

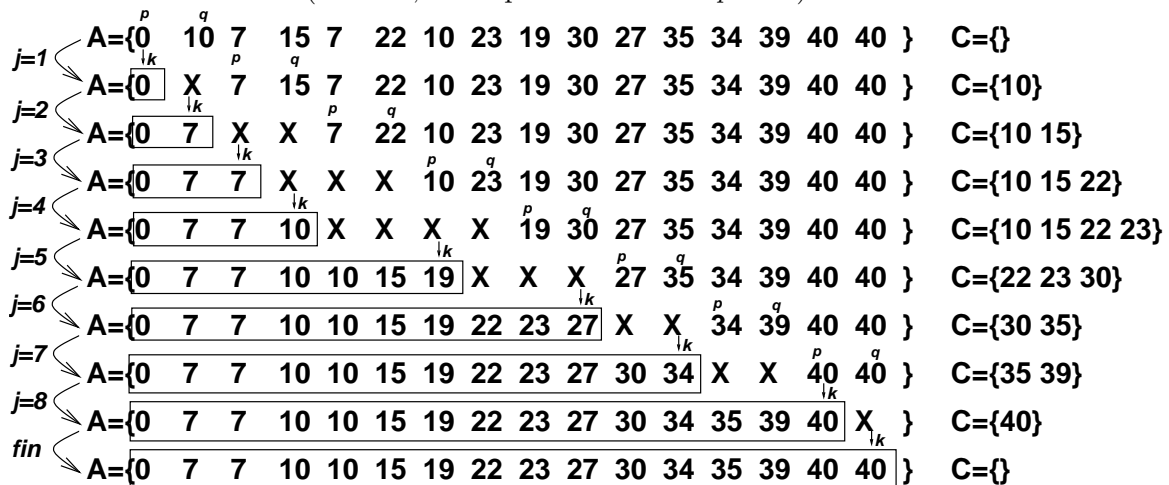
Algoritmos y Estructuras de Datos.

Parcial 3. Tema 1b. [25 de Junio de 2002]

Ej. 1.- Escribir las funciones primitivas del TAD COLA DE PRIORIDAD representada por montículos. Incluir todas las definiciones de tipo necesarias.

- (a) ANULA
- (b) INSERTA
- (c) SUPRIME_MIN

Ej. 2.- El algoritmo mergesort modificado: El procedimiento Mergesort se basa en la estrategia “*dividir para vencer*”. Primero se ordenan las posiciones impares entre sí (recursivamente), luego las pares, y luego se intercalan ambas subsecuencias. Si el intercalamiento se puede hacer en tiempo $O(n)$ entonces puede demostrarse que el número de operaciones total es $O(n \log(n))$ en el peor caso. Ya hemos visto para la representación de conjuntos como listas enlazadas clasificadas que el intercalamiento de dos listas enlazadas clasificadas se puede hacer en tiempo $O(n)$. Para vectores no es tan sencillo, y se conjetura que no se puede hacer tal operación en tiempo $O(n)$ sin uso de memoria adicional (es decir, no se puede hacer “*in place*”).



Se propone el siguiente algoritmo de intercalación para vectores que utiliza una cola auxiliar C (y por lo tanto no es “*in place*”). Por simplicidad asumimos que el número de elementos en el arreglo n es par, que las posiciones impares y pares están clasificadas entre sí, e inicialmente la cola C está vacía. Considerando el arreglo A de longitud $n = 16$ de la figura, el algoritmo lo ordena en $n/2 = 8$ pasos. En cada paso el algoritmo ordena los elementos p, q en las posiciones $2j - 1$ y $2j$. Después del paso j los primeros $2j$ elementos están ordenados y ubicados en parte en la cabecera del vector (posiciones 1 a k , marcados en la figura con una caja rectangular) y en la cola C . Entre la posición $k + 1$

y la posición $2j - 1$ hay una serie de elementos (tantos como hay en C) cuyos valores son irrelevantes (en la figura están marcados con una X). El algoritmo es el siguiente

```
for j:=1 to n div 2 do begin
  p := A[2j-1]; q := A[2j]; r := min(p,q);
  pone 'max(p,q)' en C;
  pone todos los elementos de 'C' menores o iguales que 'r' en
    la cabecera de 'A' (despues de la posicion 'k');
  pone 'r' en la cabecera de 'A';
end;
pone todos los elementos restantes de 'C' en la cabecera de 'A';
```

