Análisis de Algoritmos y Estructuras para Datos Masivos 2023

Unidad 6 - Tarea

David Aarón Ramírez Olmeda

Introducción:

En este ejercicio se planteó el problema de la intersección de listas de posteo, es decir, encontrar los documentos que contienen todas las palabras de una consulta. Para esto, se implementaron los algoritmos de unión y búsqueda dados en la lectura 3.1 Melding Algorithms y 3.2 Search algorithm de [@BLOLS2010]. Se realizó un experimento para medir el tiempo y el número de comparaciones necesarios para realizar la intersección de pares, tercias y cuartetas de listas de posteo seleccionadas aleatoriamente.

Desarrollo:

In [1]:
```
import itertools
import time
import random
import numpy as np
import matplotlib.pyplot as plt
import json
```

In [2]:
```
with open('listas-posteo-100.json') as f:
    data = f.readlines()

postings_lists = {}
for line in data:
    term, postings = json.loads(line)
    postings_lists[term] = postings
```

Definimos los algoritmos

In [3]:
```
def melding_algorithm(lists):
    merged = []
    for lst in lists:
        merged.extend(lst)
    return sorted(merged)
```

In [4]:
```
def binary_search(lst, x):
    low = 0
    high = len(lst) - 1
    while low <= high:
        mid = (low + high) // 2
        if lst[mid] == x:
            return True
        elif lst[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return False
```

In [5]:
```
def galloping_search(A, key):
    pos = 0
    jump = 1
    while pos < len(A) and A[pos] < key:
        pos += jump
        jump *= 2
    left = pos // 2
    right = min(pos, len(A)-1)
    while left <= right:
        mid = (left + right) // 2
        if A[mid] == key:
            return mid
        elif A[mid] < key:
            left = mid + 1
        else:
            right = mid - 1
    return None
```

In [6]:
```
#postings_lists = {'_url': [1, 2, 3, 4, 6], '_date': [2, 4, 5, 6, 8], '_l
# Prueba pequeña
```

In [7]:
```
pairs = list(itertools.combinations(postings_lists.values(), 2))
random.shuffle(pairs)
A = pairs[:1000]

triplets = list(itertools.combinations(postings_lists.values(), 3))
random.shuffle(triplets)
B = triplets[:1000]

quadruplets = list(itertools.combinations(postings_lists.values(), 4))
random.shuffle(quadruplets)
C = quadruplets[:1000]
```

Se generaron aleatoriamente conjuntos de pares, tercias y cuartetas de listas de posteo

