# Clase 5 – Algoritmos de Búsqueda

# Temario

- Concepto de Eficiencia
- Concepto de Búsqueda
- Búsqueda lineal
  - Concepto
  - Código
- Búsqueda Binaria
  - Concepto
  - Código
- Prueba de Tiempos de Busqueda

# Eficiencia

# Eficiencia

- Una gran parte de la informática consiste en pensar algoritmos eficientes
- Un algoritmo es **eficiente** si realiza una administración correcta de los recursos del sistema en el cual se ejecuta.
- Normalmente, la eficiencia de un programa se relaciona con el **Tiempo de ejecución** y con la cantidad de **Memoria** que necesita.
- Se puede tambien relacionar con otros recursos: espacio en disco, tráfico de red que genera, etc

# Eficiencia

- **Tiempo de ejecución**: se considera mas eficiente al algoritmo que cumple con la especificación del problema en el menor tiempo posible
- **Uso de memoria**: serán mas eficientes aquellos algoritmos que utilicen las estructuras de datos adecuadas de manera de minimizar la memoria ocupada

# Algoritmos de Búsqueda

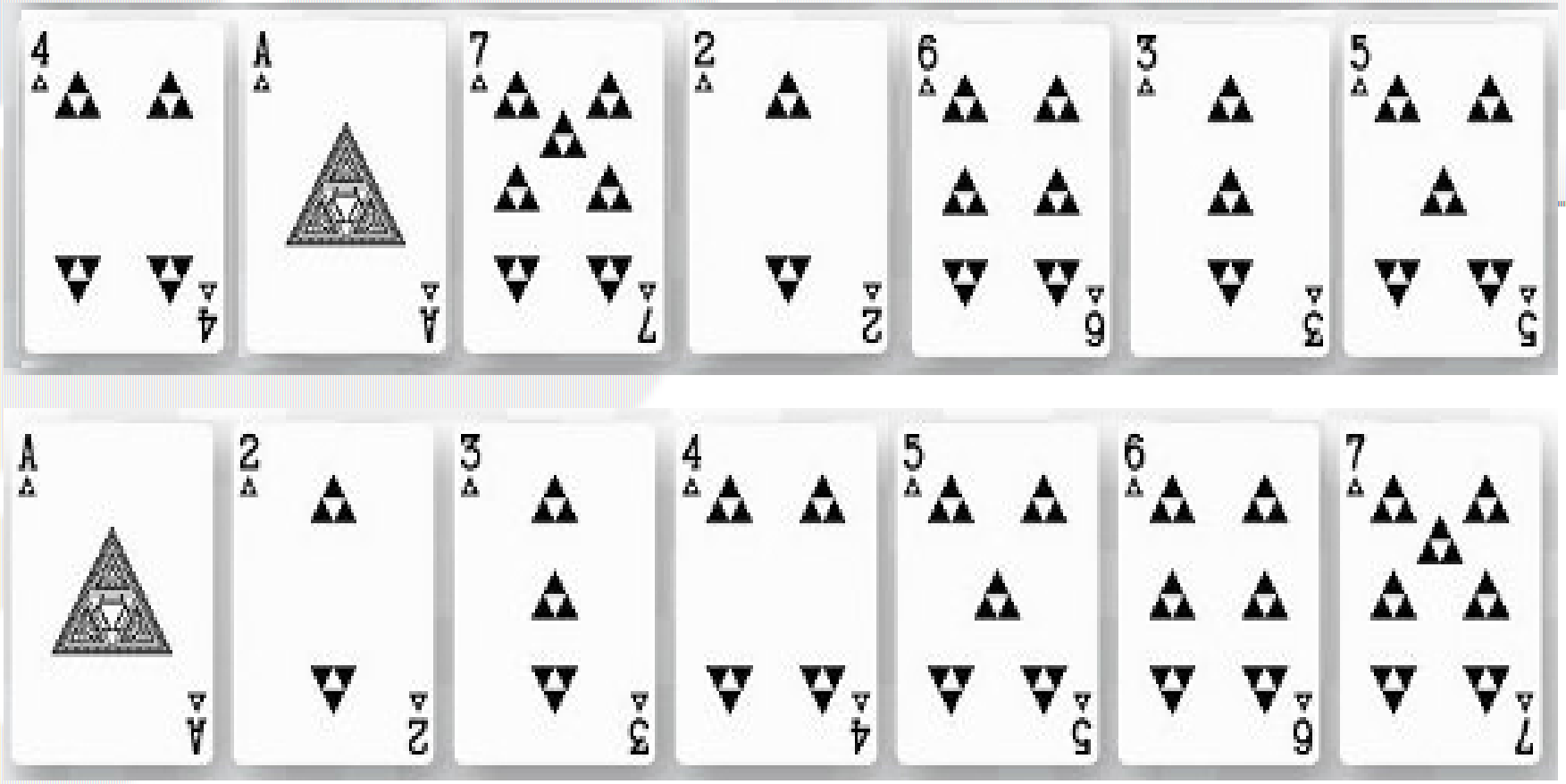
# Algoritmos de Búsqueda

- Es el proceso de ubicar información particular en una colección de datos
- En una búsqueda típica, se cuenta con una lista de ítems
- Nos interesa saber si un determinado elemento pertenece o no la colección y en caso de pertenecer, saber el **indice de la lista** en el cual esta el elemento

# El problema de la Búsqueda

Pensemos juntos, cómo sería la búsqueda de una carta particular, en cada uno de estos mazos...

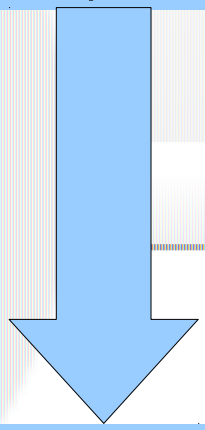
¿Cómo estan organizados cada uno de esos mazos?



# El problema de la Búsqueda

Entrada

Un número **a** y una sucesión de números L, donde
L = [a1, a2, a3, . . . . ., an]



Salida

El primer indice **i**, tal que L[i] == a ó **-1** si a no es un elemento de L

# Búsqueda Lineal

# Búsqueda Lineal

- La forma de buscar es secuencial: comenzar desde el principio hasta encontrar el elemento o llegar hasta el fin
- Los datos pueden o no estar ordenados
- Se recomienda su uso cuando los elementos están desordenados.

# Búsqueda Lineal - Código

```
def  busqueda_lineal(a, L):
    posicion = -1
    for j in range(0,len(L)):
        if a==L[j]:
            posicion = j
    return posicion
```

# Búsqueda Lineal - Mejorado

```
def  busqueda_lineal_mejorada(a, L):
