UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Tarea 06: Ordenamientos I

ALUMNOS:

Castañon Maldonado Carlos Emilio
Nepomuceno Escarcega Arizdelcy Lizbeth

PROFESOR

María de Luz Gasca Soto

AYUDANTES

Brenda Margarita Becerra Ruíz
Enrique Ehecatl Hernández Ferreiro (Link)

ASIGNATURA

Análisis de Algoritmos

1 El Problema de Selección consiste en encontrar el k-ésimo elemento más pequeño de un conjunto de n datos, $1 \leq k \leq n$
Considere los algoritmos de ordenamiento:

a) Bubble Sort;
b) Insertion Sort;
c) Shell Sort;
d) Selection Sort;
e) Local Insertion Sort;

Suponiendo que k es tal que $1 < k < n$ ¿Cuáles de los algoritmos anteriores, nos ayudan a resolver el Problema de Selección sin tener que ordenar toda la secuencia? y ¿Cuál es la estrategia usada por el algoritmo? Justifica, para cada inciso, tus respuestas.

a. **Bubble Sort:** Bubble Sort no es un buen candidato para resolver el Problema de Selección sin ordenar toda la secuencia. Este algoritmo se basa en comparar y swappear elementos adyacentes repetidamente para llevar el elemento más grande a la posición final. No proporciona información sobre el k-ésimo elemento sin ordenar toda la lista.

b. **Insertion Sort:** Insertion Sort ordena la lista de elementos uno a uno. Comienza con un subconjunto ordenado e inserta cada elemento restante en la posición adecuada dentro del subconjunto ordenado. No es eficiente para encontrar el k-ésimo elemento sin ordenar toda la secuencia.

c. **Shell Sort:** Shell Sort es una mejora de Insertion Sort que utiliza intervalos de comparación para realizar una clasificación más eficiente. Divide la lista en subgrupos y los ordena independientemente. No está diseñado para encontrar el k-ésimo elemento sin ordenar toda la lista.

d. **Selection Sort:** Selection Sort busca el elemento más pequeño en la lista y lo intercambia con el primer elemento. Luego, busca el segundo elemento más pequeño y lo intercambia con el segundo elemento, y así sucesivamente. Puede encontrar el k-ésimo elemento más pequeño después de k iteraciones, pero todavía realiza algunas operaciones de ordenamiento parcial.

e. **Local Insertion Sort:** La esencia de este enfoque radica en mantener una lista enlazada L constantemente ordenada, junto con un puntero u que señala al último elemento insertado. A partir de la posición de u, se inicia la búsqueda de la ubicación adecuada para añadir el nuevo elemento. Aunque esta estrategia es efectiva para mantener una secuencia ordenada a medida que se agregan elementos, no se presta directamente para resolver el Problema de Selección, que implica encontrar el k-ésimo elemento más pequeño en una lista sin necesidad de ordenarla por completo.

2 Indicar cuáles de los algoritmos de ordenamiento mencionados en el Ejercicio 1 son estables, además mencionar si son inplace u outplace. Justificar por qué.

a) **Bubble Sort**

Es estable ya que este solo compara los elementos adyacentes (y solo los compara si se encuentran desordenados), esto infiere que los elementos que sean iguales (con posiciones diferentes en el arreglo), mantendrán su orden relativo en la lista ordenada.

Además, el presente es in-place ya que no requiere de memoria adicional para ordenar el arreglo.

b) **Insertion Sort**

Es estable ya que comparamos e intercambiamos a los elementos adyacentes para ordenar, teniendo de esta forma que el orden no se ve afectado.

Además, el presente es in-place ya que cuando ordenamos el arreglo, lo hacemos moviendo los elementos dentro del mismo arreglo, esto sin necesidad de requerir memoria adicional.

c) **Shell Sort**

Es inestable ya que como estamos utilizando incrementos para ordenar sublistas, en el proceso podría darse el caso de que los elementos puedan moverse grandes distancias, lo cual puede provocar que el orden respecto a los elementos que tengamos iguales, varié de maneras que no podamos predecir.

Además, el presente suele considerarse in-place ya que es una variación de Insertion Sort y por que estamos trabajando con múltiples sublistas dentro de el arreglo.

d) **Selection Sort**

Es inestable ya que al seleccionar el mínimo elemento de la parte no ordenada e intercambiarlo por el elemento de la otra parte, este intercambio puede cambiar el orden relativo de los elementos iguales, haciendo de esta forma que Selection Sort sea inestable.

