Suffix structures analysis

Author: Krzysztof Chmielewski

This section is dedicated to suffix structures analysis and comparison. First imports...

In [1]:
```python
from suffix_structures import compare_suffix_structures
import pandas as pd
import matplotlib.pyplot as plt
```

Suffix structures comparison by text length

In [2]:
```python
short_text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore "
len(short_text)
```

Out[2]: 100

In [3]:
```python
medium_text = "k8PHjpK6eF5b5KizA67XvUfZ6y9Reat6WrKS9p87v0xrLuApgpRAqGmcXgftBUEpPX7Cc04ncNc9fU2SiGRWzwWfj2mbEw5Dp8PQtfhcxUxiTLkSirNY7HCynkuLStLXD5nd84qKygWm6uLTFjZB5BAyZFWJbkwfquv9K9V5mgQLMAarqJ4eDBGiFXMnVL3D9c17Dh9
len(medium_text)
```

Out[3]: 1000

In [4]:
```python
long_text = "tBkLiLRWdPeAKLiSTuYPPv2XLdc7exw4Z1xNrNELpEk26Uy3abA7KZicSDnygfuVUQkm8QV3xfNW1db5UiyhFCGV2fLtC9yA0HxJKRwrCYMiSjrwbNGiNxt3i1SMe9N05HjcapE66QFXQBcAhYVtm693jEw7d7geWkxgJc42puaPxGpHDcdhL2Xe0HAH85NucRPggaY0P
len(long_text)
```

Out[4]: 10000

In [5]:
```python
very_long_text = "wT5xGeHAVZ81rECuL5P0wg5WMUMvBYgumqYhgyBKr9RHQmGBd0nS01Uv0fGgb9giM0b3uSGJJdFxJJUuPh2JRuN5Xu4eR8474Kxkh8EHagdi7JiJWvD9bp73a6wZ194CeaVkqQAzZ7JhFRefnwD7SRURCDPRnTMgS4MYgJZgKtFBDVJgrhG2wCAfunrNUQUuE3T8
len(very_long_text)
```

Out[5]: 100000

Creating Pandas DataFrames for easier access

In [6]:
```python
d1 = compare_suffix_structures(short_text)
d2 = compare_suffix_structures(medium_text)
d3 = compare_suffix_structures(long_text)
d4 = compare_suffix_structures(very_long_text)

texts = [short_text, medium_text, long_text, very_long_text]
dicts = [d1,d2,d3,d4]

sa_compare = dict()
st_compare = dict()
for text, di in zip(texts, dicts):
    length = len(text)
    sa_compare[f"{length}"] = di["suffix_array"]
    st_compare[f"{length}"] = di["suffix_tree"]

sa_df = pd.DataFrame(sa_compare)
st_df = pd.DataFrame(st_compare)
```

In [7]:
```python
sa_df
```

Out[7]:

	100	1000	10000	100000
construction_time_ms	0.108957	1.379728	53.429842	3.120844e+03
memory_usage_kb	0.000000	256.000000	48896.000000	4.851328e+06
size	4800.000000	48000.000000	480000.000000	4.800000e+06

In [8]:
```python
st_df
```

Out[8]:

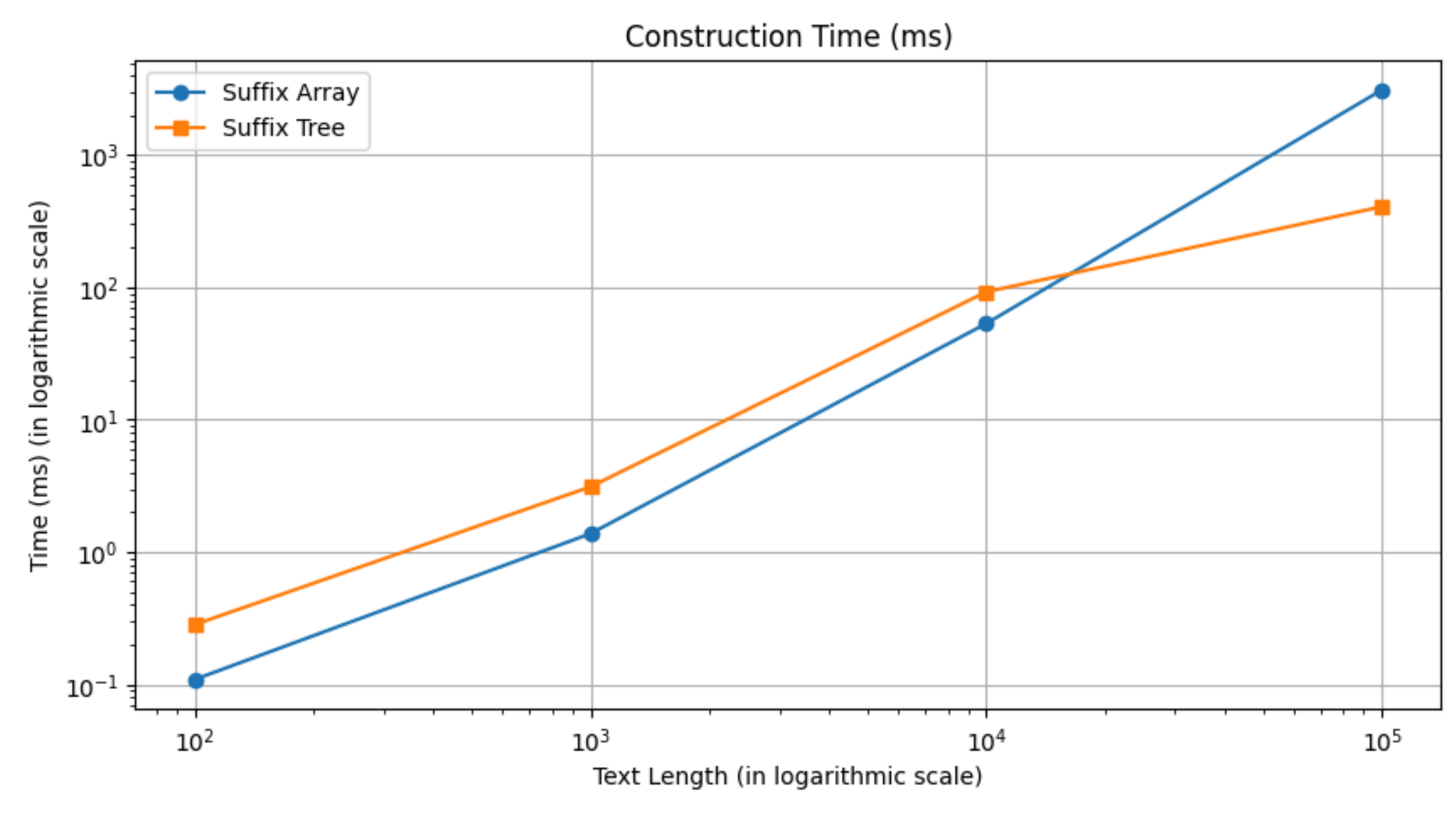
	100	1000	10000	100000
construction_time_ms	0.283241	3.118515	92.747211	4.090490e+02
memory_usage_kb	0.000000	128.000000	2304.000000	2.150400e+04
size	6864.000000	57216.000000	621216.000000	5.879136e+06

