Heapsort

Objetivo del algoritmo:

1) Dado un conjunto de claves que son ingresadas en forma aleatoria, se busca ordenar el conjunto minimizando el tiempo de búsqueda.

Introducción.

Por lo general, todos los algoritmos de ordenamiento están compuestos por un proceso de carga de los elementos en una estructura y un proceso de ordenamiento. En el caso del *heapsort* la estructura auxiliar utilizada es un árbol binario donde se establece un orden parcial entre sus nodos, en donde el padre será mayor que sus hijos. Por su parte, el algoritmo de ordenamiento tendrá los siguientes pasos:

1) Ejecutar un barrido por niveles de donde obtengo un vector
2) Intercambiar el vector(1) con el vector(n)
3) n = n - 1
4) Rearmar el árbol p/ que tenga un orden parcial entre 1 y n
5) Si n <> 1, repetir el proceso desde 1)
6) Fin del algoritmo.

Orden de Complejidad.

El orden de complejidad del proceso de carga del árbol binario es de orden *logarítmico* pues el árbol es binario es decir $\log_2 (n+1)$. Este valor es el esperado para cargar un elemento, como el proceso consta de n elementos, el orden será $n \log_2 (n+1)$

En el proceso de ordenamiento, al tratarse de un árbol binario, el orden es el mismo que en el proceso de carga. Como ambos procesos tienen el mismo orden, se concluye que el orden del heapsort es $n \log_2 (n+1)$.

Definición de Heapsort.

Arbol implementado dentro de un array por medio de un barrido por niveles.

Carga del árbol binario

1) Definir un árbol binario casi completo de n nodos donde el identificador de cada nodo es <= al identificador de su padre (árbol parcialmente ordenado) implica que la raíz es el elemento más grande del grupo. Para la ejecución del algoritmo

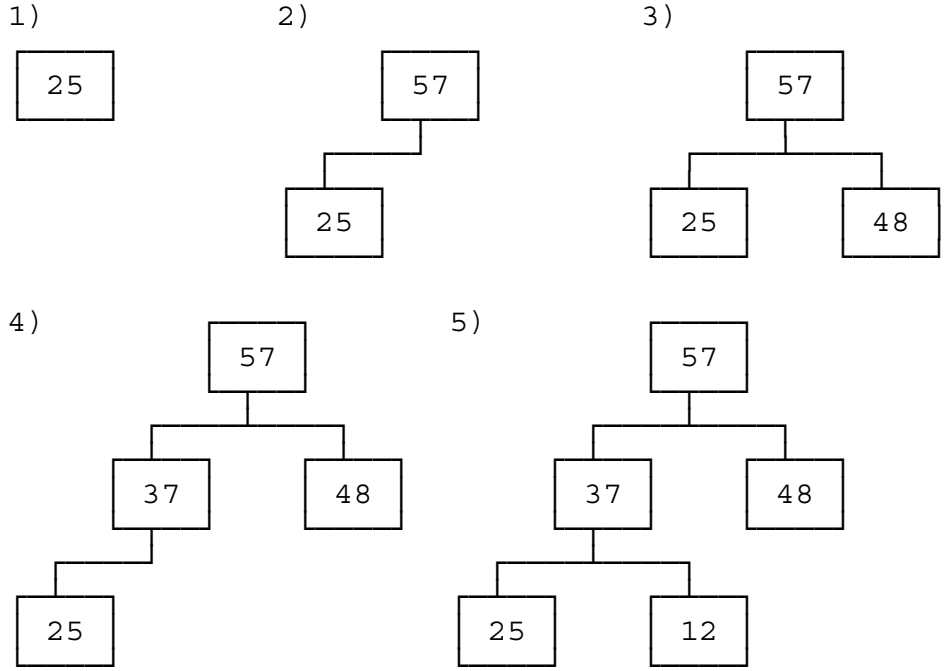
se utiliza además un vector asociado al árbol. Una vez fijada la raíz del árbol, cada elemento ingresado al arbol, será ingresado *por niveles* cumpliendo con la siguiente regla:

```
If   Hijo Izquierdo  = nil
     El nodo ingresa como Hijo izquierdo del nodo padre
else
     If   Hijo Derecho = nil
          El nodo ingresa como Hijo derecho del nodo padre
     else
          Analizo próximo nodo mismo nivel
```

