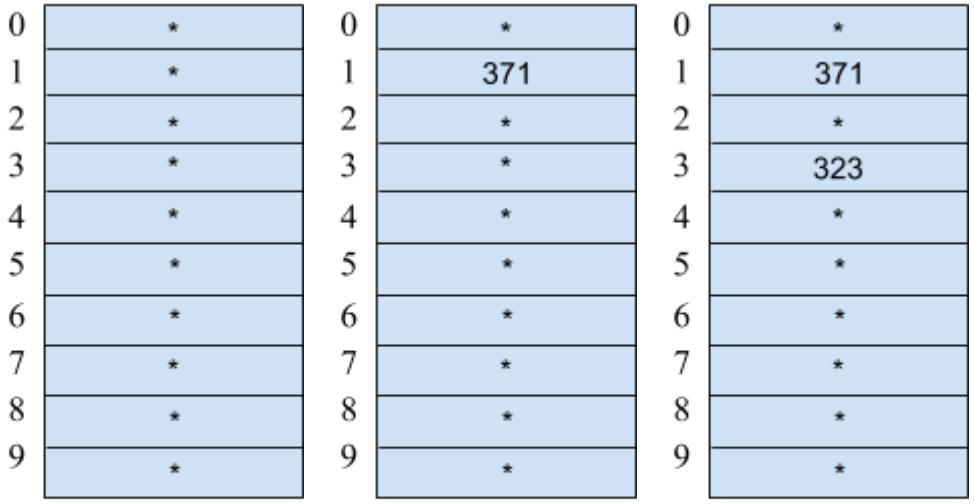
e) Compare esta variante de lista secuencial ordenada respecto de la lista secuencial ordenada habitual, dando ventajas y desventajas.

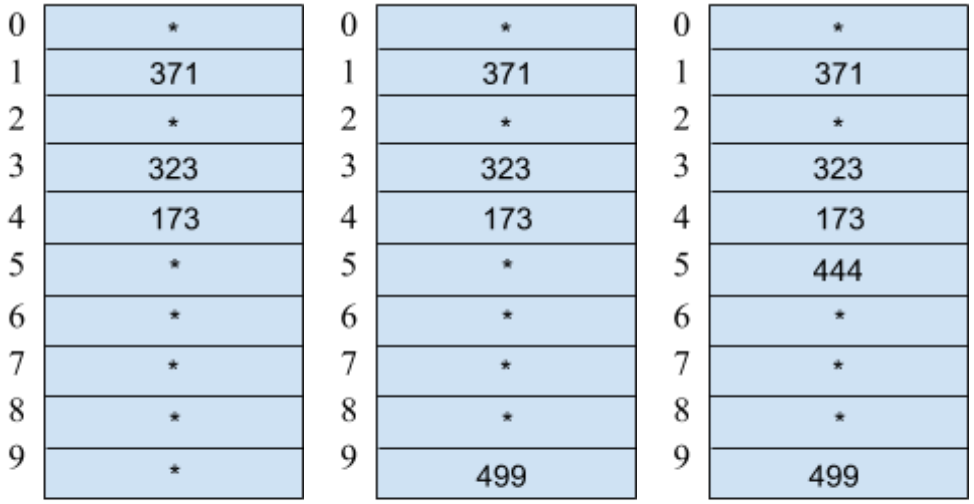
Claramente una de las ventajas de la lista secuencial ordenada es aprovechar el orden lo más posible. Como puede observarse, esta estrategia de búsqueda no lo hace. En lista secuencial ordenada se debe aplicar alguna de las variantes de búsqueda binaria, ya que se obtienen esfuerzos máximos de localización exitosa y que fracasa $O(logN)$. En este caso ambos esfuerzos máximos de localización son $O(N)$. Además, aunque los costos de altas y bajas coincidan con los de la lista secuencial ordenada, no se debe perder de vista que cada una de esas operaciones debe realizar una localización, que en este caso es más costosa.

