**Entregables**

* Publicar trabajo en Wikipedia de la plataforma.
* Informe en PDF y formato latex incluyendo los siguientes apartados:
  + Portada
  + Introducción
  + Estudio del arte con respecto al método seleccionado
  + Calculo de complejidad teórica
  + Explicación de funcionamiento del método aplicado a tres problemas distintos.
  + Mediciones de tiempos
    - Se pide realizar 3 experimentos distintos programando el método con RUBY, cada uno con set de datos con N [1.000, 10.000, 100.000, 1.000.000, 10.000.000, 100.000.000]. Estos experimentos deben ajustarse al mejor caso, el caso promedio y el peor caso.
    - Se deben presentar y explicar los resultados.
  + Conclusiones (5 como mínimo)
* Video explicativo del método usando naipes (El curso debe coordinar un canal youtube y publicar la totalidad de los videos en el).
* Presentación en PDF-Latex con Beamer

Pauta de Evaluación

* Informe 10%
* Wikipedia 10%
* Presentación 20%
* Conclusiones 20%
* Desarrollo de programa para experimento 20%
* Interrogación 20%

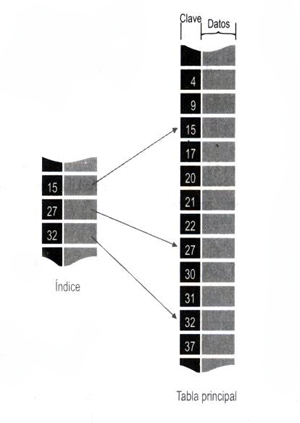
**Técnica de Búsqueda Secuencial Indexada.**

**Descripción de la técnica.**

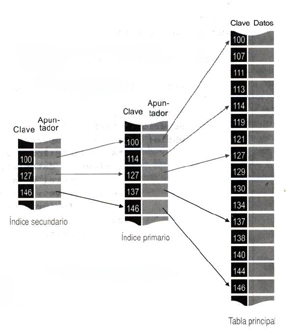
Un método popular para superar las desventajas de los archivos secuenciales es el del archivo secuencias indexado; pero implica un aumento en la cantidad de espacio requerida.

Funciona de la siguiente manera:

Se reserva una taba auxiliar llamada indice ademas del archivo ordenado mismo. Cada elemento en el indice consta de una llave kindex y un apuntador al registro en el archivo que corresponde a kindex. Los elementos en el indice al igual que los elementos en el archivo, deben estar ordenados en la llave. Si el indice es de un octavo del tamaño del archivo, se representa en el indice cada octavo registra el archivo.



Si el indice comienza a crecer tanto que se vuelve ineficaz se puede usar un indice secundario que funciona casi de la misma forma que el indice principal, solo que apunta a este, no a la tabla principal la busqueda empieza con una exploracion por el indice secundario; esto nos lleva a un subarreglo en el indice principal; despues el procesamiento continua normalmente. Un ejemplo de lo anterior es la siguiente figura.

****

**Ventajas de la técnica.**

Permite procesar el archivo secuencialmente por orden lógico y también procesarlo al azar.

La ventaja real del método secuencial indexado es que los elementos en la tabla pueden ser examinados en forma secuencial si todos los registros en el archivo deben ser accesados, pero sin embargo, el tiempo de búsqueda para algún elemento en particular se reduce considerablemente. La búsqueda secuencial se realiza en la tabla de índices que es más pequeña en lugar de la tabla más grande. Una vez que se ha encontrado un índice correcto, se hace una segunda búsqueda secuencial únicamente en la parte reducida de la tabla que contiene los registros.

La organización secuencial indexado es conveniente para archivos con mediana volatilidad, actividad variable y tamaño relativamente estable.

Las eliminaciones de una tabla secuencial indexada se pueden hacer fácilmente mediante la asignación de banderas a las entradas que son eliminadas. Durante la búsqueda secuencial a través de la tabla, se ignoran las entradas que han sido eliminadas.