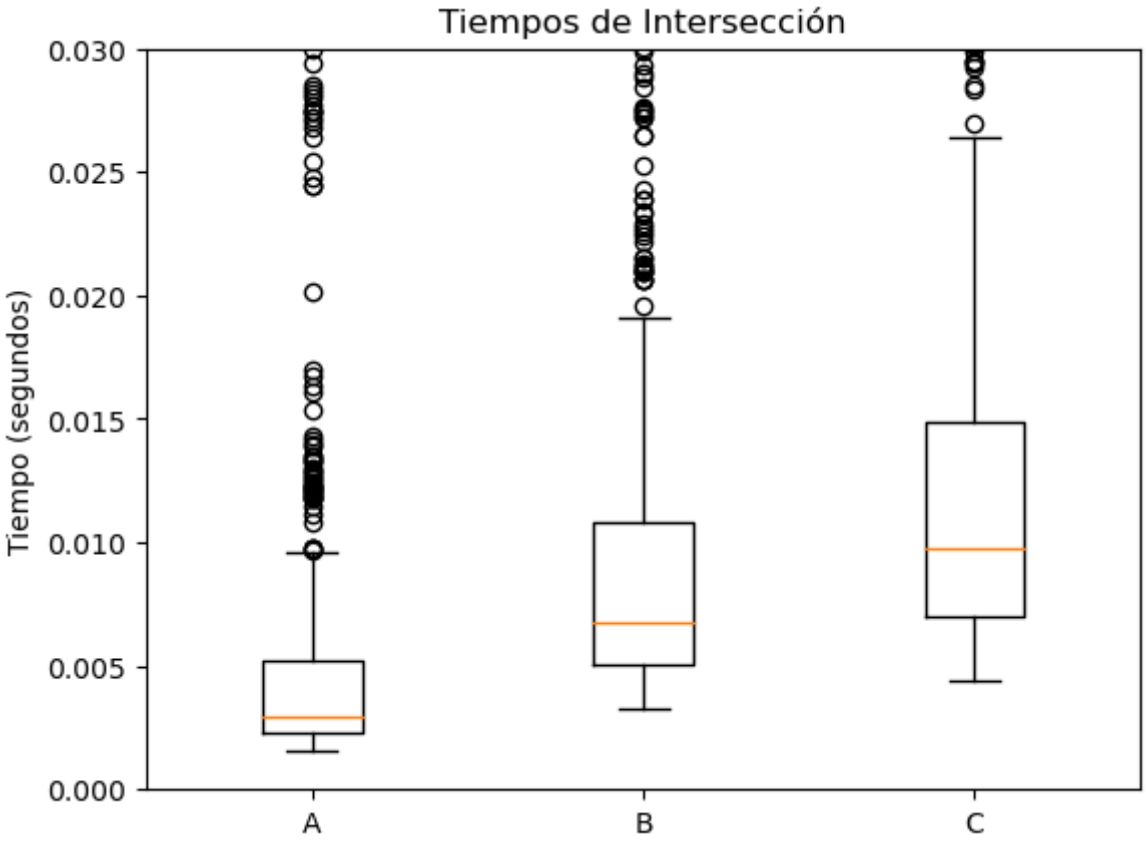
In [8]:
```
results = {'A': [], 'B': [], 'C': []}
for i, lsts in enumerate(A):
    start_time = time.time()
    intersection = melding_algorithm(lsts)
    comparisons = len(lsts) - 1  # número de comparaciones en el algoritm
    for j in range(len(lsts) - 1):
        comparisons += len(intersection) - 1  # número de comparaciones e
        intersection = [x for x in intersection if binary_search(lsts[j+1
    end_time = time.time()
    elapsed_time = end_time - start_time
    results['A'].append({'time': elapsed_time, 'comparisons': comparisons

for i, lsts in enumerate(B):
    start_time = time.time()
    intersection = melding_algorithm(lsts)
    comparisons = len(lsts) - 1
    for j in range(len(lsts) - 1):
        comparisons += len(intersection) - 1
        intersection = [x for x in intersection if binary_search(lsts[j+1
    end_time = time.time()
    elapsed_time = end_time - start_time
    results['B'].append({'time': elapsed_time, 'comparisons': comparisons

for i, lsts in enumerate(C):
    start_time = time.time()
    intersection = melding_algorithm(lsts)
    comparisons = len(lsts) - 1
    for j in range(len(lsts) - 1):
        comparisons += len(intersection) - 1
        intersection = [x for x in intersection if binary_search(lsts[j+1
    end_time = time.time()
    elapsed_time = end_time - start_time
    results['C'].append({'time': elapsed_time, 'comparisons': comparisons
```