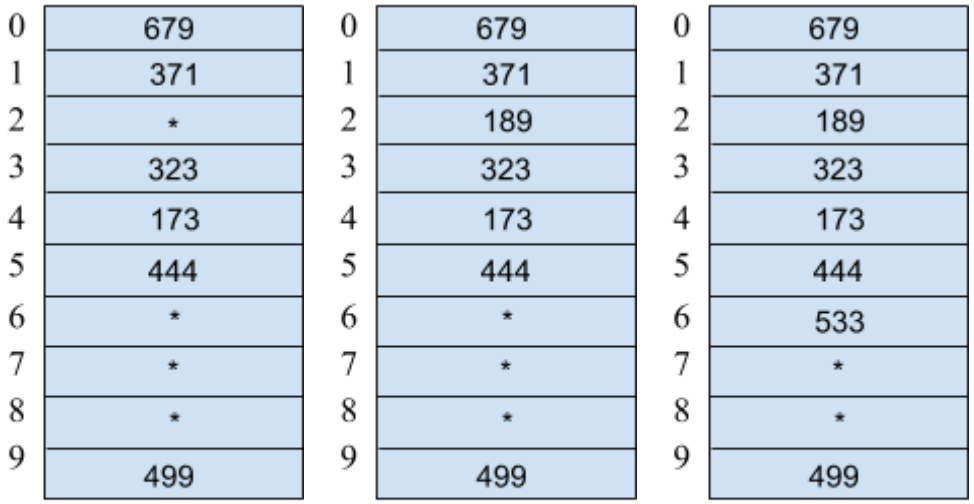
Ejercicio 2: Se quiere almacenar la secuencia: 371, 323, 173, 499, 444, 679, 189, 533 en una estructura de rebalse donde M = 10 y la función $h(x) = x \ mod\ 10$.

a) Mostrar, paso a paso, la correspondiente tabla de rebalse abierto lineal que se obtiene luego insertar los elementos de la secuencia.

Por ser la función usada $h(x) = x \ mod\ 10$, los valores de h para cada uno de los elementos serían: $h(371) = 1$, $h(323) = 3$, $h(173) = 3$, $h(499) = 9$, $h(444) = 4$, $h(679) = 9$, $h(189) = 9$, $h(533) = 3$.


0 * 0 * 0 *
1 * 1 371 1 371
2 * 2 * 2 *
3 * 3 * 3 323
4 * 4 * 4 *
5 * 5 * 5 *
6 * 6 * 6 *
7 * 7 * 7 *
8 * 8 * 8 *
9 * 9 * 9 *


0 * 0 * 0 *
1 371 1 371 1 371
2 * 2 * 2 *
3 323 3 323 3 323
4 173 4 173 4 173
5 * 5 * 5 444
6 * 6 * 6 *
7 * 7 * 7 *
8 * 8 * 8 *
9 * 9 499 9 499


0 679 0 679 0 679
1 371 1 371 1 371
2 * 2 189 2 189
3 323 3 323 3 323
4 173 4 173 4 173
5 444 5 444 5 444
6 * 6 * 6 533
7 * 7 * 7 *
8 * 8 * 8 *
9 499 9 499 9 499

b) Sobre la tabla del punto a), dar de baja los elementos 173 y 444, mostrando la estructura resultante.

0	679	0	679	0	679
1	371	1	371	1	371
2	189	2	189	2	189
3	323	3	323	3	323
4	173	4	+	4	+
5	444	5	444	5	+
6	533	6	533	6	533
7	*	7	*	7	*
8	*	8	*	8	*
9	499	9	499	9	499

c) Sobre la tabla del punto b), dar de alta el 549 y el 444, mostrando la estructura resultante. Si insertáramos estos dos elementos en orden inverso ¿obtendríamos la misma estructura?

0	679	0	679	0	679
1	371	1	371	1	371
2	189	2	189	2	189
3	323	3	323	3	323
4	+	4	549	4	549
5	+	5	+	5	444
6	533	6	533	6	533
7	*	7	*	7	*
8	*	8	*	8	*
9	499	9	499	9	499

Si los insertáramos en orden inverso tendríamos:

0	679
1	371
2	189
3	323
4	+
5	+
6	533
7	*
8	*
9	499

0	679
1	371
2	189
3	323
4	444
5	+
6	533
7	*
8	*
9	499

0	679
1	371
2	189
3	323
4	444
5	549
6	533
7	*
8	*
9	499

Con lo cual puede observarse que la estructura no es la misma.

Ejercicio 3: Sabiendo que se ha almacenado en una Parva de Mínimos un conjunto de *N* nuplas de la forma (*prioridad, elemento*):

a) Diseñar en pseudo-código la rutina que permita realizar el *Alta* de una nueva nupla en el conjunto, considerando que cuenta con las rutinas de *Hundir* y *Flotar*.

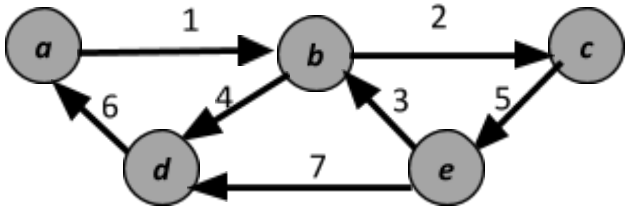
Se tiene:

```
nupla  record { prioridad: entero;
                elemento: datos; //datos es un registro (x,y) donde x es
                                 //clave e y es la información asociada
              }

Parva es arreglo de [1..M] de nupla

N: entero  //cantidad de elementos presentes en la parva

Además ya están programadas:
Flotar(pos: entero, P: Parva)  //rutina para reorganizar la Parva

Hundir(pos: entero, cant_elem: entero, P: Parva)//rutina que reorganiza la Parva

Alta(elem: nupla, P: Parva, exito boolean){
      si (N<M) entonces
                P[N++] := elem;
                Flotar(N, P);
                exito := true;   //alta en la parva
      sino
                exito := false;  //fracasa por falta de espacio
}
```

b) ¿Cómo se debería modificar su rutina para que trabajara sobre una Parva de Máximos?