Otra fuente

*La ventaja del método estriba en que la búsqueda se realiza inicialmente en el array de índices, cuantitativamente más pequeña que el array de elementos. Cuando se ha encontrado el intervalo correcto, se hace una segunda búsqueda en una parte reducida del array. Estas dos búsquedas pueden ser secuenciales o binarias y el tiempo de ejecución dependerá del tipo de búsqueda utilizado en cada uno de los arrays.*

**Desventajas de la técnica.**

Pero implica un aumento en la cantidad de espacio requerida, porque se ocupa un indice y “se pone a un lado además del fichero clasificado a sí mismo”.

La inserción en una tabla secuencial indexada es un poco más difícil debido a que puede qe no exista espacio entre dos entradas en la tabla, siendo necesario mover un gran número de elementos en la tabla.

El uso de una lista ligada (indice) da una gran sobrecarga de espacio y tiempo para los apuntadores que se utilizan en la busqueda de registros.

Los registros deben ser de longitud fija. El archivo debe estar soportado por una memoria de masa tal como el disco; no se utiliza en cinta magnética. A veces todo el archivo debe estar presente en línea.

**Principales Aplicaciones.**

Un uso en la cual esta busqueda se aplica, es donde se presenta el ingreso de datos (registros) sin ningun tipo de orden especifico; pero en cada determinado momento su campo llave es almacenado en un indice, en el cual esas llaves estan ordenadas de menor a mayor o de mayor a menor dependiendo el uso que se le de. De esta manera, para agilizar la busqueda de un registro en particular se accesa a ese registro por medio de su campo llave alacenado en el indice.

Un ejemplo de nuestra vida diaria y donde se aplica esta busqueda es en un negocio mediano (negocio de carnes frias , refaccionaria), ya que aquí se necesita una busqueda eficiente con una sola clave de acceso y otorgandonos la informacion requerida.

Ordenamiento Quicksort

Quicksort en acción sobre una lista de números aleatorios. Las líneas horizontales son valores **pivote**.

El **ordenamiento rápido** (**quicksort** en [inglés](http://es.wikipedia.org/wiki/Idioma_ingl%C3%A9s)) es un [algoritmo](http://es.wikipedia.org/wiki/Algoritmo) creado por el científico británico en computación [C. A. R. Hoare](http://es.wikipedia.org/wiki/C._A._R._Hoare) basado en la técnica de [divide y vencerás](http://es.wikipedia.org/wiki/Divide_y_vencer%C3%A1s), que permite, en promedio, [ordenar](http://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento) *n* elementos en un tiempo proporcional a*n* log *n*.

Descripción del algoritmo

El algoritmo trabaja de la siguiente forma:

* Elegir un elemento de la lista de elementos a ordenar, al que llamaremos **pivote**.
* Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
* La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
* Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

* En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño. En este caso, el orden de complejidad del algoritmo es [**O**](http://es.wikipedia.org/wiki/Cota_superior_asint%C3%B3tica)**(n·log n)**.
* En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de **O(n²)**. El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas. Pero principalmente depende del pivote, si por ejemplo el algoritmo implementado toma como pivote siempre el primer elemento del [array](http://es.wikipedia.org/wiki/Array" \o "Array), y el array que le pasamos está ordenado, siempre va a generar a su izquierda un array vacío, lo que es ineficiente.
* En el caso promedio, el orden es **O(n·log n)**.

No es extraño, pues, que la mayoría de optimizaciones que se aplican al algoritmo se centren en la elección del **pivote**.