Además, el presente es in-place ya que seleccionamos el elemento mínimo de la parte no ordenada y lo intercambiamos con el primer elemento no ordenado, todo esto dentro del arreglo original, por lo cual no requerimos de memoria adicional.

e) **Local Insertion Sort**

Es estable ya que mantiene el orden relativo de los elementos iguales al ejecutar el ordenamiento, ya que al encontrar elementos iguales, no se cambia el orden, asegurándonos de esta forma que los elementos iguales del arreglo original permanecerán en el mismo orden en el arreglo ordenado.

Además, el presente es in-place ya que estamos operando dentro del mismo arreglo, no requiriendo necesariamente de tener memoria adicional para poder funcionar forzosamente.

3 Considera el algoritmo Shell Sort

a) **Proporciona un ejemplar de al menos 35 dígitos distintos, para el cual la versión con incrementos de Shell realice menos operaciones que la versión con incrementos de Hibbard. Realiza la ejecución completa sobre el ejemplar dado.**
Shell Sort:

```
A : [101, 102, 103, 104, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 116, 117, 118, 119, 120, 19, 27, 20, 28, 21, 29, 22, 30,
23, 31, 131, 132, 133, 134, 135]
```

a)

```
A : [101, 102, 103, 104, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 116, 117, 118, 119, 120, 19, 27, 20, 28, 21, 29, 22, 30,
23, 31, 131, 132, 133, 134, 135]

i = 1
h = 17
```

S1:[101 118 135] S2:[102 119] S3:[103 120] S4:[104 19] S5:[3 27] S6:[11 20] S7:[4 28] S8:[12 21] S9:[5 29] S10:[13 22] S11:[6 30]
S12:[14 23] S13:[7 31] S14:[15 131] S15:[8 132] S16:[116 133] S17:[117 134]

S1:[101 118 135] S2:[102 119] S3:[103 120] S4:[19 104] S5:[3 27] S6:[11 20] S7:[4 28] S8:[12 21] S9:[5 29] S10:[13 22] S11:[6 30]
S12:[14 23] S13:[7 31] S14:[15 131] S15:[8 132] S16:[116 133] S17:[117 134]

b)

```
A : [101, 102, 103, 19, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 116, 117, 118, 119, 120, 104, 27, 20, 28, 21, 29, 22, 30,
23, 31, 131, 132, 133, 134, 135]

i = 2
h = 8
```

S1:[101 5 117 21 133] S2:[102 13 118 29 134] S3:[103 6 119 22 135] S4:[19 14 120 30] S5:[3 7 104 23] S6:[11 15 27 31]
S7:[4 8 20 131] S8:[12 116 28 132]

S1:[5 21 101 117 133] S2:[13 29 102 118 134] S3:[6 22 103 119 135] S4:[14 19 30 120] S5:[3 7 23 104] S6:[11 15 27 31]
S7:[4 8 20 131] S8:[12 28 116 132]

c)

```
A : [5, 13, 6, 14, 3, 11, 4, 12, 21, 29, 22, 19, 7, 15, 8, 28, 101, 102, 103, 30, 23, 27, 20, 116, 117, 118, 119, 120,
104, 31, 131, 132, 133, 134, 135]

i = 3
h = 4
```

S1:[5 3 21 7 101 23 117 104 133] S2:[13 11 29 15 102 27 118 31 134] S3:[6 4 22 8 103 20 119 131 135] S4:[14 12 19 28 30 116 120 132]
S1:[3 5 7 21 23 101 104 117 133] S2:[11 13 15 27 29 31 102 118 134] S3:[4 6 8 20 22 103 119 131 135] S4:[12 14 19 28 30 116 120 132]

d)

```
A : [3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 19, 21, 27, 20, 28, 23, 29, 22, 30, 101, 31, 103, 116, 104, 102, 119, 120,
117, 118, 131, 132, 133, 134, 135]

i = 4
h = 2
```

S1:[3 4 5 6 7 8 21 20 23 22 101 103 104 119 117 131 133 135] S2:[11 12 13 14 15 19 27 28 29 30 31 116 102 120 118 132 134]

S1:[3 4 5 6 7 8 20 21 22 23 101 103 104 117 119 131 133 135] S2:[11 12 13 14 15 19 27 28 29 30 31 102 116 118 120 132 134]

e)

```
A : [3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 19, 21, 27, 20, 28, 23, 29, 22, 30, 101, 31, 103, 116, 104, 102, 119, 120,
117, 118, 131, 132, 133, 134, 135]

i = 5
h = 1
```

