Contexto Problemático

Un gran banco desea desarrollar un software que modele el funcionamiento de una de sus sedes con mayor flujo de personas. Para ello, lo ha contratado a usted y a su equipo de trabajo con el objetivo de construir un programa capaz de solventar todas las necesidades del cliente, entre las cuales se encuentran el proceso de turnos al momento del ingreso (fila para clientes y fila para personas con diferentes prioridades), manejo de tablas de datos, entre otros.

Con el fin de llevar a cabo procesos estadísticos y de agilizar el servicio de atención, esta sede bancaria registra el nombre y la cédula de todos los usuarios que ingresan al establecimiento a la hora de obtener su turno. Con ello no sólo se busca en cuál de las dos filas ubicar a la persona, sino que permite que la persona encargada de la atención de dicho usuario pueda buscarlo de manera eficiente en la base de datos y obtener toda su información antes de que este llegue a su despacho. Los datos que encontrará el encargado serán: nombre, cédula, cuenta bancaria, tarjetas de débito/crédito, fecha de pago de la tarjeta de crédito y fecha en que se incorporó al banco. Dicha información deberá mostrarse en pantalla.

Una vez sea el turno de atender al usuario, éste podrá realizar una o más de las siguientes operaciones:

a) **Retiro/consignación:** el usuario podrá modificar el monto de su cuenta de ahorros al solicitar un retiro o consignación.

b) **Cancelación de cuenta:** borra sus datos de la base de datos de clientes y los incorpora a una exclusiva para aquellos que desertan de dicho banco. Asimismo, se guardará tanto la fecha como el motivo de cancelación.

c) **Pago de tarjeta:** el usuario podrá pagar el monto utilizado con la tarjeta de crédito hasta el momento. Puede realizar el pago en efectivo o a través de su cuenta de ahorros.

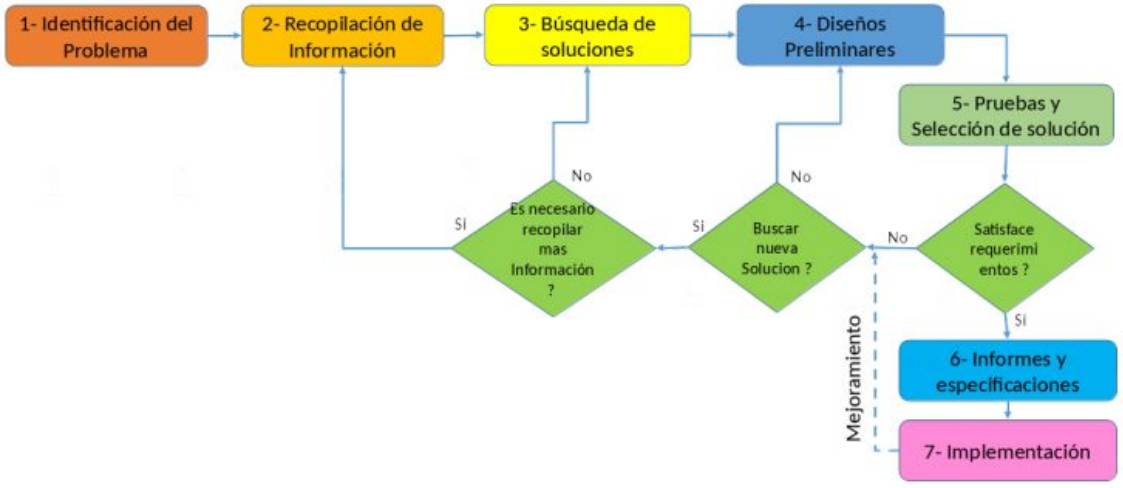
La información de todos los usuarios presentes en la sede bancaria deberá visualizarse en una tabla tipo hoja de cálculo y podrá ser organizada respecto a cuatro (4) parámetros de su escogencia. Con el objetivo de encontrar el algoritmo más óptimo

para el problema, el gerente le ha solicitado implementar un método de ordenamiento de su parecer por cada uno de los parámetro escogidos, con la restricción de que solamente uno (1) de ellos puede tener complejidad temporal promedio de (n2).