La rutina de Alta no se debe modificar. El cambio debe realizarse en la rutina Flotar para que la comparación de prioridades sea P[pos/2].prioridad < P[pos].prioridad.

Ejercicio 4: Sea el siguiente dígrafo:

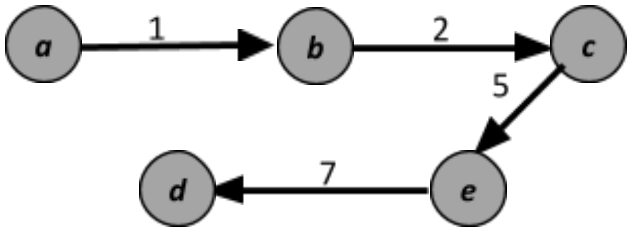


a) ¿Es un grafo bipartito?

No, porque por ejemplo si el vértice *b* perteneciera a una de las partes, los vértices *a*, *c*, *d* y *e* deberían pertenecer a la otra (porque *b* es adyacente a todos esos vértices), pero *a* no puede estar en la misma parte de *d* porque es adyacente a él. Así, no podría particionarse el conjunto de vértices en dos conjuntos disjuntos tal que su unión forme el conjunto de todos los vértices.

b) De un subgrafo parcial que sea arborescencia y que tenga 5 vértices.

Uno posible podría ser el subgrafo parcial cuyo conjunto de vértices sea $\{a, b, c, d, e\}$ y cuyo conjunto de arcos sea $\{1, 2, 5, 7\}$.



Es arborescencia porque:

1) es cuasi-fuertemente conexo (para cada par de vértices existe un vértice que alcanza a cada uno de los vértices del par mediante un camino) y sin ciclos (no existen cadenas cerradas y simples).

2) es un árbol (grafo conexo y sin ciclos) y tiene una raíz (el vértice *a* que alcanza a todos los vértices mediante un camino).

3) es cuasi-fuertemente conexo y tiene $n - 1$ arcos (en este caso, $n = 5$ y tiene 4 arcos).

4) El vértice *a* (la raíz de la arborescencia) está unido a todo vértice con un camino único.

5) es cuasi-fuertemente conexo, y si se elimina cualquiera de los arcos se pierde esta propiedad.

6) es cuasi-fuertemente conectado y $(\exists a)(g^-(a) = 0) \wedge (\forall x \neq a)(g^-(x) = 1)$.

7) No tiene ciclos y $(\exists a)(g^-(a) = 0) \wedge (\forall x \neq a)(g^-(x) = 1)$.

c) Considerando el subgrafo parcial del punto anterior:

1) ¿Es un árbol?

Como el subgrafo parcial es una arborescencia, necesariamente es un árbol. Una arborescencia es también un dígrafo conexo y sin ciclos.

2) ¿Es conexo?

Es conexo, ya que es posible encontrar un camino entre cada par de vértices. Además, gráficamente se puede observar que todos los vértices están conectados, porque no hay componentes desconectadas. No podría ser árbol ni arborescencia sin ser conexo.

3) ¿Es fuertemente conexo?

No, porque por ejemplo no existe camino desde el vértice *d* al vértice *e*.

4) ¿Es regular?

No, porque por ejemplo el vértice *a* tiene grado 1 y el vértice *d* tiene grado 2.

5) ¿Es cuasi-fuertemente conexo?

En particular, por ser una arborescencia debe ser cuasi-fuertemente conexo. Además, se puede verificar que para cada par de vértices del grafo se puede encontrar un vértice que llegue a ambos mediante un camino.

6) ¿Es trivial?

No, porque su conjunto de arcos no es vacío.

7) ¿Tiene raíces? En caso de existir, indique cuáles son.

Si tiene, y es sólo el vértice *a* que llega a todos los demás vértices del dígrafo por un camino.

8) ¿Tiene fuentes y sumideros? En caso de existir, indique cuáles son.

Si tiene. El vértice *a* es fuente porque no le llegan arcos. Y el vértice *d* es sumidero por que no tiene arcos de salida.

9) En caso de ser posible, muestre un camino que pase por todos los vértices. ¿Es simple? ¿Es elemental?