S1:[3 11 4 12 5 13 6 14 7 15 8 19 20 27 21 28 22 29 23 30 101 31 103 102 104 116 117 118 119 120 131 132 133 134 135]

S1:[3 4 5 6 7 8 11 12 13 14 15 19 20 21 22 23 27 28 29 30 31 101 102 103 104 116 117 118 119 120 131 132 133 134 135]

```
A : [3 4 5 6 7 8 11 12 13 14 15 19 20 21 22 23 27 28 29 30 31 101 102 103 104 116 117 118 119 120 131 132 133 134 135]
```

Hibbard:

```
A : [101, 102, 103, 104, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 116, 117, 118, 119, 120, 19, 27, 20, 28, 21, 29, 22, 30,
23, 31, 131, 132, 133, 134, 135]
```

a)

```
A : [101, 102, 103, 104, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 116, 117, 118, 119, 120, 19, 27, 20, 28, 21, 29, 22, 30,
23, 31, 131, 132, 133, 134, 135]

i = 1
h = 31
```

S1:[101 132] S2:[102 133] S3:[103 134] S4:[104 135] S5:[3] S6:[11] S7:[4] S8:[12] S9:[5] S10:[13] S11:[6] S12:[14] S13:[7] S14:[15]
S15:[8] S16:[116] S17:[117] S18:[118] S19:[119] S20:[120] S21:[19] S22:[27] S23:[20] S24:[28] S25:[21] S26:[29] S27:[22] S28:[30]
S29:[23] S30:[31] S31:[131]

S1:[101 132] S2:[102 133] S3:[103 134] S4:[104 135] S5:[3] S6:[11] S7:[4] S8:[12] S9:[5] S10:[13] S11:[6] S12:[14] S13:[7] S14:[15]
S15:[8] S16:[116] S17:[117] S18:[118] S19:[119] S20:[120] S21:[19] S22:[27] S23:[20] S24:[28] S25:[21] S26:[29] S27:[22] S28:[30]
S29:[23] S30:[31] S31:[131]

b)

```
A : [101, 102, 103, 104, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 116, 117, 118, 119, 120, 19, 27, 20, 28, 21, 29, 22, 30,
23, 31, 131, 132, 133, 134, 135]

i = 2
h = 15
```

S1:[101 116 131] S2:[102 117 132] S3:[103 118 133] S4:[104 119 134] S5:[3 120 135] S6:[11 19] S7:[4 27] S8:[12 20] S9:[5 28] S10:[13 21]
S11:[6 29] S12:[14 22] S13:[7 30] S14:[15 23] S15:[8 31]

S1:[101 116 131] S2:[102 117 132] S3:[103 118 133] S4:[104 119 134] S5:[3 120 135] S6:[11 19] S7:[4 27] S8:[12 20] S9:[5 28] S10:[13 21]
S11:[6 29] S12:[14 22] S13:[7 30] S14:[15 23] S15:[8 31]

c)

```
A : [101, 102, 103, 104, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 116, 117, 118, 119, 120, 19, 27, 20, 28, 21, 29, 22, 30,
23, 31, 131, 132, 133, 134, 135]

i = 3
h = 7
```

S1:[101 12 8 27 23] S2:[102 5 116 20 31] S3:[103 13 117 28 131] S4:[104 6 118 21 132] S5:[3 14 119 29 133] S6:[11 7 120 22 134]
S7:[4 15 19 30 135]

S1:[8 12 23 27 101] S2:[5 20 31 102 116] S3:[13 28 103 117 131] S4:[6 21 104 118 132] S5:[3 14 29 119 133] S6:[7 11 22 120 134]
S7:[4 15 19 30 135]

d)

```
A : [8, 5, 13, 6, 3, 7, 4, 12, 20, 28, 21, 14, 11, 15, 23, 31, 103, 104, 29, 22, 19, 27, 102, 117, 118, 119, 120, 30,
101, 116, 131, 132, 133, 134, 135]

i = 4
h = 3
```

S1:[8 6 4 28 11 31 29 27 118 30 131 134] S2:[5 3 12 21 15 103 22 102 119 101 132 135] S3:[13 7 20 14 23 104 19 117 120 116 133]
S1:[4 6 8 11 27 28 29 30 31 118 131 134] S2:[3 5 12 15 21 22 101 102 103 119 132 135] S3:[7 13 14 19 20 23 104 116 117 120 133]

e)

```
A : [4, 3, 7, 6, 5, 13, 8, 12, 14, 11, 15, 19, 27, 21, 20, 28, 22, 23, 29, 101, 104, 30, 102, 116, 31, 103, 117, 118,
119, 120, 131, 132, 133, 134, 135]

i = 5
h = 1
```