Teniendo en cuenta los errores humanos que se introducen por parte de los cajeros ya sea por equivocaciones propias al digitar la solicitud del cliente, se le solicita al equipo de ingenieros agregar una funcionalidad de *undo* para poder deshacer las equivocaciones, incluso después de haberlas guardado.

Desarrollo de la Solución

Para resolver la situación anterior se eligió el Método de la Ingeniería para desarrollar la solución siguiendo un enfoque sistemático y acorde con la situación problemática planteada. Con base en la descripción del Método de la Ingeniería del libro "Introduction to Engineering" de Paul Wright, se definió el siguiente diagrama de flujo, cuyos pasos seguiremos en el desarrollo de la solución.



Paso1. Identificación del problema

Identificación del problema:

El gran banco requiere de un software para mejorar el flujo de clientes en sus sedes para agilizar de esta forma el servicio de atención y llevar procesos estadísticos.

Identificación de necesidades y síntomas:

- ***Registrar un usuario en el banco*:** registra el nombre y la cédula de todos los usuarios que ingresan al establecimiento a la hora de obtener su turno.

- ***Buscar Usuario:*** permite que la persona encargada de la atención de dicho usuario pueda buscarlo de manera eficiente en la base de datos y obtener toda su información antes de que este llegue a su despacho. (nombre, cédula, cuenta bancaria, tarjetas de débito/crédito, fecha de pago de la tarjeta de crédito y fecha en que se incorporó al banco)
- ***Retiro/consignación*****:** el usuario podrá modificar el monto de su cuenta de ahorros al solicitar un retiro o consignación.
- ***Cancelación de cuenta*****:** borra sus datos de la base de datos de clientes y los incorpora a una exclusiva para aquellos que desertan de dicho banco. Asimismo, se guardará tanto la fecha como el motivo de cancelación.
- ***Pago de tarjeta:*** el usuario podrá pagar el monto utilizado con la tarjeta de crédito hasta el momento. Puede realizar el pago en efectivo o a través de su cuenta de ahorros.
- ***Undo:*** permite deshacer los errores en las transacciones incluso después de haberse guardado.
- Solo uno de los algoritmos de ordenamiento puede ser de complejidad temporal n^2 todos los demás deben ser menores.
- La información de todos los usuarios presentes en la sede bancaria deberá visualizarse en una tabla tipo hoja de cálculo.

Paso 2. Recopilación de información.

Para cumplir con las necesidades del gran banco es necesario recopilar toda la información posible sobre los métodos más óptimos de ordenamiento de datos para cumplir así con las expectativas de los requerimientos y dar una solución efectiva y eficaz al problema.

Definiciones

¿Qué es ordenamiento?

En computación y matemáticas un algoritmo de ordenamiento es un algoritmo que pone elementos de una lista o un vector en una secuencia dada por una relación de orden, es decir, el resultado de salida ha de ser una permutación —o reordenamiento— de la entrada que satisfaga la relación de orden dada. Las relaciones de orden más usadas son el orden numérico y el orden lexicográfico. Ordenamientos eficientes son importantes para optimizar el uso de otros algoritmos (como los de búsqueda y fusión) que requieren

listas ordenadas para una ejecución rápida. También es útil para poner datos en forma canónica y para generar resultados legibles por humanos.

Es la operación de arreglar los registros de una tabla en algún orden secuencial de acuerdo a un criterio de ordenamiento.

El ordenamiento se efectúa con base en el valor de algún campo en un registro.

El propósito principal de un ordenamiento es el de facilitar las búsquedas de los miembros del conjunto ordenado.

Tipos de ordenamientos:

Los 2 tipos de ordenamientos que se pueden realizar son: los internos y los externos.

Los internos:

Son aquellos en los que los valores a ordenar están en memoria principal, por lo que se asume que el tiempo que se requiere para acceder cualquier elemento sea el mismo (a[1], a[500], etc).

Los externos:

Son aquellos en los que los valores a ordenar están en memoria secundaria (disco, cinta, cilindro magnético, etc), por lo que se asume que el tiempo que se requiere para acceder a cualquier elemento depende de la última posición accesada (posición 1, posición 500, etc).