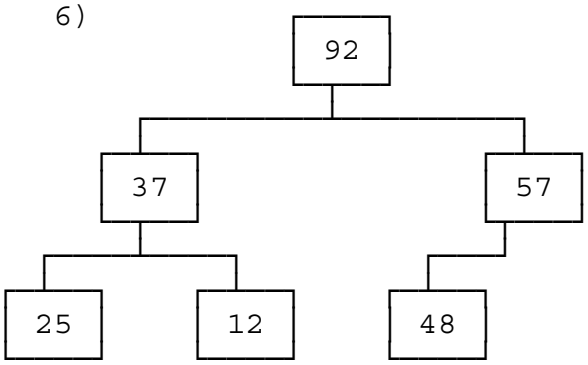
A medida que los nodos son ingresados en el árbol, tambien son ingresados en el vector en la primer posición libre. Esto se realiza para mejorar la performance del algoritmo. Supongamos que el conjunto de claves que serán ingresadas es el siguiente:

< 25, 57, 48, 37, 12, 92, 86, 33 >

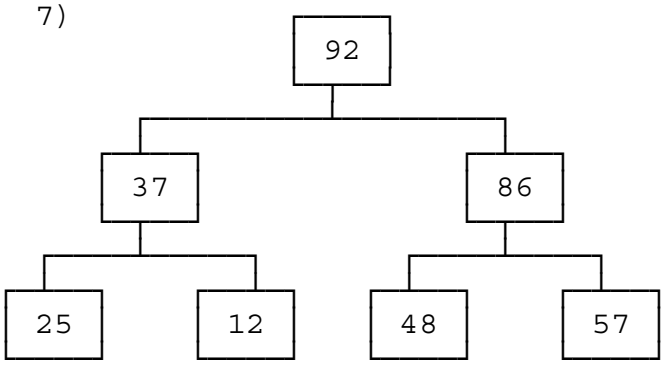
El proceso de carga será como se presenta a continuación:



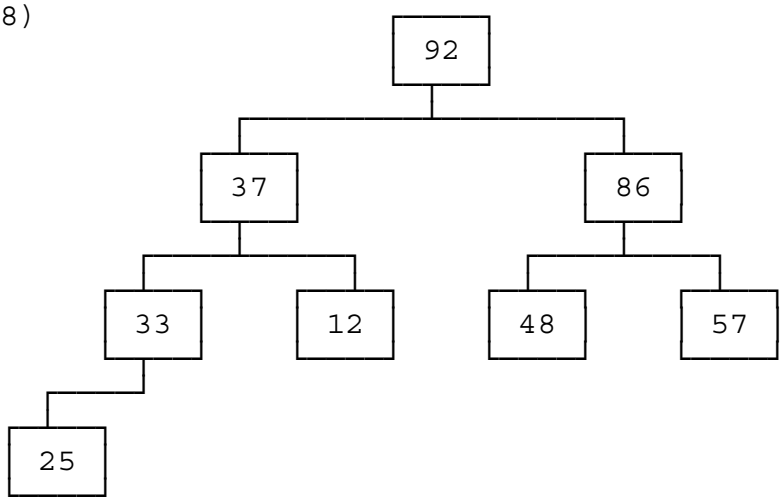
6)



7)



8)



El vector asociado se irá completando de la siguiente manera:

1)

25	00	00	00	00	00	00	00

2)

57	25	00	00	00	00	00	00

3)

57	25	48	00	00	00	00	00

4)

57	37	48	25	00	00	00	00

5)

57	37	48	25	12	00	00	00

6)

92	37	57	25	12	48	00	00

7)

92	37	86	25	12	48	57	00

8)

92	37	86	33	12	48	57	25

i : 1 2 3 4 5 6 7 8

Posición de los Hijos en el vector.

i	i_{izq}	i_{der}
1	2	3
2	4	5
3	6	7
4	8	
5		
6		
7		
8		

El subíndice con la posición de los hijos de un nodo es importante en la etapa del ordenamiento pues, en ese momento se valida que el padre cumpla con la relación de orden parcial respecto de sus hijos.

de donde :

