Lorena Yaneth Pérez Pérez
Carnet 20200396
Estructura de Datos

# Algoritmos de ordenamiento y búsqueda

**Ordenamiento:**

1. **Insertion Sort:** Se trata de un algoritmo de ordenación en el que los elementos se transfieren de uno en uno a la posición correcta. Es uno de los algoritmos más sencillos. El procedimiento consta de tomar elemento por elemento del arreglo y recorrerlo hacia su posición con respecto a los que se fueron ordenando. Y así, sigue su proceso hasta el último elemento, hasta recorrer todas las posiciones del arreglo.

   **Código:**

```
void Sortporinsercion (int A[], int N)
{
        int j,P,tmp;
                {
                        A[0]=Min int;
                        for(P=2;P<=N;P++)
                                {
                                j=P;
                                tmp=A[P];
                                while(tmp < A[j-1])
                                {
                                        A[j] = A[j-1];
                                        j-=1;
                                }
                                A[j]=tmp;
                                }
                }
}
```

2. **Shell Sort:** El Shell Sort es una generalización del ordenamiento por inserción, teniendo en cuenta dos observaciones:

a. El ordenamiento por inserción es eficiente si la entrada está "casi ordenada".
b. El ordenamiento por inserción es ineficiente, en general, porque mueve los valores solo una posición cada vez.

El algoritmo Shell Sort mejora el ordenamiento por inserción comparando elementos separados por un espacio de varias posiciones. Los pasos múltiples sobre los datos se hacen con tamaños de espacio cada vez más pequeños. El último paso del Shell Sort es un simple ordenamiento por inserción, pero para entonces, ya está garantizado que los datos del vector están casi ordenados.

**Código:**

```
void shellsort ( int a[], int n)
/* Este procedimiento recibe un arreglo a ordenar a[] y el tamaño del arreglo n. Utiliza en este caso una serie de t=6 incrementos h=[1,4,13,40,121,364] para el proceso (asumimos que el arreglo no es muy grande). */
{
        int x,i,j,inc,s;

        for(s=1; s < t; s++)   /* recorre el arreglo de incrementos */
        {
                inc = h[s];
                for(i=inc+1; i < n; i++)
                {
                        x = a[i];
                        j = i-inc;
                        while( j > 0 && a[j] > x)
                        {
                                a[j+h] = a[j];
                                j = j-h;
                        }
                        a[j+h] = x;
                }
        }
}
```

**Búsqueda:**

1. Búsqueda lineal:

   Consiste en recorrer y examinar cada uno de los elementos del array hasta encontrar el o los elementos buscados, o hasta que se han mirado todos los elementos del array.

Este es el método de búsqueda más lento, pero si nuestra información se encuentra completamente desordenada es el único que nos podrá ayudar a encontrar el dato que buscamos.

**Código:**

```
int busqueda lineal (int A[], int clave, int n)
{
        for(int i=0;i<n;i++)
                if (A[i]==clave) return i;
        return -1;
}
```

2. Búsqueda binaria:

   El algoritmo consiste en reducir paulatinamente el ámbito de búsqueda a la mitad de los elementos, basándose en comparar el elemento a buscar con el elemento que se encuentra en la mitad del intervalo y en base a esta comparación:

   a. Si el elemento buscado es menor que el elemento medio, entonces sabemos que el elemento está en la mitad inferior de la tabla.
   b. Si es mayor es porque el elemento está en la mitad superior.
   c. Si es igual se finaliza con éxito la búsqueda ya que se ha encontrado el elemento.

**Código:**

```
def busquedaBinaria(unaLista, item):
        primero = 0
        ultimo = len(unaLista)-1
        encontrado = False

        while primero<=ultimo and not encontrado:
                puntoMedio = (primero + ultimo)//2
                if unaLista[puntoMedio] == item:
                        encontrado = True
                else:
                        if item < unaLista[puntoMedio]:
                                ultimo = puntoMedio-1
                        else:
                                primero = puntoMedio+1
        return encontrado
listaPrueba = [0, 1, 2, 8, 13, 17, 19, 32, 42]
```