Métodos de ordenamiento

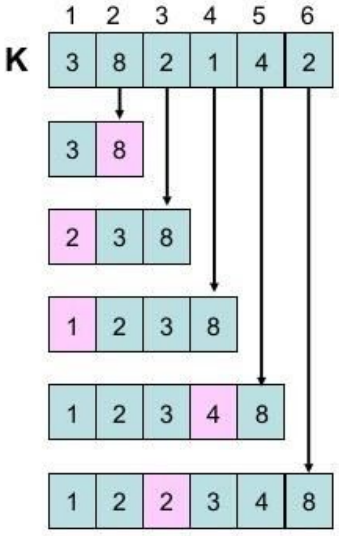
Método de inserción.

Este método toma cada elemento del arreglo para ser ordenado y lo compara con los que se encuentran en posiciones anteriores a la de él dentro del arreglo. Si resulta que el elemento con el que se está comparando es mayor que el elemento a ordenar, se recorre hacia la siguiente posición superior. Si por el contrario, resulta que el elemento con el que se está comparando es menor que el elemento a ordenar, se detiene el proceso de comparación pues se encontró que el elemento ya está ordenado y se coloca en su posición (que es la siguiente a la del último número con el que se comparó).

Ordenamiento por inserción directa

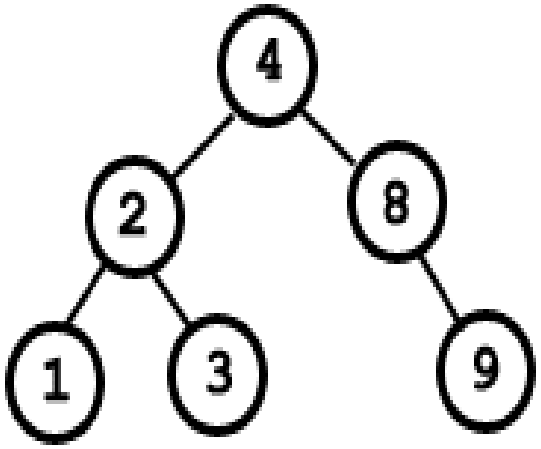
Variables
- K arreglo de datos a ordenar
- V variable auxiliar
- i, j índices para el arreglo
- N número de elementos

```
InserciónDirecta
Inicio
   Para i=2 hasta N incremento 1
     v = K(i)   //elemento a acomodar
      j = i
      Mientras (j > 1) y (K(j-1) > v)
           K(j) = K(j-1)  //mueve elementos
           j = j-1
     K(j) = v   // inserta el elemento actual
Fin
```



Ordenamiento con árbol binario

El ordenamiento con árbol binario es un algoritmo de ordenamiento, el cual ordena sus elementos haciendo uso de un árbol binario de búsqueda. Se basa en ir construyendo poco a poco el árbol binario introduciendo cada uno de los elementos, los cuales quedarán ya ordenados. Después, se obtiene la lista de los elementos ordenados recorriendo el árbol en inorden.



Método de ordenamiento por mezcla (Divide y vencerás)