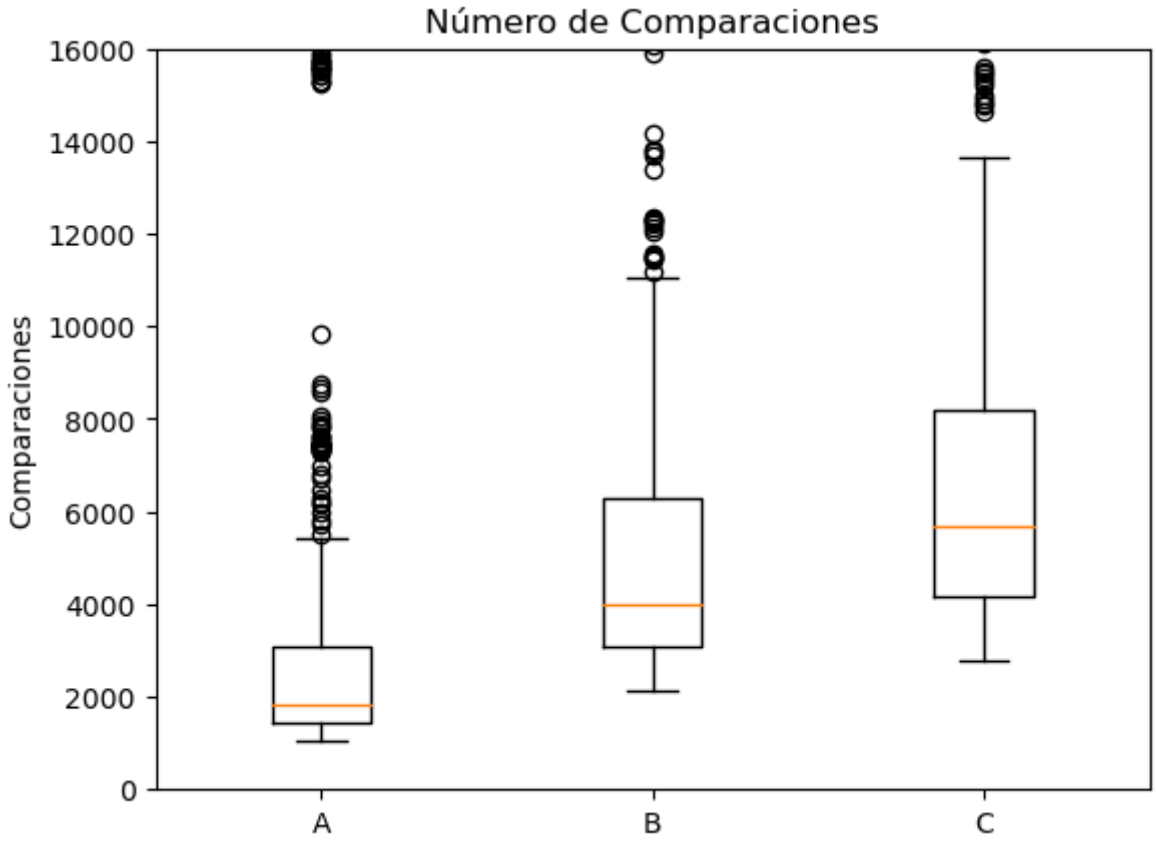
Para cada conjunto, se realizó la intersección de las listas usando los algoritmos de unión y búsqueda definidos anteriormente. Se midió el tiempo y el número de comparaciones necesarios para realizar cada intersección.

