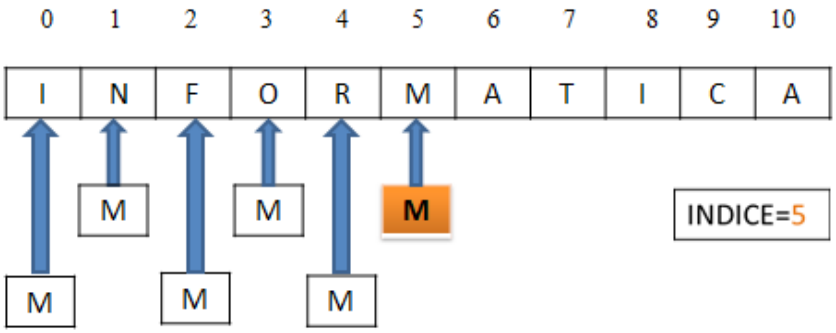
Anexo I.- Arrays busca y ordenación

Métodos de cerca

1. Busca lineal en un array no ordenado

Buscar, de forma secuencial, un elemento en un array hasta que lo encontremos. Si el elemento no se encuentra, se debe de indicar de alguna forma.

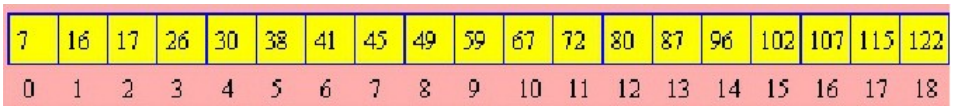


Su complejidad temporal es de:

- En el mejor de los casos: 1 (el elemento se encuentra al principio).
- En el peor de los casos: n (el elemento no está al vector).

2. Busca lineal en un array ordenado

La diferencia respecto a buscar en un array no ordenado es que el proceso se puede parar antes (no siempre). Si buscamos un valor en un array ordenado ascendentemente, en el momento que encontramos un valor mayor que el elemento que estamos buscando, sabemos que el elemento que buscamos ya no estará a partir de esa posición.



Si buscamos el elemento 35 solo necesitaremos hacer 6 comparaciones (hasta la posición 5) puesto que contiene el número 38 y por tanto los valores que quedan son mayores.

Su complejidad temporal es de:

- En el mejor de los casos: 1 (el elemento se encuentra al principio).
- En el peor de los casos: n (el elemento no está al vector).

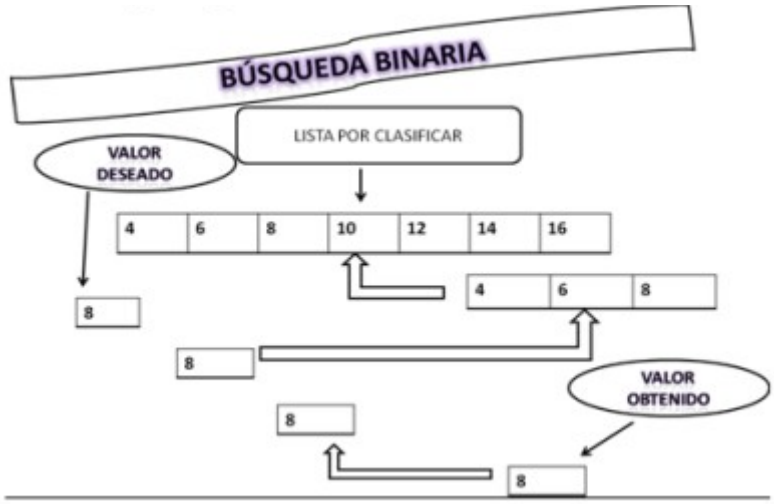
3. Busca binaria o dicotómica en un array ordenado

Consiste al utilizar la técnica "Divide y vencerás". Este método consiste en calcular la posición central de la array y repetir el proceso de la siguiente manera:

- Si el elemento buscado es igual al contenido de la posición central: se ha encontrado.
- Si el elemento buscado es menor que el contenido de la posición central: se repetirá el proceso en

la parte izquierdo.

- Si el elemento buscado es mayor que el contenido de la posición central: se repetirá el proceso en la parte derecha.