S1:[4 3 7 6 5 13 8 12 14 11 15 19 27 21 20 28 22 23 29 101 104 30 102 116 31 103 117 118 119 120 131 132 133 134 135]
S1:[3 4 5 6 7 8 11 12 13 14 15 19 20 21 22 23 27 28 29 30 31 101 102 103 104 116 117 118 119 120 131 132 133 134 135]

```
A : [3 4 5 6 7 8 11 12 13 14 15 19 20 21 22 23 27 28 29 30 31 101 102 103 104 116 117 118 119 120 131 132 133 134 135]
```

b) **Proporciona un ejemplar de al menos 35 datos diferentes donde la versión con incrementos de Shell realice más operaciones que la versión con incrementos de Hibbard. Realiza la ejecución completa sobre el ejemplar dado.**
Shell Sort:

```
A : [1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 16, 17, 25, 18, 26, 19, 27, 20, 28, 21, 29, 22, 30, 23, 31, 24, 32, 33, 34, 35]
```

a)

```
A : [1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 16, 17, 25, 18, 26, 19, 27, 20, 28, 21, 29, 22, 30, 23, 31, 24, 32, 33, 34, 35]

i = 1
h = 17
```

S1:[1 25 35] S2:[9 18] S3:[2 26] S4:[10 19] S5:[3 27] S6:[11 20] S7:[4 28] S8:[12 21] S9:[5 29] S10:[13 22] S11:[6 30] S12:[14 23]
S13:[7 31] S14:[15 24] S15:[8 32] S16:[16 33] S17:[17 34]

S1:[1 25 35] S2:[9 18] S3:[2 26] S4:[10 19] S5:[3 27] S6:[11 20] S7:[4 28] S8:[12 21] S9:[5 29] S10:[13 22] S11:[6 30] S12:[14 23]
S13:[7 31] S14:[15 24] S15:[8 32] S16:[16 33] S17:[17 34]

b)

```
A : [1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 16, 17, 25, 18, 26, 19, 27, 20, 28, 21, 29, 22, 30, 23, 31, 24, 32, 33, 34, 35]

i = 2
h = 8
```

S1:[1 5 17 21 33] S2:[9 13 25 29 34] S3:[2 6 18 22 35] S4:[10 14 26 30] S5:[3 7 19 23] S6:[11 15 27 31] S7:[4 8 20 24] S8:[12 16 28 32]
S1:[1 5 17 21 33] S2:[9 13 25 29 34] S3:[2 6 18 22 35] S4:[10 14 26 30] S5:[3 7 19 23] S6:[11 15 27 31] S7:[4 8 20 24] S8:[12 16 28 32]

c)

```
A : [1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 16, 17, 25, 18, 26, 19, 27, 20, 28, 21, 29, 22, 30, 23, 31, 24, 32, 33, 34, 35]

i = 3
h = 4
```

```
S1:[1 3 5 7 17 19 21 23 33] S2:[9 11 13 15 25 27 29 31 34] S3:[2 4 6 8 18 20 22 24 35] S4:[10 12 14 16 26 28 30 32]
S1:[1 3 5 7 17 19 21 23 33] S2:[9 11 13 15 25 27 29 31 34] S3:[2 4 6 8 18 20 22 24 35] S4:[10 12 14 16 26 28 30 32]
```

d)

```
A : [1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 16, 17, 25, 18, 26, 19, 27, 20, 28, 21, 29, 22, 30, 23, 31, 24, 32, 33, 34, 35]

i = 4
h = 2
```

```
S1:[1 2 3 4 5 6 7 8 17 18 19 20 21 22 23 24 33 35] S2:[9 10 11 12 13 14 15 16 25 26 27 28 29 30 31 32 34]
S1:[1 2 3 4 5 6 7 8 17 18 19 20 21 22 23 24 33 35] S2:[9 10 11 12 13 14 15 16 25 26 27 28 29 30 31 32 34]
```

e)

```
A : [1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 16, 17, 25, 18, 26, 19, 27, 20, 28, 21, 29, 22, 30, 23, 31, 24, 32, 33, 34, 35]

i = 5
h = 1
```

```
S1:[1 9 2 10 3 11 4 12 5 13 6 14 7 15 8 16 17 25 18 26 19 27 20 28 21 29 22 30 23 31 24 32 33 34 35]
S1:[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35]
```

```
A: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35]
```