El camino (1,2,5,7) pasa por todos los vértices del grafo, es simple porque no repite arcos y es elemental porque no se pasa por ningún vértice más de una vez.

10) Muestre un cociclo elemental.

Por ejemplo el cociclo {1}, basado en {*a*} o su complemento es elemental, ya que el dígrafo es conexo y si se elimina el arco 1 deja de serlo y quedan dos componentes conexas (una más de las que existían previamente).

Ejercicio 5: Sea el siguiente algoritmo que determina el vértice centro en un 1-digrafo $G = (X, E)$, siendo $|X| = N$, $E \subseteq X^2$ y $|E| = M$. C es la matriz de costos de los arcos:

```
Centro (in C, out v)

    ⋮
for i = 1 to N do
    for j = 1 to N do
        A[i, j] = C[i, j]
for i = 1 to N do
    A[i, i] = 0
for k = 1 to N do
    for i = 1 to N do
        for j = 1 to N do
            if A[i, j] > A[i, k] + A[k, j] then
                A[i, j] = A[i, k] + A[k, j]
for i = 1 to N do
    Ex[i] = EXCENTRICIDAD (A, i)
min = Ex[1]
v = 1
for i = 2 to N do
    if Ex[i] < min then
        min = Ex[i]
        v = i
```

Se pide calcular el tiempo en notación asintótica que lleva la ejecución del procedimiento ***CENTRO***, sabiendo que el costo de la función ***EXCENTRICIDAD*** independientemente del valor de sus parámetros, toma tiempo $N \log N$.

Suponiendo que:

1. las asignaciones simples y comparaciones simples tienen orden constante O(1).
2. para selecciones el orden será:
   $O(\text{máx}(\text{Tiempo}_{\text{condición}}, \text{Tiempo}_{\text{rama\_verdadera}}, \text{Tiempo}_{\text{rama\_falsa}}))$, si la rama falsa no existiera
   $O(\text{máx}(\text{Tiempo}_{\text{condición}}, \text{Tiempo}_{\text{rama\_verdadera}}))$
3. dados P1 y P2 son trozos de programa y T1(n), T2(n) son los tiempos de ejecución de cada uno de ellos y además $T1(n) \in O(f)$ y $T2(n) \in O(g)$, entonces
   a. si se ejecutan en secuencia P1 y P2 se tiene que T1(n) + T2(n) es O(máx(f, g)) y
   b. si se ejecutan anidados P1 y P2 su tiempo de ejecución $T1(n) \times T2(n)$, es $O(f \times g)$.
4. las asignaciones y comparaciones con llamados a funciones tienen el orden de la función.

Centro **(in** *C*, **out** *v*)

⋮

for *i* = 1 **to** *N* **do** O(N*N) = O(N^2) **(T1)**
  for *j* = 1 **to** *N* **do** O(1*N) = O(N) } (por 3.b.)
    A[*i*, *j*] = *C*[*i*, *j*] O(1) } (por 3.b.)
      (por regla 1.)

for *i* = 1 **to** *N* **do** O(1*N) = O(N) **(T2)**
  A[*i*, *i*] = 0 O(1) } (por 3.b.)
    (por regla 1.)

for *k* = 1 **to** *N* **do** O(N^3) **(T3)**
  for *i* = 1 **to** *N* **do** O(N^2) } (por 3.b.)
    for *j* = 1 **to** *N* **do** O(N) } (por 3.b.)
      if *A*[*i*, *j*] > *A*[*i*, *k*] + *A*[*k*, *j*] **then** O(1) (por 1.) O(máx(1,1))=O(1) } (por 3.b.)
        A[*i*, *j*] = *A*[*i*, *k*] + *A*[*k*, *j*] O(1) (por 1.) } (por 2.)

for *i* = 1 **to** *N* **do** O(N*(N*log N)) = O (N^2*log N) **(T4)**
  Ex[*i*] = ***EXCENTRICIDAD*** (*A*, *i*) O(N*log N) (por 4.) } (por 3.b.)

min = *Ex*[1] O(1) (por 1.) **(T5)**

v = 1 O(1) (por 1.) **(T6)**

for *i* = 2 **to** *N* **do** O(N) **(T7)**
  if *Ex*[*i*] < *min* **then** O(1) } (por 3.b.)
    min = *Ex*[*i*] O(1) (por 1.) O(máx(1,1))=O(1) } (por 2.)
    v = *i* O(1) (por 1.) } (por 2.)

luego sumando los tiempos de cada bloque:

T1+T2+T3+T4+T5+T6+T7 es O(máx(N^2, N, N^3, (N^2*log N), 1, 1, N)) = O(N^3)
(por regla 3.a.)