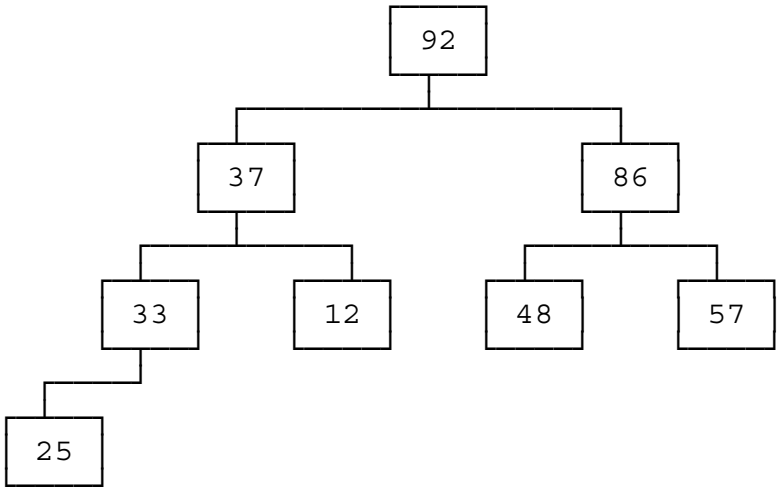
$$i_{izq} = i * 2$$
$$i_{der} = (i * 2) + 1$$

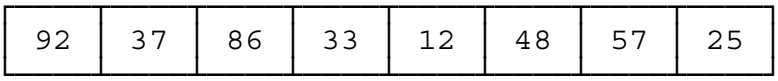
de donde : $i_{padre} = int [i / 2]$

Más adelante veremos como se arma el vector en función de la dirección del padre

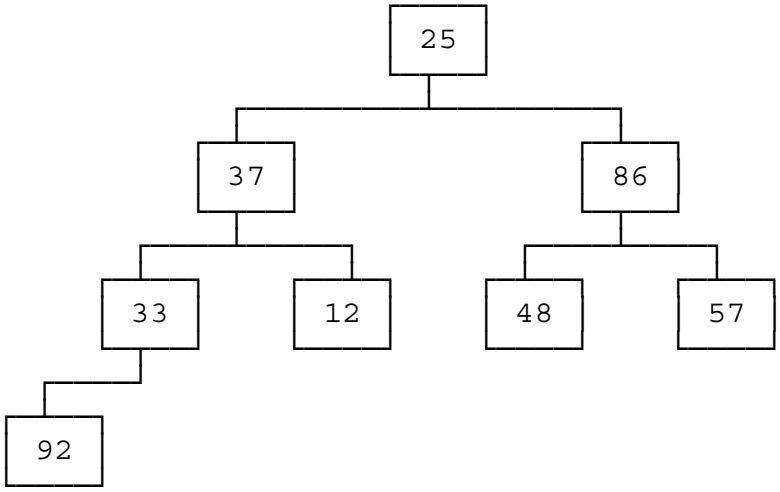
Ejecución del algoritmo

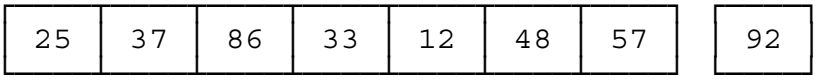
1) Partiendo del árbol obtenido al ingresar los n nodos y del vector asociado.





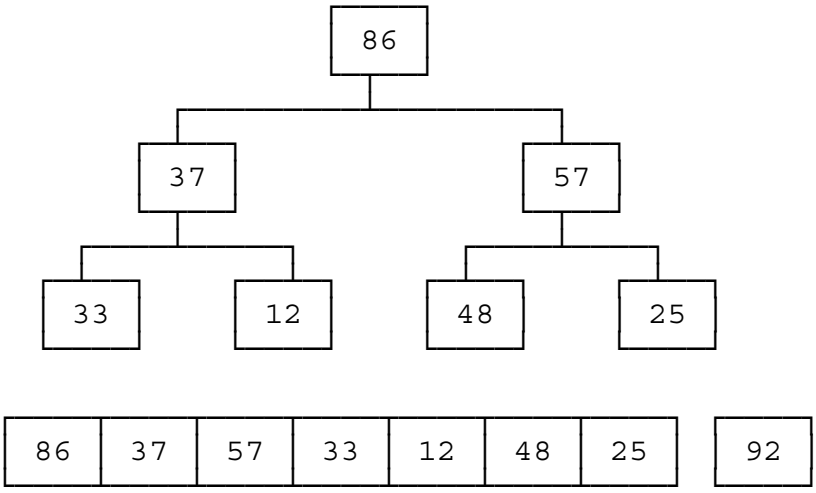
2) Intercambiar el vector(1) con el vector(n)