Hibbard:

```
A : [1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 16, 17, 25, 18, 26, 19, 27, 20, 28, 21, 29, 22, 30, 23, 31, 24, 32, 33, 34, 35]
```

a)

```
A : [1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 16, 17, 25, 18, 26, 19, 27, 20, 28, 21, 29, 22, 30, 23, 31, 24, 32, 33, 34, 35]

i = 1
h = 31
```

S1:[1 32] S2:[9 33] S3:[2 34] S4:[10 35] S5:[3] S6:[11] S7:[4] S8:[12] S9:[5] S10:[13] S11:[6] S12:[14] S13:[7] S14:[15] S15:[8] S16:[16]
S17:[17] S18:[25] S19:[18] S20:[26] S21:[19] S22:[27] S23:[20] S24:[28] S25:[21] S26:[29] S27:[22] S28:[30] S29:[23] S30:[31] S31:[24]

S1:[1 32] S2:[9 33] S3:[2 34] S4:[10 35] S5:[3] S6:[11] S7:[4] S8:[12] S9:[5] S10:[13] S11:[6] S12:[14] S13:[7] S14:[15] S15:[8] S16:[16]
S17:[17] S18:[25] S19:[18] S20:[26] S21:[19] S22:[27] S23:[20] S24:[28] S25:[21] S26:[29] S27:[22] S28:[30] S29:[23] S30:[31] S31:[24]

b)

```
A : [1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 16, 17, 25, 18, 26, 19, 27, 20, 28, 21, 29, 22, 30, 23, 31, 24, 32, 33, 34, 35]

i = 2
h = 15
```

S1:[1 16 24] S2:[9 17 32] S3:[2 25 33] S4:[10 18 34] S5:[3 26 35] S6:[11 19] S7:[4 27] S8:[12 20] S9:[5 28] S10:[13 21] S11:[6 29]
S12:[14 22] S13:[7 30] S14:[15 23] S15:[8 31]

S1:[1 16 24] S2:[9 17 32] S3:[2 25 33] S4:[10 18 34] S5:[3 26 35] S6:[11 19] S7:[4 27] S8:[12 20] S9:[5 28] S10:[13 21] S11:[6 29]
S12:[14 22] S13:[7 30] S14:[15 23] S15:[8 31]

c)

```
A : [1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15, 8, 16, 17, 25, 18, 26, 19, 27, 20, 28, 21, 29, 22, 30, 23, 31, 24, 32, 33, 34, 35]

i = 3
h = 7
```

S1:[1 12 8 27 23] S2:[9 5 16 20 31] S3:[2 13 17 28 24] S4:[10 6 25 21 32] S5:[3 14 18 29 33] S6:[11 7 26 22 34] S7:[4 15 19 30 35]
S1:[1 8 12 23 27] S2:[5 9 16 20 31] S3:[2 13 17 24 28] S4:[6 10 21 25 32] S5:[3 14 18 29 33] S6:[7 11 22 26 34] S7:[4 15 19 30 35]

d)

```
A : [1, 5, 2, 6, 3, 7, 4, 8, 9, 13, 10, 14, 11, 15, 12, 16, 17, 21, 18, 22, 19, 23, 20, 24, 25, 29, 26, 30, 27, 31, 28, 32, 33, 34, 35]

i = 4
h = 3
```

```
S1:[1 6 4 13 11 16 18 23 25 30 28 34] S2:[5 3 8 10 15 17 22 20 29 27 32 35] S3:[2 7 9 14 12 21 19 24 26 31 33]
S1:[1 4 6 11 13 16 18 23 25 28 30 34] S2:[3 5 8 10 15 17 20 22 27 29 32 35] S3:[2 7 9 12 14 19 21 24 26 31 33]
```

e)

```
A : [1, 5, 2, 6, 3, 7, 4, 8, 9, 13, 10, 14, 11, 15, 12, 16, 17, 21, 18, 22, 19, 23, 20, 24, 25, 29, 26, 30, 27, 31, 28, 32, 33, 34, 35]

i = 5
h = 1
```

```
S1:[1 3 2 4 5 7 6 8 9 11 10 12 13 15 14 16 17 19 18 20 21 23 22 24 25 27 26 28 29 31 30 32 33 34 35]
S1:[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35]
```

```
A: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35]
```

4 Realiza la siguiente modificación al algoritmo Insertion Sort: Para buscar la posición del nuevo elemento, el que está en revisión, usar Búsqueda Binaria, en vez de hacerlo secuencialmente.

a) Determina el desempeño computacional del algoritmo modificado.