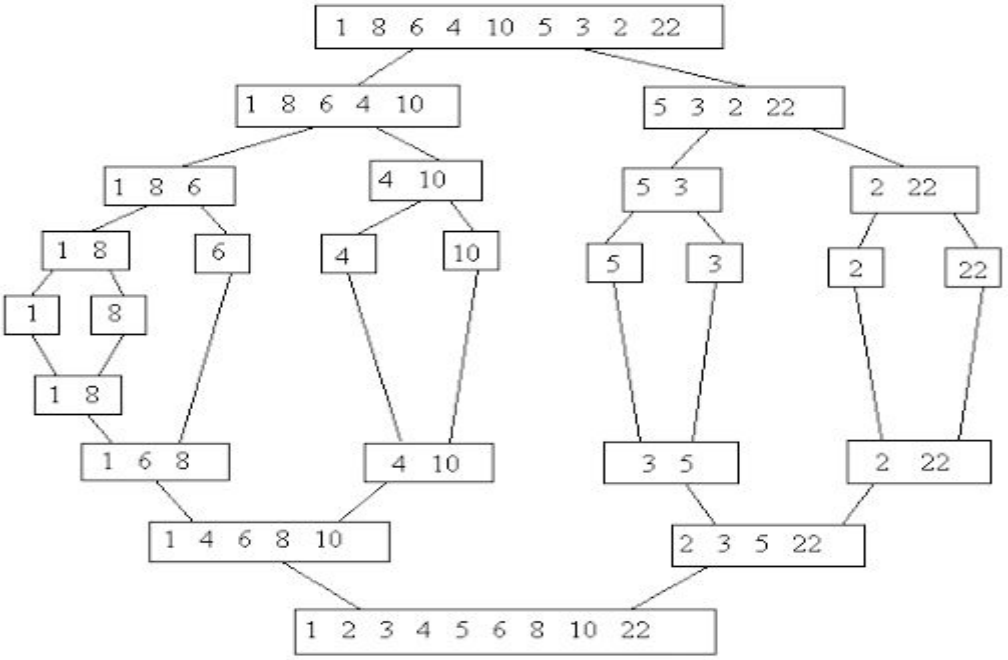
Consiste en dividir el problema a resolver en subproblemas del mismo tipo que a su vez se dividirán, mientras no sean suficientemente pequeños o triviales.

Conceptualmente, el ordenamiento por mezcla funciona de la siguiente manera:

1. Si la longitud de la lista es 0 ó 1, entonces ya está ordenada. En otro caso:
2. Dividir la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño.
3. Ordenar cada sublista recursivamente aplicando el ordenamiento por mezcla.
4. Mezclar las dos sublistas en una sola lista ordenada.

El ordenamiento por mezcla incorpora dos ideas principales para mejorar su tiempo de ejecución:

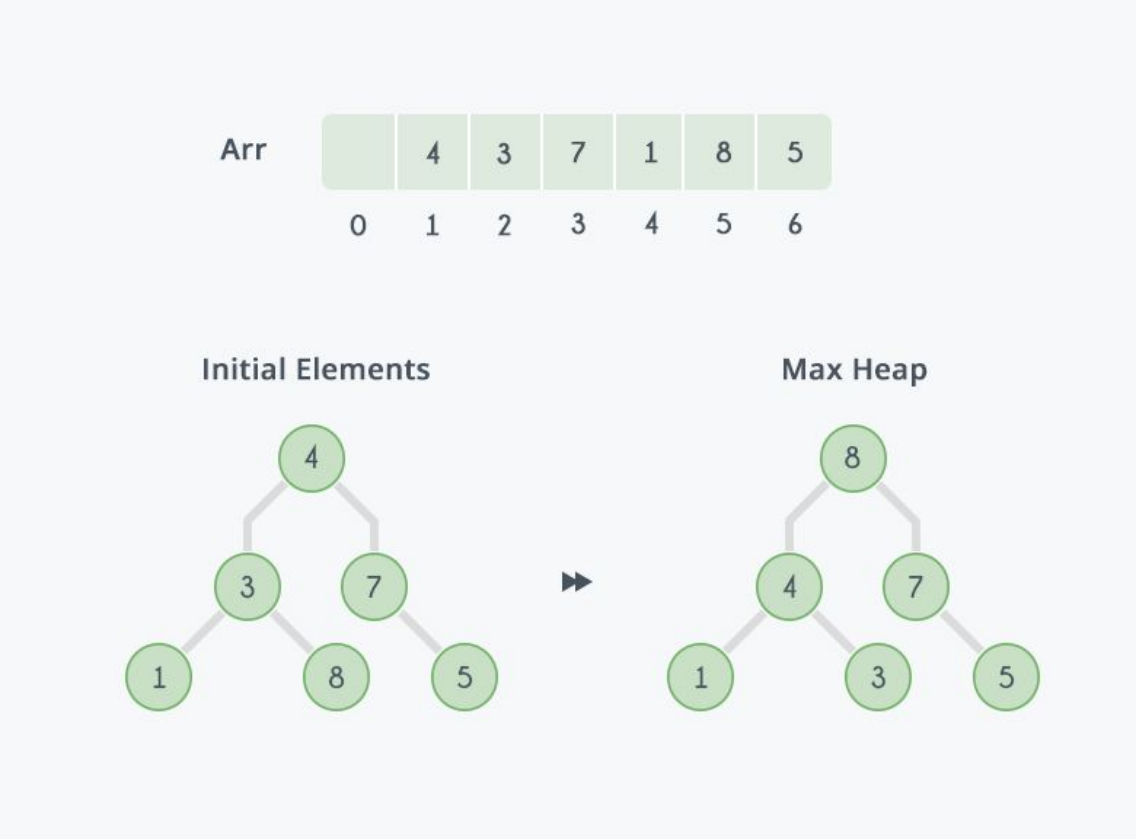
1. Una lista pequeña necesitará menos pasos para ordenarse que una lista grande.
2. Se necesitan menos pasos para construir una lista ordenada a partir de dos listas también ordenadas, que a partir de dos listas desordenadas. Por ejemplo, sólo será necesario entrelazar cada lista una vez que están ordenadas.