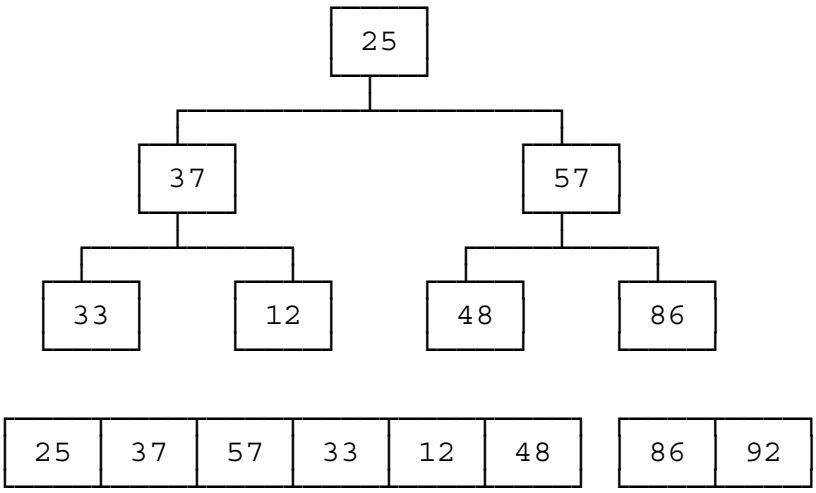


3) En este momento sabemos que en la última posición del vector se encuentra el elemento con identificador mayor, es decir el 92. Este elemento ocupará definitivamente esa posición en el vector y por lo tanto el proceso de reorganización para su ordenamiento se efectuará sobre todos los elementos del árbol (del vector) excepto el último elemento, por lo tanto es válido asignarle a n el valor n - 1 y rearmar el árbol para los n elementos (ahora 7) restantes.

4) Rearmar el arbol entre 1 y 7.

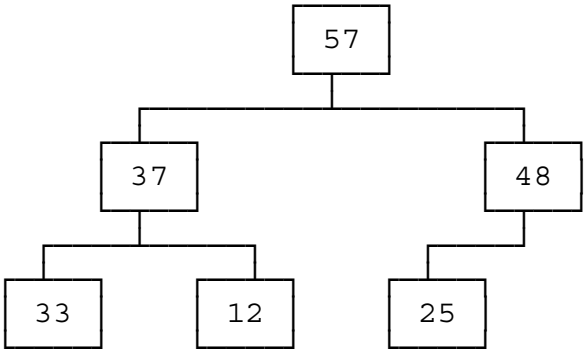


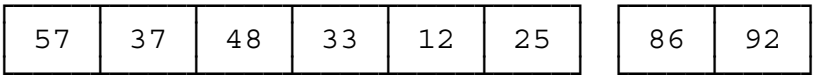
Luego de intercambiar el vector(1) con el vector(7)



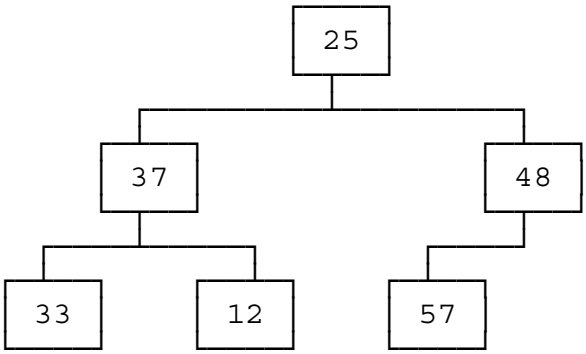
Ahora las dos últimas posiciones del vector se encuentran ordenadas, por lo tanto el mismo proceso se repite hasta que todos los elementos ocupen el lugar que le corresponden. Resumiendo :

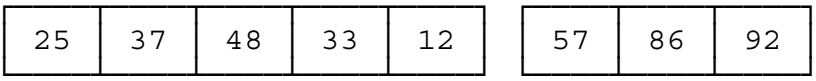
Rearmar el arbol entre 1 y 6



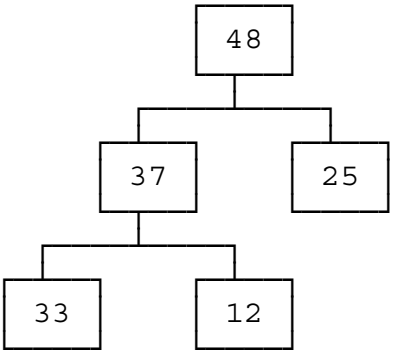


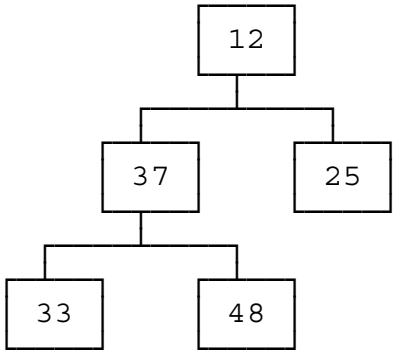
Intercambiar el vector(1) con el vector(6)