Notemos que en nuestro algoritmo modificado que para cada índice i del arreglo con $2 \leq i \leq n$ haremos búsqueda binaria en un arreglo de tamaño i, además en el peor de los casos tendríamos un arreglo ordenado de forma descendente tendríamos que búsqueda binaria tomaría al menos $log(i)$ operaciones, ya que el elemento en el peor de los casos podría ir en la primera posición del arreglo, lo que implicaría que tendríamos que mover el resto de elementos a la derecha, esto quiere decir que para cada j con $0 \leq j \leq i$ moveríamos el elemento de la posición j a la posición $j+1$, por lo que para cada elemento tendríamos que aplicar al menos $i + log(i)$ operaciones con $2 \leq i \leq n$, por lo que el número total de operaciones sería:

$$\sum_{i=2}^{n}(\log(i) + i) \geq \frac{1}{2}(n^2 + n - 2) + log(n-1) \tag{1}$$

Lo cuál es $O(n^2)$.

b) ¿Mejora el desempeño computacional del proceso total? Justifica.

No mejora el desempeño, ya que daunque ambos algoritmos son $O(n^2)$, el algoritmo modificado tarda un poco más, ya que en el peor de los casos este toma $\frac{1}{2}(n^2 - n)$ operaciones, lo cuál es menor a las $\frac{1}{2}(n^2 + n - 2) + log(n-1)$ que toma como mínimo el algoritmo modificado.

```
// Precondiciones: X[a,b] es un arreglo ordenado no vacío y finito de enteros,
// a<=b+1 el arreglo tiene al menos un elemento.
// Postcondiciones: X no sufre cambios.
// regresa el índice donde debe estar el elemento

BusquedaBinIn(array X, int a, int b, int z){
  int mid = (a+b) div 2; // Calcula el índice medio entre a y b.
// Comprueba si z está en el rango entre X[mid] y X[mid+1].
  if((z > X[mid] && z <= X[mid+1]) || (z >= X[mid] && z <= X[mid+1])
  || (z >= X[mid] && z < X[mid+1]) ){
    return mid+1; // Devuelve el índice donde debería estar el elemento z.
  }else if(z < X[mid]){
    return BusquedaBinInAux(X, a, mid-1, z); // Llama a la búsqueda en la mitad inferior.
  }else{
    return BusquedaBinInAux(X, mid+1, b, z); // Llama a la búsqueda en la mitad superior.
  }
}

// Precondiciones: A un arreglo que contiene una secuencia S con n elementos
// Postcondiciones: A está ordenado en orden ascedente

insertionSortBin(array A){
  int i,j,k,temp,indice,aux;
  int n = A.length; // Obtiene la longitud del arreglo A.
// Recorre los elementos de A, comenzando desde el segundo elemento (i=2).
  for(i = 2; i = n; i++){
  // Encuentra la posición donde debe insertarse el elemento A[i] en A,
  // utilizando una búsqueda binaria para mantener la eficiencia.
    indice = BusquedaBinIn(A,0,i-1,i);
    aux = i; // Almacena el valor actual de i en la variable aux.
  // Realiza el proceso de inserción de A[i] en su posición correcta en el arreglo A.
    for(j = indice; j=i-1; j++){
      temp = A[j]; // Almacena el valor actual de A[j] en la variable temp.
      A[j] = aux; // Coloca el valor de aux en A[j].
      aux = A[j+1]; // Actualiza aux con el valor siguiente de A[j].
      A[j+1] = temp; // Coloca el valor de temp en la posición siguiente de A[j].
    }
  }
  return A; // Devuelve el arreglo A ordenado en orden ascendente.
}
```

5 Problema ϕ: Suponga que tiene n intervalos cerrados sobre la recta real: $[a(i), b(i)]$, con $1 \leq i \leq n$
Encontrar la máxima k tal que existe un punto x que es *cubierto* por los k intervalos.

a) Diseña un algoritmo que solucione el problema ϕ.
No dar código, proporciona la estrategia.

La estrategia será la siguiente;

1) : Ordenamos los intervalos de menor a mayor.

2) : Comenzamos con el primer intervalo y verificamos si x está contenido en él, si lo está, aumentamos k en 1 y continuamos con el siguiente intervalo.

3) : Si x no está contenido en el primer intervalo, continuar con el siguiente intervalo.