Si buscamos el número 8 solo se tienen que hacer 3 comparaciones.

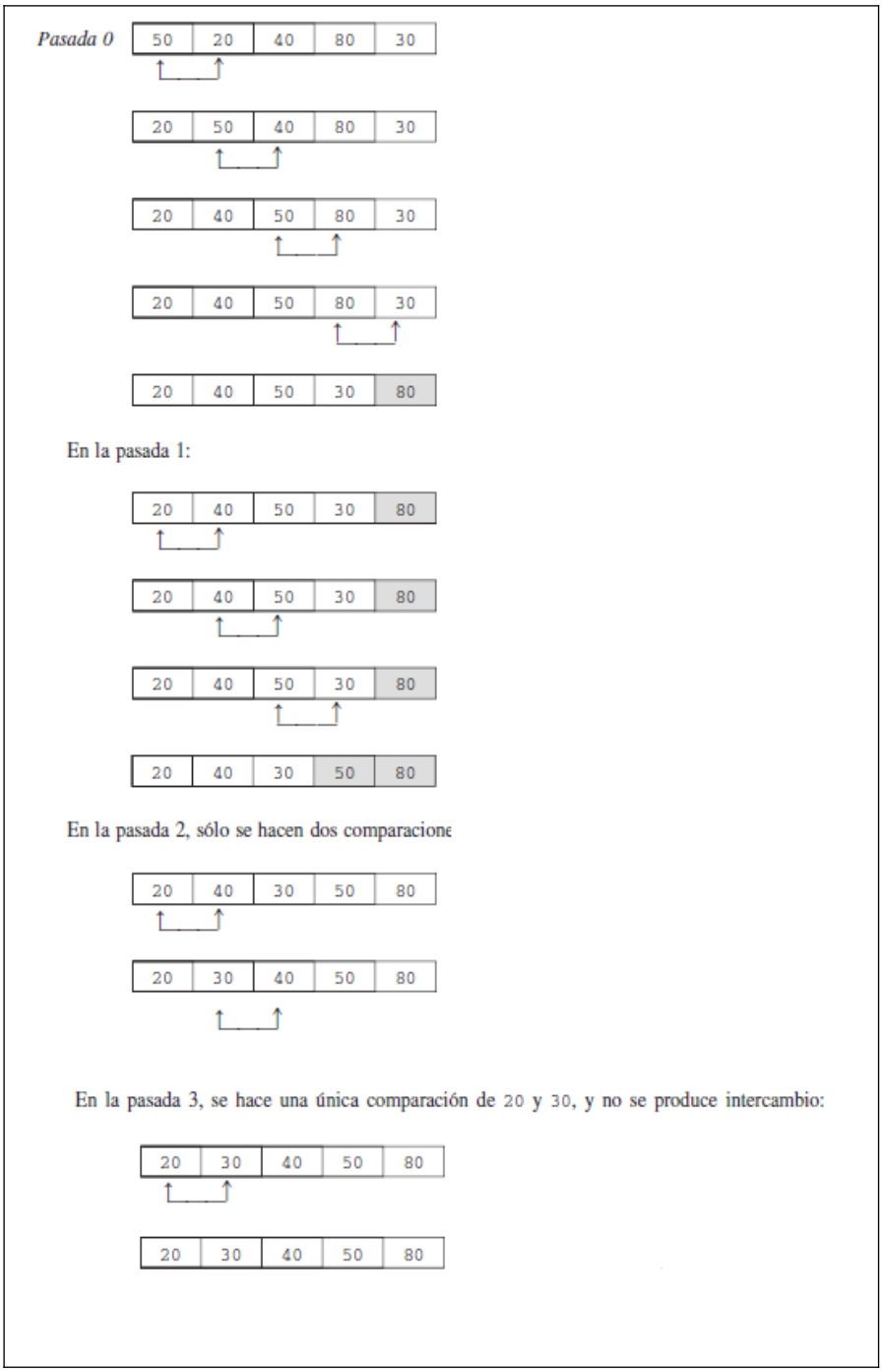
Su complejidad temporal es de:

- En el mejor de los casos: 1 (el elemento se encuentra a la posición central).
- En el peor de los casos: log n (el elemento no está al vector).