Rearmar el arbol entre 1 y 5 y cambiar el vector(1) con el vector(5)



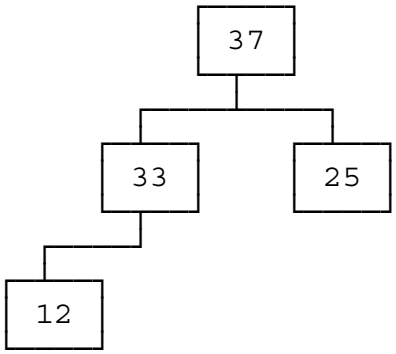


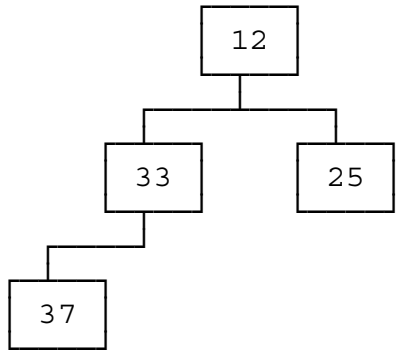
48	37	25	33	12

12	37	25	33

48	57	86	92

Rearmar el arbol entre 1 y 4 y cambiar el vector(1) con el vector(4)



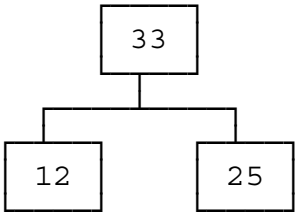


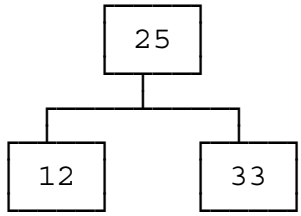
37	33	25	12

12	33	25

37	48	57	86	92

Rearmar el arbol entre 1 y 3 y cambiar el vector(1) con el vector(3)



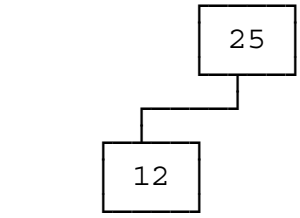


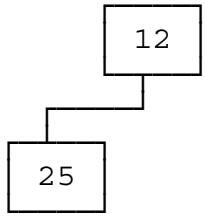
33	12	25

25	12

33	37	48	57	86	92

Rearmar el arbol entre 1 y 2 y cambiar el vector(1) con el vector(2)





25	12

12	25	33	37	48	57	86	92

Carga del árbol mediante el vector asociado

Se ingresa la primer clave

25	00	00	00	00	00	00	00

Se ingresa la segunda clave

25	57	00	00	00	00	00	00

Al ingresar el segundo elemento debe preguntarse si el valor de la clave del elemento ingresado es mayor que el valor de su padre. De ser así, debe procederse al intercambio de claves. En nuestro caso, la posición del vector donde se encuentra el padre de la clave 57 se obtiene mediante i_{padre} = int [i / 2] por lo tanto la dirección del padre de la clave 57 está en el subíndice 1. Al ser de valor mayor deben intercambiarse las claves.

57	25	00	00	00	00	00	00

Se ingresa la tercer clave

57	25	48	00	00	00	00	00

El padre de 48 esta en $i_{padre\ 48}$ = int [3 / 2] = 1. Como 57 > 48, no hay intercambio.

Se ingresa la cuarta clave

57	25	48	37	00	00	00	00

El padre de 37 esta en $i_{padre\ 37}$ = int [4 / 2] = 2. Como 25 < 37, debe haber intercambio.

57	37	48	25	00	00	00	00

Nuevamente debe analizarse como es 37 respecto del valor de su padre. El nuevo padre de 37 esta en $i_{padre\ 37}$ = int [2 / 2] = 1. Como 57 > 37, no hay intercambio, por lo tanto esta es la posición que hasta ahora ocupa la clave 37.