Así, por ejemplo, en el paso 6 del ejemplo de la figura consideramos los elementos $p = 27$, $q = 35$ en las posiciones 11 y 12. El mínimo de ambos es $r = 27$ de manera que insertamos 35 en C . Tomamos los elementos de C menores que r (22 y 23), y los ponemos en la cabecera de A y finalmente ponemos r en la cabecera. Durante este paso el número de elementos ordenados en la cabecera se incrementó en 3 y la longitud de la cola C se redujo en 1. Si bien este algoritmo no es “*in place*” el tamaño de la cola C es en promedio de longitud $O(\sqrt{n})$ (puede ser n en el peor caso).

Consigna: Escribir un procedimiento MERGEQ que intercala dos secuencias de acuerdo al algoritmo descripto previamente.

```
tipo_elemento = integer;
type vector = array[1..N] of tipo_elemento;
procedure MERGEQ(var A:vector; N:tipo_elemento);
```

Usar las primitivas del TAD COLA:

```
procedure ANULA(var C:cola);
procedure PONE(x:tipo_elemento; var C:cola);
procedure QUITA(var C:cola);
function FRENTE(C:cola): tipo_elemento;
function VACIA(C:cola): boolean;
```

Ej. 3.- Preguntas sobre clasificación: [Responder según el sistema “*multiple choice*”, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente “descabelladas” y tienen puntajes **negativos**!]

- (a) Todos los algoritmos de clasificación genérica (es decir, que se pueden aplicar a conjuntos infinitos de claves, como los enteros o los reales) tienen un número de operaciones $T(n)$ que es...

- ☐ $T(n) > O(n^2)$
☐ $T(n) > O(1/n)$

- ☐ $T(n) > O(n \log(n))$
☐ $T(n) > O(\sqrt{n})$
- (b) Las operaciones INSERTA y SUPRIME_MIN sobre el TAD COLA DE PRIORIDAD representado por montículos requiere un número de operaciones que está relacionado con ...
- ☐ ... el cuadrado del número de elementos en el montículo n
☐ ... el logaritmo de la semilla.
☐ ... la longitud del camino en el árbol desde la raíz hasta el punto de inserción o supresión.
☐ ... el número de elementos en el montículo n
- (c) ¿Cuál de los siguientes algoritmos de clasificación es $O(n \log(n))$ en *promedio* y $O(n^2)$ en el *peor caso*?
- ☐ Clasificación por montículos (*“heap-sort”*)
☐ Inserción
☐ Clasificación rápida (*“quick-sort”*)
☐ Clasificación súbita (*“snake-sort”*)
- (d) ¿Cuál de los siguientes algoritmos *“lentos”* realiza *siempre* exactamente n intercambios?
- ☐ Inserción
☐ Selección
☐ Repetición (*“humble-sort”*)
☐ Burbuja (*“bubble-sort”*)

Ej. 4.- Realizar las siguientes tareas

- (a) Dados los enteros $\{97, 85, 78, 63, 62, 58, 50, 33, 19, 9\}$ ordenarlos por el método de *“montículos”* (*“heap-sort”*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.
- (b) Dados los enteros $\{685, 405, 520, 51, 811\}$ ordenarlos por el método de urnas (*“bin-sort”*), con $B = 10$ urnas, y tantas pasadas como sea necesario. Mostrar la tabla de urnas y su contenido después de cada pasada. Especificar cuál es la instrucción Pascal que se debe utilizar para ubicar los elementos en las urnas durante cada pasada.
- (c) Dados las cadenas de caracteres {Borodin, Vivaldi, Mendelsohn, Strauss, Beethoven, Mozart, Litz} insertarlos, en ese orden, en un *“árbol binario de búsqueda”*. Luego, mostrar las operaciones necesarias para eliminar los elementos {Vivaldi, Mozart}