Ordenamiento por montículos

Es un algoritmo de ordenación no recursivo, no estable, con complejidad computacional O (n log n).

Este algoritmo consiste en almacenar todos los elementos del vector a ordenar en un montículo (heap), y luego extraer el nodo que queda como nodo raíz del montículo (cima) en sucesivas iteraciones obteniendo el conjunto ordenado. Basa su funcionamiento en una propiedad de los montículos, por la cual, la cima contiene siempre el menor elemento (o el mayor, según se haya definido el montículo) de todos los almacenados en él.

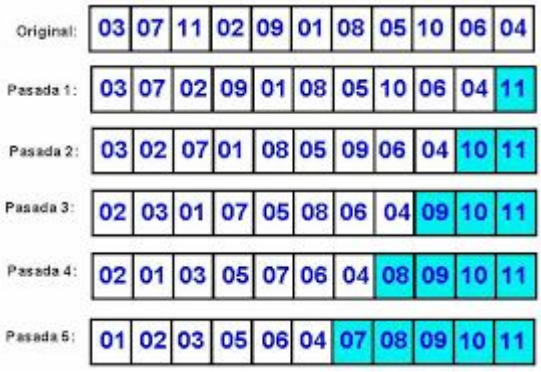


Ordenamiento Burbuja:

Es un algoritmo de ordenamiento no recursivo, con una complejidad temporal $O(n^2)$

Consiste en comparar pares de elementos adyacentes en un array y si están desordenados intercambiarlos hasta que estén todos ordenados.

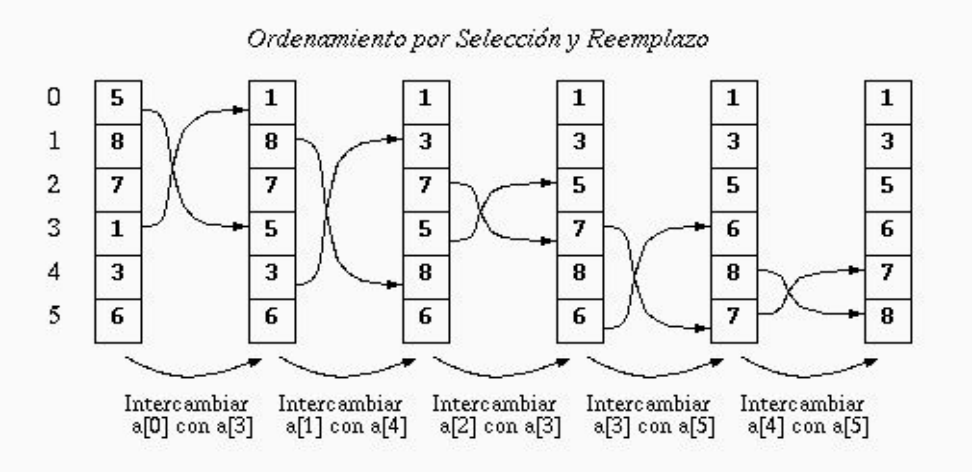
Si A es el array a ordenar, se realizan A.length-1 pasadas. Si la variable i es la que cuenta el número de pasadas, en cada pasada i se comprueban los elementos adyacentes desde el primero hasta A.length-i-1 ya que el resto hasta el final del array están ya ordenados. Si los elementos adyacentes están desordenados se intercambian.



http://puntocomnoesunlenguaje.blogspot.com/2012/07/metodo-de-ordenacion-burbuja.html

Ordenamiento de Selección.