Se ingresar la quinta clave

57	37	48	25	12	00	00	00

Luego de realizar el mismo análisis, vemos que la clave 12 no modifica su posición.

Se ingresa la sexta clave

57	37	48	25	12	92	00	00

$i_{padre\ 92}$ = int [6 / 2] = 3 como 48 < 92 hay intercambio

$i_{padre\ 92}$ = int [3 / 2] = 1 como 57 < 92 hay intercambio

el vector luego de los intercambios quedará de esta manera

92	37	57	25	12	48	00	00

Se ingresa la septima clave

92	37	57	25	12	48	86	00

$i_{padre\ 86}$ = int [7 / 2] = 3 como 57 < 86, hay intercambio

$i_{padre\ 86}$ = int [3 / 2] = 1 como 92 > 86, no hay intercambio

el vector luego de los intercambios quedará de esta manera

92	37	86	25	12	48	57	00

Se ingresa la octava y última clave

92	37	57	25	12	48	86	33

$i_{padre\ 33}$ = int [8 / 2] = 4 como 25 < 33, hay intercambio

$i_{padre\ 33}$ = int [4 / 2] = 2 como 37 > 33, no hay intercambio

el vector luego de los intercambios quedará, ya en forma definitiva de esta manera :

92	37	86	33	12	48	57	25

De esta manera se habrá conseguido implementar un árbol binario en un vector por medio de un barrido por niveles, *sin haber realizado el barrido* consiguiendo así reducir la complejidad de los algoritmos sin la necesidad de recurrir al uso de otra estructura externa que un vector ni de demasiados punteros.

A continuación se presenta un programa en pascal con el algoritmo del heapsort :

6.4. program heapsort;

```
Const CantNum = 8;

Type  Numeros = integer;
      Vector  = array [1..CantNum] of Numeros;

var   Number    : Numeros;
      VarVector : Vector;

Procedure CargoVector (var WorkVector : Vector);
var i  : integer;
Begin
  For i := 1 to CantNum do
  begin
    clrscr;
    Write('Ingrese Numero : ');
    Readln(Number);
    Writeln(Number);
    Writeln('  ');
    WorkVector[i] := Number;
  end;
end;

Procedure Listar (var WorkVector : Vector);
var i  :  integer;
begin
  For i := 1 to CantNum do
  begin
    writeln(WorkVector[i]);
  end;
end;

Procedure Heap (var VarNum : Vector);
var   i     : integer;
      izq   : integer;
      der   : integer;
      X     : Numeros;
      n     : Numeros;
begin
  n := CantNum;
  repeat
  begin
    X         := VarNum[n];
    VarNum[n] := VarNum[1];
    VarNum[1] := X;
    n         := n - 1;
    For i := 1 to n do
    begin
      izq := i * 2;
      der := izq + 1;
      if  (izq <= n)
      then begin
             if  (der <= n)
```

```
                then begin
                       { * Hijo Izq. : SI  Hijo Der. : SI * }
                       if   (VarNum[izq] > VarNum[der])
                       then begin
                              if   (VarNum[i] < VarNum[izq])
                              then begin
                                     X           := VarNum[i];
                                     VarNum[i]   := VarNum[izq];
                                     VarNum[izq] := X;
                                   end
                              else begin
                                     if   (VarNum[i] < VarNum[der])
                                     then begin
                                            X := VarNum[i];
                                            VarNum[i]   :=
                                              VarNum[der];
                                            VarNum[der] := X;
                                          end
                                   end
                            end
                       else begin
                              if   (VarNum[i] < VarNum[der])
                              then begin
                                     X           := VarNum[i];
                                     VarNum[i]   := VarNum[der];
                                     VarNum[der] := X;
                                   end
                             end
                     end
                else begin
                       { * Hijo Izq. : SI  Hijo Der. : NO * }
                       if   (VarNum[i] < VarNum[izq])
                       then begin
                              X           := VarNum[i];
                              VarNum[i]   := VarNum[izq];
                              VarNum[izq] := X;
                            end
                     end
         else begin
                { * Hijo Izq. : NO  Hijo Der. : NO * }
                i := n;
              end;
      end;
    end;
    until ( n = 1 );
  end;

  Begin
       CargoVector(VarVector);
       Listar(VarVector);
       writeln('  ');
       Heap(VarVector);
       Listar(VarVector);
```

```
  end.
```