[[editar](http://es.wikipedia.org/w/index.php?title=Quicksort&action=edit&section=2)]**Demostración de un caso particular**

[[editar](http://es.wikipedia.org/w/index.php?title=Quicksort&action=edit&section=3" \o "Editar sección: Algoritmo en C)]**Algoritmo en**[**C**](http://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n))

int colocar(int \*v, int b, int t)

{

int i;

int pivote, valor\_pivote;

int temp;

pivote = b;

valor\_pivote = v[pivote];

for (i=b+1; i<=t; i++){

if (v[i] < valor\_pivote){

pivote++;

temp=v[i];

v[i]=v[pivote];

v[pivote]=temp;

}

}

temp=v[b];

v[b]=v[pivote];

v[pivote]=temp;

return pivote;

}

void Quicksort(int\* v, int b, int t)

{

int pivote;

if(b < t){

pivote=colocar(v, b, t);

Quicksort(v, b, pivote-1);

Quicksort(v, pivote+1, t);

}

}

Supongamos que el número de elementos a ordenar es una potencia de dos, es decir, n = 2^k para algún natural k. Inmediatamente k = log_2(n), donde k es el número de divisiones que realizará el algoritmo.

En la primera fase del algoritmo habrá n comparaciones. En la segunda fase el algoritmo instanciará dos sublistas de tamaño aproximadamente n/2. El número total de comparaciones de estas dos sublistas es: 2(n/2) = n. En la tercera fase el algoritmo procesará 4 sublistas más, por tanto el número total de comparaciones en esta fase es 4(n/4) = n.

En conclusión, el número total de comparaciones que hace el algoritmo es:

n + n + n + .....+n = kn, donde k = log_2(n), por tanto el tiempo de ejecución del algoritmo en el mejor caso es **O(n.log_2 n)**.

[[editar](http://es.wikipedia.org/w/index.php?title=Quicksort&action=edit&section=4)]**Técnicas de elección del pivote**

El algoritmo básico del método Quicksort consiste en tomar cualquier elemento de la lista al cual denominaremos como pivote, dependiendo de la partición en que se elija, el algoritmo será más o menos eficiente.

* Tomar un elemento cualquiera como pivote tiene la ventaja de no requerir ningún cálculo adicional, lo cual lo hace bastante rápido. Sin embargo, esta elección «a ciegas» siempre provoca que el algoritmo tenga un orden de O(n²) para ciertas permutaciones de los elementos en la lista.
* Otra opción puede ser recorrer la lista para saber de antemano qué elemento ocupará la posición central de la lista, para elegirlo como pivote. Esto puede hacerse en **O(n)** y asegura que hasta en el peor de los casos, el algoritmo sea **O(n·log n)**. No obstante, el cálculo adicional rebaja bastante la eficiencia del algoritmo en el caso promedio.
* La opción a medio camino es tomar tres elementos de la lista - por ejemplo, el primero, el segundo, y el último - y compararlos, eligiendo el valor del medio como pivote.

[[editar](http://es.wikipedia.org/w/index.php?title=Quicksort&action=edit&section=5)]**Técnicas de reposicionamiento**

Una idea preliminar para ubicar el **pivote** en su posición final sería contar la cantidad de elementos menores que él, y colocarlo un lugar más arriba, moviendo luego todos esos elementos menores que él a su izquierda, para que pueda aplicarse la recursividad.

Existe, no obstante, un procedimiento mucho más efectivo. Se utilizan dos índices: **i**, al que llamaremos índice izquierdo, y **j**, al que llamaremos índice derecho. El algoritmo es el siguiente:

* Recorrer la lista simultáneamente con i y j: por la izquierda con i (desde el primer elemento), y por la derecha con j (desde el último elemento).
* Cuando lista[i] sea mayor que el pivote y lista[j] sea menor, se intercambian los elementos en esas posiciones.
* Repetir esto hasta que se crucen los índices.
* El punto en que se cruzan los índices es la posición adecuada para colocar el pivote, porque sabemos que a un lado los elementos son todos menores y al otro son todos mayores (o habrían sido intercambiados).