    posicion = -1
    j = 0
    terminar = False
    while (j < len(L)) and not terminar:
        if a==L[j]:
            posicion = j
            terminar = True
        j = j + 1
    return posicion
```

# Búsqueda Binaria

# Búsqueda Binaria

- Es mas rápida que la búsqueda lineal
- Necesita una lista ordenada
- Es un proceso recursivo
- Consiste en reducir paulatinamente el ámbito de búsqueda a la mitad de los elementos, comparando el elemento a buscar, con el elemento que se encuentra en la mitad del intervalo. En caso de no encontrarse, tomar solo la mitad donde se encuentra el elemento.

# Búsqueda Binaria

Proceso

- Se compara el elemento buscado con el que se encuentra en el medio de la lista
- si los elementos son iguales retorna el elemento,
- si el elemento buscado es menor, se continua la búsqueda con la primer mitad de la lista
- si el elemento buscado es mayor, se continua la búsqueda con las segunda mitad de la lista

# Búsqueda Binaria – Código

```
def  busqueda_binaria(a, L):
    menor=0
    mayor=len(L)-1
    terminar = False
    posicion = -1
    while (menor <= mayor) and not terminar:
        mitad=(menor + mayor) / 2
        if L[mitad] == a:
            posicion = mitad
            terminar = True
        elif L[mitad] > a:
            mayor=mitad - 1
        elif L[mitad] < a:
            menor=mitad +1
    return posicion
```

# Búsqueda Binaria – Código recursivo

```
def busqueda_binariaRecursiva(a, L):
    menor=0
    mayor=len(L)-1
    mitad = (menor + mayor) / 2
    if L == []:
        return -1
    elif L[mitad]== a:
        return mitad
    elif L[mitad] > a:
        result = busqueda_binariaRecursiva(a,L[0:mitad])
        return result
    elif L[mitad] < a:
        result=  busqueda_binariaRecursiva(a,L[mitad+1:mayor+1])
        if(result == -1):
            return -1
        return result + mitad + 1
    return -1
```

# Tiempos de Búsqueda

# Tiempos de Búsqueda

- Lo que queremos hacer es medir los tiempos de búsqueda, para eso lo que hacemos es tomar el tiempo al inicio, luego a la salida y restamos los mismos

```
def tiempo_general(argumentos):
    t1=time.time()
    general(argumentos)
    t2=time.time()
    print(t2-t1)
```

- Donde **general** es el nombre de la función, tenemos que reemplazarlo