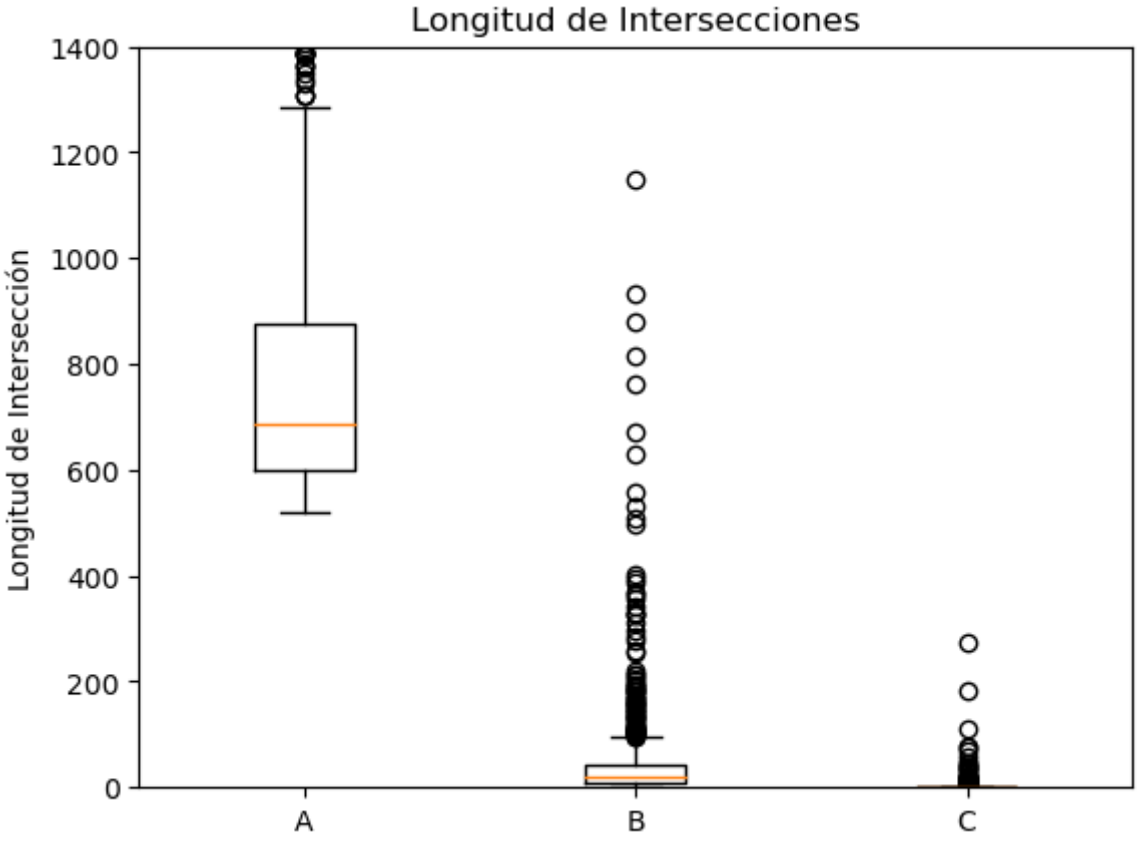
In [9]:

```
tiempos = []
comparaciones = []
intersecciones = []

for experimento in ['A', 'B', 'C']:
    tiempos.append([result['time'] for result in results[experimento]])
    comparaciones.append([result['comparisons'] for result in results[exp
    intersecciones.append([result['length'] for result in results[experim

fig1, ax1 = plt.subplots()
ax1.boxplot(tiempos)
ax1.set_title('Tiempos de Intersección')
ax1.set_xticklabels(['A', 'B', 'C'])
ax1.set_ylabel('Tiempo (segundos)')
ax1.set_ylim([0, 0.03])

fig2, ax2 = plt.subplots()
ax2.boxplot(comparaciones)
ax2.set_title('Número de Comparaciones')
ax2.set_xticklabels(['A', 'B', 'C'])
ax2.set_ylabel('Comparaciones')
ax2.set_ylim([0, 16000])

fig3, ax3 = plt.subplots()
ax3.boxplot(intersecciones)
ax3.set_title('Longitud de Intersecciones')
ax3.set_xticklabels(['A', 'B', 'C'])
ax3.set_ylabel('Longitud de Intersección')
ax3.set_ylim([0, 1400])

print('Para el algoritmo binary_search se tiene:')
plt.show()
```

```
Para el algoritmo binary_search se tiene:
```


Tiempos de Intersección
Tiempo (segundos)
0.030
0.025
0.020
0.015
0.010
0.005
0.000
A
B
C


Número de Comparaciones
Comparaciones
16000
14000
12000
10000
8000
6000
4000
2000
0
A
B
C