4) : Repetimos los pasos **2)** y **3)** hasta que x esté contenido en un intervalo o hasta que se hayan examinado todos los intervalos.

b) Justifica que tu propuesta de algoritmo es correcta.
La propuesta es correcta porque estamos garantizando que encontraremos el mayor k posible, además, si encontramos un punto x que está contenido en k intervalos, entonces no hay posibilidad de encontrar otro punto x que esté contenido en $k+1$ intervalos.

c) Calcula, con detalle, la complejidad computacional de tu propuesta.
Comencemos con el ordenamiento de los intervalos, este se puede realizar utilizando un algoritmo de ordenamiento de comparación, como el de Quick Sort o el de Merge Sort (para los fines de este ejercicio tomaremos a Merge Sort).

Ahora, recordemos que la complejidad de mergesort es $O(n \log n)$.

En el peor de los casos, el algoritmo de ordenamiento puede dividir la lista de intervalos en sublistas de tamaño 1, en este caso, el algoritmo tendrá que realizar $n-1$ comparaciones, lo que nos dará una complejidad de $O(n \log n)$.

d) Proporciona un pseudo-código del algoritmo propuesto.

```
def kIntervalos(intervalos):
  //  Pre-Condiciones: Una lista de n intervalos cerrados, finita, no vacia
  //  y que cumpla que esten sobre la
  //  recta real [a(i), b(i)], con 1 =< i =< n

  // Post-Condiciones: Devuelve la maxima k tal que existe un punto x que
  // es cubierto por los k intervalos

  mergesort(intervalos)
  k <- 1  // El primer intervalo siempre cubre un punto
  navi <- intervalos[0][1] // Hey listen, El extremo der del primer intervalo
  for interval in intervalos: // Iteramos sobre los intervalos
    if interval[0] > navi: // Si el extremo izquierdo del intervalo actual
                           // es mayor al extremo der del intervalo anterior
                           // entonces el punto x no es cubierto por los
                           // intervalos anteriores
        k += 1             // Aumentamos el contador de puntos cubiertos
        navi <- interval[1] // Actualizamos el extremo derecho del intervalo anterior
                            // al extremo derecho del intervalo actual
  return k
```

6 **Opcional.**
Problema Γ: Dados n puntos en el plano, encontrar un polígono que tenga como vértices los n puntos dados.
(a) Diseña un algoritmo que solucione el problema Γ.
No des código, proporciona la estrategia.
(b) Justifica que tu propuesta de algoritmo es correcta.
(c) Calcule, con detalle, la complejidad computacional de tu propuesta.
(d) Proporciona un pseudo-código del algoritmo propuesto.

Entrada: Un arreglo A de tuplas ordenadas (de tamaño n) de la forma (P_i, x, y, α) que modelan un punto, donde x representa la coordenada en el eje x del punto, y representa la coordenada en el eje y del punto, P_i representa un identificador del punto y α representa un ángulo (inicialmente todos los puntos tendrán ángulo 0).

Salida: Un arreglo B de segmentos (de tamaño n) representados por un par de la forma (P_i, P_j) con $P_i \neq P_j$, donde P_i, P_j son los identificadores de los puntos.

a) **Algoritmo:**

(a) Usamos el algoritmo **Búsqueda Lineal** para encontrar el elemento menor respecto a la coordenada y de los puntos, Sea $(P_m, x_m, y_m, 0)$ elemento menor respecto a la coordenada y en A.

(b) Para cada punto $(P_i, x_i, y_i, \alpha_i)$ distinto de $(P_m, x_m, y_m, 0)$, calculamos el ángulo entre la recta $f(x) = y_m$ y el segmento (P_i, P_m), esto lo haremos de la siguiente manera: si $\frac{x_m - x_i}{y_m - y_i} > 0$ entonces $\alpha_i = \arctan(\frac{x_m - x_i}{y_m - y_i})$, en otro caso $\alpha_i = 180 - \arctan(|\frac{x_m - x_i}{y_m - y_i}|)$.

(c) Aplicamos el algoritmo de ordenamiento **Merge Sort** tomando en cuenta los ángulos de los puntos.

(d) Creamos un nuevo arreglo de segmentos B vacío de tamaño n, y para cada i con $0 \leq i \leq n-1$, creamos el segmento (P_i, P_{i+1}) donde P_i, P_{i+1} son los identificadores de los puntos que están en la posición i e $i+1$ del arreglo A y añadimos el segmento a B.