Es un ordenamiento no recursivo que busca el primer elemento más pequeño del arreglo y lo intercambia con el primer elemento del arreglo.Luego, busca el segundo más pequeño del arreglo y lo intercambia con el segundo elemento, y así procede hasta llegar al último elemento.



https://users.dcc.uchile.cl/~lmateu/CC10A/Apuntes/ordenamiento/seleccion.gif

Paso 3. Búsqueda de soluciones creativas.

Para este problema no solo hemos usado los de ordenamientos conocidos por el equipo de desarrolladores de software sino que hemos buscado otros métodos de ordenamiento que nos ayudarán a desarrollar una solución más eficiente.

Dado que en la especificación del problema sólo podemos tener un ordenamiento con complejidad temporal O(n^2), empezamos descartando los métodos que tienen una complejidad igual o mayor a la nombrada anteriormente.

Siguiendo este orden de ideas encontramos las siguientes alternativas para la solución del problema.

Alternativa 1:

Nuestro primer conjunto de métodos de ordenamiento es el siguiente teniendo en cuenta que solo 1 de los métodos de ordenamiento puede ser de complejidad temporal O(n^2):

- Método Burbuja
- Divide y vencerás
- Búsqueda de Árbol Binario
- Ordenamiento por montículos

Alternativa 2:

Nuestro segundo conjunto de métodos de ordenamiento es el siguiente teniendo en cuenta que solo 1 de los métodos de ordenamiento puede ser de complejidad temporal O(n^2):

- Método de Selección
- Divide y vencerás
- Búsqueda de Árbol Binario
- Ordenamiento por montículos

Alternativa 3:

Nuestro tercer conjunto de métodos de ordenamiento es el siguiente teniendo en cuenta que solo 1 de los métodos de ordenamiento puede ser de complejidad temporal (n^2):

- QuickSort
- Divide y vencerás
- Búsqueda de Árbol Binario
- Ordenamiento por montículos

Paso 4. Transición de la formulación de ideas a los diseños preliminares

Por comodidad descartamos la alternativa 1 y 3 debido a que sus métodos (n^2) son iguales en el peor de los casos en el mejor de los casos no son tan eficiente como el método de ordenamiento por selección en el mejor de los casos (esto es totalmente arbitrario pero nos parece una razón más que suficiente para usar esta alternativa).

La revisión de las alternativas más minuciosamente sería la siguiente:

Alternativa 1:

Este conjunto de métodos de ordenamiento se usa como opción para el método (n^2) el método de ordenamiento burbuja que es el método de ordenamiento menos eficiente de las opciones (n^2) disponible para este software, es debido a que en el mejor y el peor de los casos en el algoritmo de ordenamiento va tener una complejidad temporal O(n^2).

Alternativa 3:

Este conjunto de métodos de ordenamiento escogió como (n^2) fue el método QuickSort, es un poco menos eficiente que el método selección debido a que en cuanto a complejidad espacial el QuickSort es O(log n) a comparación del selección que es O(1)

Selección de la mejor solución

En este paso definimos unos criterios por los cuales calificar para asi poder darle un valor numérico a cada alternativa y con base en esto reconsiderar cuál sería mejor para la solución del problema.

Criterio A: Complejidad en el caso promedio de su algoritmo O(n^2)

- **[1]** es muy parecido a n^2 a en el caso promedio
- **[2]** es un poco más eficiente que n^2 en el caso promedio
- **[3]** es mucho más eficiente que n^2 en el caso promedio

Criterio B: Complejidad espacial

-**[2]** Su complejidad espacial es mayor a O(1)

-**[1]** Su complejidad espacial es mucho mayor a O(1)

-**[3]** Su complejidad espacial es O(1)