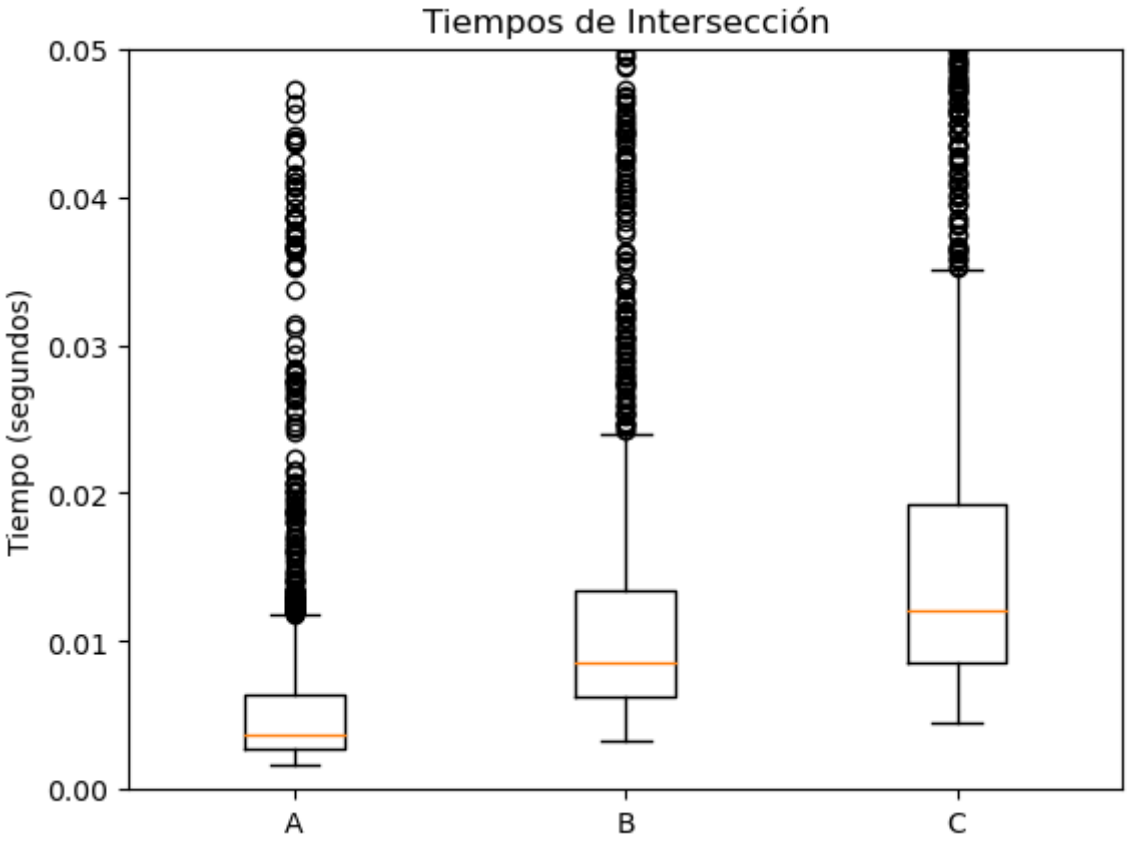

Longitud de Intersecciones
Longitud de Intersección
1400
1200
1000
800
600
400
200
0
A
B
C

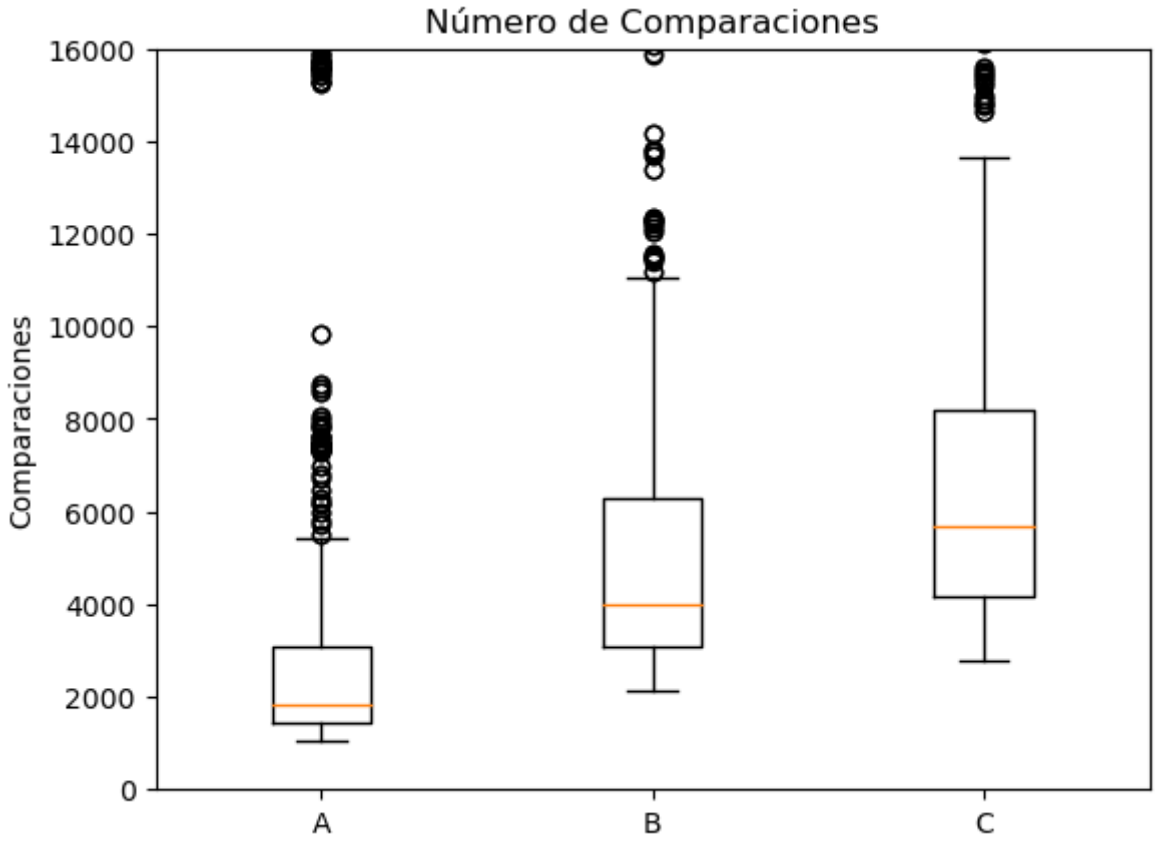
In [10]:

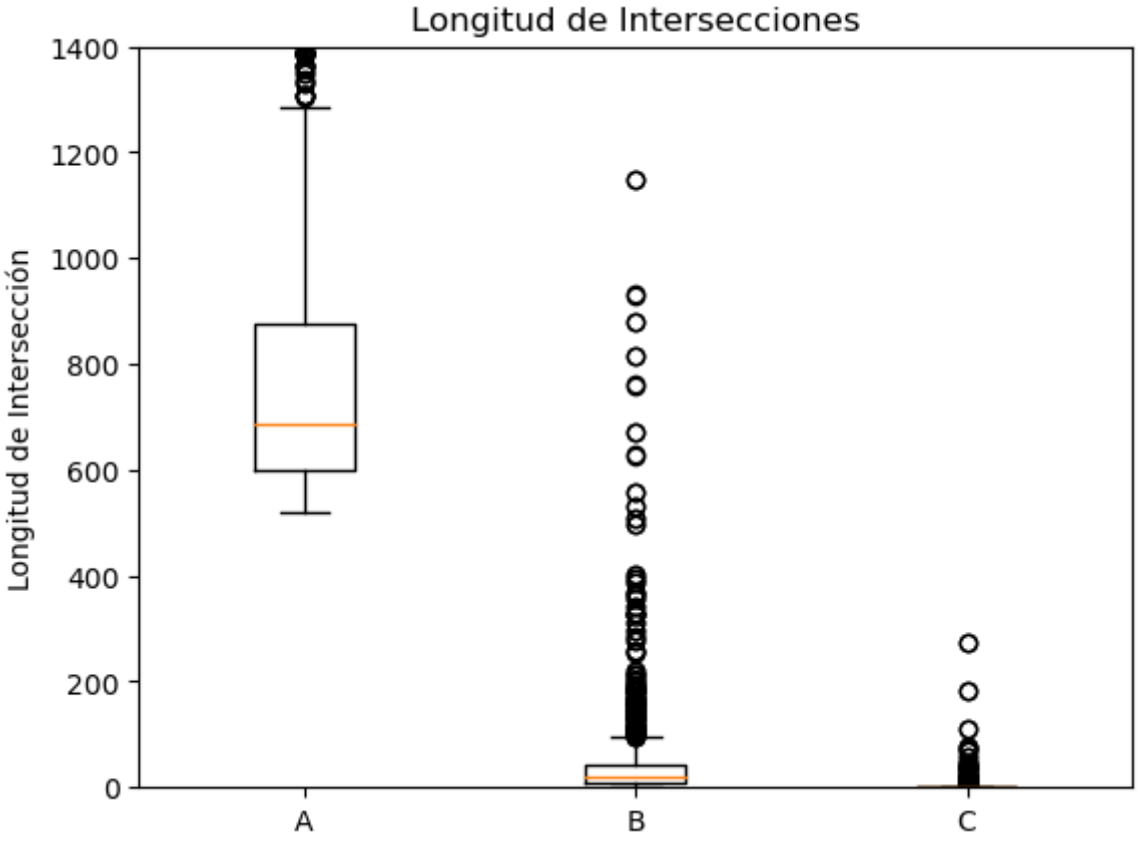
```
for i, lsts in enumerate(A):
    start_time = time.time()
    intersection = melding_algorithm(lsts)
    comparisons = len(lsts) - 1
    for j in range(len(lsts) - 1):
        comparisons += len(intersection) - 1
        intersection = [x for x in intersection if galloping_search(lsts[
    end_time = time.time()
    elapsed_time = end_time - start_time
    results['A'].append({'time': elapsed_time, 'comparisons': comparisons

for i, lsts in enumerate(B):
    start_time = time.time()
    intersection = melding_algorithm(lsts)
    comparisons = len(lsts) - 1
    for j in range(len(lsts) - 1):
        comparisons += len(intersection) - 1
        intersection = [x for x in intersection if galloping_search(lsts[
    end_time = time.time()
    elapsed_time = end_time - start_time
    results['B'].append({'time': elapsed_time, 'comparisons': comparisons

for i, lsts in enumerate(C):
    start_time = time.time()
    intersection = melding_algorithm(lsts)
    comparisons = len(lsts) - 1
    for j in range(len(lsts) - 1):
        comparisons += len(intersection) - 1
        intersection = [x for x in intersection if galloping_search(lsts[
    end_time = time.time()
    elapsed_time = end_time - start_time
    results['C'].append({'time': elapsed_time, 'comparisons': comparisons
```