[[editar](http://es.wikipedia.org/w/index.php?title=Quicksort&action=edit&section=6)]**Transición a otro algoritmo**

Como se mencionó anteriormente, el algoritmo quicksort ofrece un orden de ejecución **O(n²)** para ciertas permutaciones "críticas" de los elementos de la lista, que siempre surgen cuando se elige el pivote «a ciegas». La permutación concreta depende del pivote elegido, pero suele corresponder a secuencias ordenadas. Se tiene que la probabilidad de encontrarse con una de estas secuencias es inversamente proporcional a su tamaño.

* Los últimos pases de quicksort son numerosos y ordenan cantidades pequeña de elementos. Un porcentaje medianamente alto de ellos estarán dispuestos de una manera similar al peor caso del algoritmo, volviendo a éste ineficiente. Una solución a este problema consiste en ordenar las secuencias pequeñas usando otro algoritmo. Habitualmente se aplica el [algoritmo de inserción](http://es.wikipedia.org/wiki/Insertion_sort) para secuencias de tamaño menores de 8-15 elementos.
* Pese a que en secuencias largas de elementos la probabilidad de hallarse con una configuración de elementos "crítica" es muy baja, esto no evita que sigan apareciendo (a veces, de manera intencionada). El algoritmo [introsort](http://es.wikipedia.org/w/index.php?title=Introsort&action=edit&redlink=1" \o "Introsort (aún no redactado)) es una extensión del algoritmo quicksort que resuelve este problema utilizando [heapsort](http://es.wikipedia.org/wiki/Heapsort" \o "Heapsort) en vez de quicksort cuando el número de recursiones excede al esperado.

*Nota:* Los tres parámetros de la llamada inicial a Quicksort serán array[0], 0, numero\_elementos -1, es decir, si es un array de 6 elementos array, 0, 5

[[editar](http://es.wikipedia.org/w/index.php?title=Quicksort&action=edit&section=7)]Ejemplo

En el siguiente ejemplo se marcan el pivote y los índices i y j con las letras p, i y j respectivamente.

Comenzamos con la lista completa. El elemento pivote será el 4:

5 - 3 - 7 - 6 - 2 - 1 - 4

p

Comparamos con el 5 por la izquierda y el 1 por la derecha.

5 - 3 - 7 - 6 - 2 - 1 - 4

i j p

5 es mayor que 4 y 1 es menor. Intercambiamos:

1 - 3 - 7 - 6 - 2 - 5 - 4

i j p

Avanzamos por la izquierda y la derecha:

1 - 3 - 7 - 6 - 2 - 5 - 4

i j p

3 es menor que 4: avanzamos por la izquierda. 2 es menor que 4: nos mantenemos ahí.

1 - 3 - 7 - 6 - 2 - 5 - 4

i j p

7 es mayor que 4 y 2 es menor: intercambiamos.

1 - 3 - 2 - 6 - 7 - 5 - 4

i j p

Avanzamos por ambos lados:

1 - 3 - 2 - 6 - 7 - 5 - 4

iyj p

En este momento termina el ciclo principal, porque los índices se cruzaron. Ahora intercambiamos lista[i] con lista[sup] (pasos 16-18):

1 - 3 - 2 - 4 - 7 - 5 - 6

p

Aplicamos recursivamente a la sublista de la izquierda (índices 0 - 2). Tenemos lo siguiente:

1 - 3 - 2

1 es menor que 2: avanzamos por la izquierda. 3 es mayor: avanzamos por la derecha. Como se intercambiaron los índices termina el ciclo. Se intercambia lista[i] con lista[sup]:

1 - 2 - 3

El mismo procedimiento se aplicará a la otra sublista. Al finalizar y unir todas las sublistas queda la lista inicial ordenada en forma ascendente.

1 - 2 - 3 - 4 - 5 - 6 - 7

**[**[**editar**](http://es.wikipedia.org/w/index.php?title=Quicksort&action=edit&section=8)**]**