Criterio C: Complejidad de implementación

- **[3]** fácil, es decir no resulta difícil implementar esto en el software

- **[2]** normal, es medianamente difícil de implementar

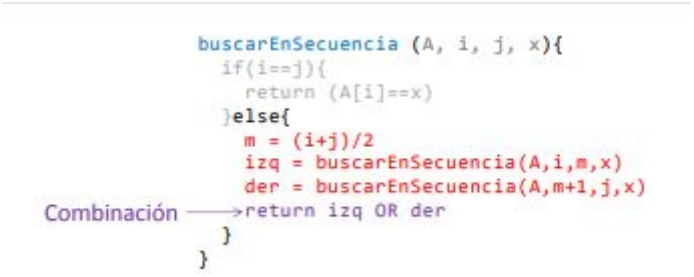
- **[1]** difícil, resulta difícil o complejo de implementar

Alternativa	Criterio A	Criterio B	Criterio C	Puntuación
Alternativa 1	**1**	**3**	**2**	**6**
Alternativa 2	**1**	**3**	**3**	**7**
Alternativa 3	**2**	**3**	**1**	**6**

De acuerdo con la tabla anterior la mejor alternativa es la *Alternativa 2* ya que según los criterios de evaluación obtuvo el mayor puntaje por lo tanto será la opción que se llevará a cabo.

Paso 6. Preparación de informes y especificaciones

El método del divide y vencerás para buscar un elemento de una lista de elemento de manera recursiva se puede ilustrar de la siguiente manera



Tomado de las Slides del profesor Andres Alberto Aristizabal Pinzon

Para el ordenamiento del divide y vencerás se debe tener en cuenta el siguiente método que será responsable de combinar las listas divididas

```
MERGE(A, p, q, r)
 1  n_1 = q − p + 1
 2  n_2 = r − q
 3  let L[1..n_1 + 1] and R[1..n_2 + 1] be new arrays
 4  for i = 1 to n_1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n_2
 7      R[j] = A[q + j]
 8  L[n_1 + 1] = ∞
 9  R[n_2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

https://stackoverflow.com/questions/43163061/merge-sort-implementation-questions-in-java

El árbol binario de búsqueda es otra manera de ordenar los datos de un arreglo es por ello que a continuación se mostrará la inserción de elementos en un árbol

Pseudocódigo

```
TREE-INSERT(T, z)
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4      y = x
 5      if z.key < x.key
 6          x = x.left
 7      else x = x.right
 8  z.p = y
 9  if y == NIL
10      T.root = z          // tree T was empty
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
```

https://www.chegg.com/homework-help/questions-and-answers/implement-binary-search-trees-java-build-trees-using-insertion-function-pseudo-code-attach-q11796035

Para el ordenamiento por montículos debemos manejar dos métodos auxiliares como se presenta en el siguiente código

For Problems #4 through #6, consider the following pseudocode (ref: CLRS pp.154–160):

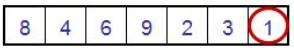
```
01 HEAPSORT(A)
02   BUILD-MAX-HEAP(A)
03   for i = A.length downto 2
04     exchange A[1] and A[i]
05     A.heapSize = A.heapSize - 1
06     MAX-HEAPIFY(A,1)

07 BUILD-MAX-HEAP(A)
08   A.heapSize = A.length
09   for i = floor(A.length/2) downto 1
10     MAX-HEAPIFY(A,i)

11 MAX-HEAPIFY(A,i)
12   L = 2*i
13   R = 2*i + 1
14   if L <= A.heapSize and A[L] > A[i]
15       largest = L
16     else
17       largest = i
18   if R <= A.heapSize and A[R] > A[largest]
19     largest = R
20   if largest != i
21     exchange A[i] and A[largest]
22     MAX-HEAPIFY(A,largest)
```

https://www.chegg.com/homework-help/questions-and-answers/heap-sort-02-build-max-heap-middot-length-downto-2-exchange-1-middot-heapsize-middot-heaps-q21279747

Por último el método de ordenamiento por selección va tener la siguiente forma

Pseudocódigo

```
SELECTION-SORT(A)
  n ← length[A]
  for j ← 1 to n - 1
      do smallest ← j
          for i ← j + 1 to n
              do if A[i] < A[smallest]
                    then smallest ← i
          exchange A[j] ↔ A[smallest]
```



http://javaexplorer03.blogspot.com/2017/05/insertion-sort-vs-selection-sort.html