In [11]:
```
tiempos = []
comparaciones = []
intersecciones = []

for experimento in ['A', 'B', 'C']:
    tiempos.append([result['time'] for result in results[experimento]])
    comparaciones.append([result['comparisons'] for result in results[exp
    intersecciones.append([result['length'] for result in results[experim

fig1, ax1 = plt.subplots()
ax1.boxplot(tiempos)
ax1.set_title('Tiempos de Intersección')
ax1.set_xticklabels(['A', 'B', 'C'])
ax1.set_ylabel('Tiempo (segundos)')
ax1.set_ylim([0, 0.05])

fig2, ax2 = plt.subplots()
ax2.boxplot(comparaciones)
ax2.set_title('Número de Comparaciones')
ax2.set_xticklabels(['A', 'B', 'C'])
ax2.set_ylabel('Comparaciones')
ax2.set_ylim([0, 16000])

fig3, ax3 = plt.subplots()
ax3.boxplot(intersecciones)
ax3.set_title('Longitud de Intersecciones')
ax3.set_xticklabels(['A', 'B', 'C'])
ax3.set_ylabel('Longitud de Intersección')
ax3.set_ylim([0, 1400])

print('Para el algoritmo galloping_search se tiene:')
plt.show()
```

```
Para el algoritmo galloping_search se tiene:
```


Tiempos de Intersección
Tiempo (segundos)
0.05
0.04
0.03
0.02
0.01
0.00
A
B
C


Número de Comparaciones
Comparaciones
16000
14000
12000
10000
8000
6000
4000
2000
0
A
B
C



Conclusión

Se observó que los algoritmos implementados lograron encontrar la intersección de las listas de posteo en tiempos razonables. Además, se encontró que el número de comparaciones necesarias aumenta con el número de listas de posteo que se intersectan.

En las gráficas se observó que el tiempo de ejecución y el número de comparaciones variaron ampliamente entre los diferentes conjuntos de listas de posteo. Estos resultados sugieren que el rendimiento de los algoritmos puede ser muy dependiente de las características de las listas de posteo que se están intersecando.