Métodos de ordenación

1. Método de la burbuja

Es trata de comparar el primer elemento con el segundo y si el segundo es menor que el primero, se intercambian los valores. Después el segundo con el tercero y así sucesivamente. Si no hay intercambios, es que el array está ordenado (esto es una mejora inmediata: si en una pasada del bucle no se produce ningún intercambio, está ordenado y se puede parar el proceso). Tiene una complejidad (coste temporal) de $n^{2,}$ es decir, que aumenta de forma exponencial (potencia de dos) cuando mayor sea el número de elementos a ordenar. Si se hace la mejora, su coste temporal sería **de n**.



2. Inserción

Consiste en recorrer todo el array empezando desde el segundo elemento hasta el final. Para cada elemento, se trata de colocarlo en el lugar correcto entre todos los elementos anteriores a él, es decir, entre los elementos a su izquierda en el array.

Dada una posición actual p, el algoritmo se basa en que los elementos A[0], A[1], ..., A[p-1] ya están ordenados.

Tiene una complejidad (coste temporal) de $\mathbf{n^2}$ (en el peor de los casos) y en el mejor de los casos (cuando lo array ya está ordenado) la complejidad es **de n**.

De forma gráfica el proceso que sigue el método de inserción directa es el siguiente:

30	15	2	21	44	8	Array original
30	**15**	2	21	44	8	Se empieza por el segundo elemento. Se compara con el primero. Como 15 < 30 se desplaza el 30 hacia la derecha y se coloca el 15 en su lugar
15	**30**	2	21	44	8	
15	**30**	**2**	21	44	8	Seguimos por el tercer elemento. Se compara con los anteriores y se van desplazando hasta que el 2 queda en su lugar.
2	**15**	**30**	21	44	8	
2	**15**	**30**	**21**	44	8	Continuamos por el cuarto elemento. Se compara con los anteriores y se van desplazando hasta que el 21 queda en su lugar.
2	**15**	**21**	**30**	44	8	
2	**15**	**21**	**30**	**44**	8	Lo mismo para el quinto elemento En este caso ya está en su posición correcta respecto a los anteriores.
2	**15**	**21**	**30**	**44**	8	
2	**15**	**21**	**30**	**44**	**8**	Y finalmente se coloca el último elemento El array queda ordenado
2	**8**	**15**	**21**	**30**	**44**	

3. Selección

El método de ordenación por selección consiste a repetir los siguientes pasos:

1. Se busca el elemento más pequeño del array y se coloca en la primera posición.
2. Entre los restantes, se busca el elemento más pequeño y se coloca en la segunda posición.
3. Entre los restantes se busca el elemento más pequeño y se coloca en la tercera posición.
4. …….
5. Este proceso se repite hasta colocar el último elemento.

Tiene una complejidad (coste temporal) de $\mathbf{n^2}$ (en el peor de los casos) y en el mejor de los casos (cuando lo array ya está ordenado) la complejidad es de **n**.

De forma gráfica el proceso sería el siguiente:

Array original a ordenar: [50, 26, 7, 9, 15, 27]

50	26	7	9	15	27	Array original
7	26	50	9	15	27	Se coloca el 7 en primera posición. Se intercambian 7 y 50
7	9	50	26	15	27	Se coloca el 9 en segunda posición. Se intercambian 9 y 26
7	9	15	26	50	27	Se coloca el 15 en tercera posición. Se intercambian 15 y 50
7	9	15	26	50	27	El menor de los que quedan es el 26. Se deja en su posición
7	9	15	26	27	50	Se coloca el 27 en quinta posición. Se intercambian 27 y 50

4. Quicksort

Consiste en ordenar un array mediante un pivote, que es un punto intermedio en el array, haciendo que a la izquierda del pivote queden los menores y a la derecha los mayores. Se debe de repetir el proceso en cada una de las mitades del array. Utiliza recursividad (por lo tanto se invocará asimismo). Tiene de media una complejidad (coste temporal) de **n log n**, en el peor de los casos $\mathbf{n^2}$.