(e) Finalmente añadimos el segmento (P_{n-1}, P_0) al final del arreglo B, donde P_{n-1}, P_0, son los identificadores del último punto del arreglo A y del primer punto del arreglo A respectivamente.

(f) Devolvemos B.

b) Recordemos que un polígono es una figura cerrada de n lados donde ninguno de sus lados se intercecta entre sí, para justificar que nuestro algoritmo funciona, justificaremos que nuestro algoritmo en efecto crea un polígono, ya que por la construcción del algoritmo, este siempre devuelve una figura donde sus vértices son los puntos en el plano y los lados son los segmentos del arreglo de salida.

(a) La figura es cerrada: esto se tiene porque podemos hacer un recorrido de los vértices del polígono y llegar al punto de inicio, dicho recorrido se vería de la siguiente forma $(P_0, P_1, P_2, ...Pn-1, P_0)$, donde tomamos a los puntos considerando el arreglo de entrada ordenado.

(b) Por demostrar que para todos par de segmentos s_1, s_2, s_1 no se intercecta con s_2, procederemos por contradicción, supongamos que existen dos segmentos $s_1 = (B, A)$, $s_2 = (C, D)$, tal que s_1 se intersecta con s_2.

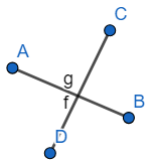

Figura 1. Segmentos que se intersectan

Pero notemos que si esto pasa, implicaría que debería existir el segmento que va del punto B al C, ya que el ángulo de C es menor que el de A, lo cuál es una contradicción, ya que el ángulo de A es menor que el de C, porque el arreglo se ordeno respecto a los ángulos, y con este orden creamos los segmentos.

Por lo tanto el algoritmo crea un polígono, ya que crea una figura cerrada donde sus lados no se intersectan.

c) Calculemos la complejidad del algoritmo, para esto especificaremos la complejidad de cada paso.

(a) Buscar el elemento más pequeño de un arreglo es $O(n)$.

(b) Calcular el ángulo para un punto es contante, por lo que hacer esto para los n puntos tendría complejidad $O(n)$.

(c) Ordenar el arreglo toma tiempo $O(n\ log(n))$.

(d) Crear un segmento es contante, por lo que hacer los n segmentos tomaría tiempo $O(n)$.

(e) Crear el último segmento tomaría tiempo $O(1)$.

(f) Devolver B es $O(1)$.

Notemos que al final la complejidad es $O(3n + n\ log(n))$, lo cual es $O(n\ log(n))$.

d) Pseudocódigo

```
//Precodiciones: Un arreglo A de tuplas ordenadas (de tamaño n)
//de la forma (Pi, x, y, α) que modelan un punto, donde x
//representa la coordenada en el eje x del punto, y representa
//la coordenada en el eje y del punto, Pi representa un
//identificador del punto y α representa un ángulo
//(inicialmente todos los puntos tendrán águlo 0).

//Postcondiciones: Un arreglo B de segmentos (de tamaño n) representados
//por un par de la forma (Pi, Pj ) con Pi ≠ Pj , donde
//Pi, Pj son los identificadores de los puntos.

creaPolinomio(array A){ //Función principal que crea el polinomio a partir del arreglo A.
  PuntoMenor = (Pi, xi, yi, ai) = BusquedaMenor(A); //Encuentra el punto con las coordenadas más
                                                    //pequeñas en A y almacena sus valores en PuntoMenor.
  for( (Pi, xi, yi, ai) in A){ //Itera sobre cada tupla en el arreglo A.
    if((xm-xi)/(ym-yi) > 0){ //Comprueba si la razón de las coordenadas es mayor que 0.
      ai = arctan( (xm-xi) / (ym-yi)); //Calcula el ángulo ai si la condición es verdadera.
    }else{
      ai = 180 - arctan(|(xm-xi)/(ym-yi)|);//Calcula el ángulo ai de otra manera si la condición es falsa.
    }
  }
  mergeSortAng(A); //Ordena el arreglo A basado en los valores de ángulo ai.
  array B; //Crea un nuevo arreglo B para almacenar los segmentos.
  for((Pi, xi, yi, ai) in A){//Itera nuevamente sobre el arreglo A después de la ordenación.
    B.add((Pi,Pi+1));//Agrega un par de puntos (Pi, Pi+1) al arreglo B, conectando los puntos ordenados.
  }
  B.add((Pn,P0));
  return B; // Devuelve el arreglo B que contiene los segmentos generados.
}
```