Plotting results

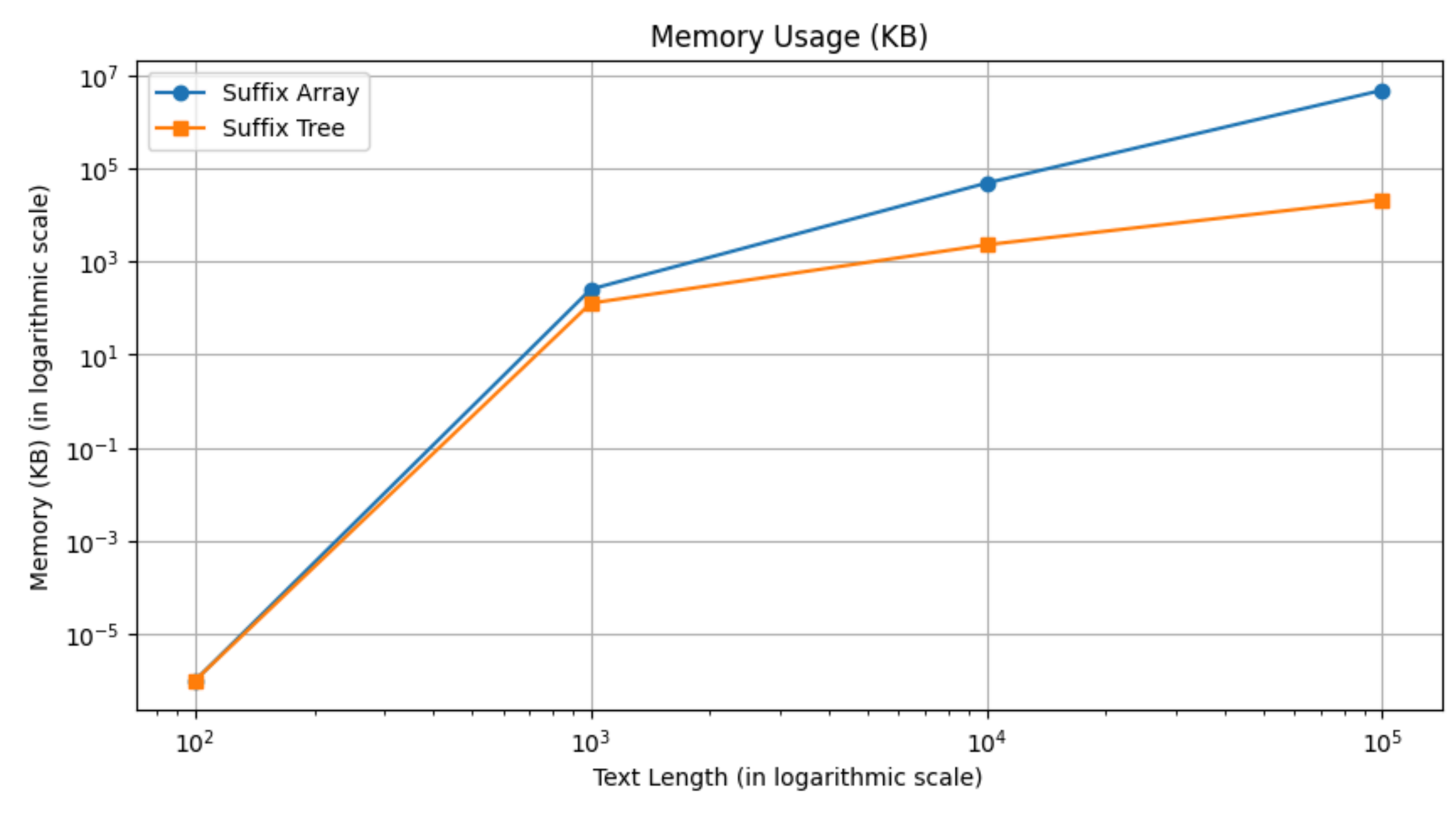
In [9]:
```python
sa_df.columns = sa_df.columns.astype(int)
st_df.columns = st_df.columns.astype(int)

text_sizes = sa_df.columns.tolist()
sa_label, st_label = 'Suffix Array', 'Suffix Tree'
```

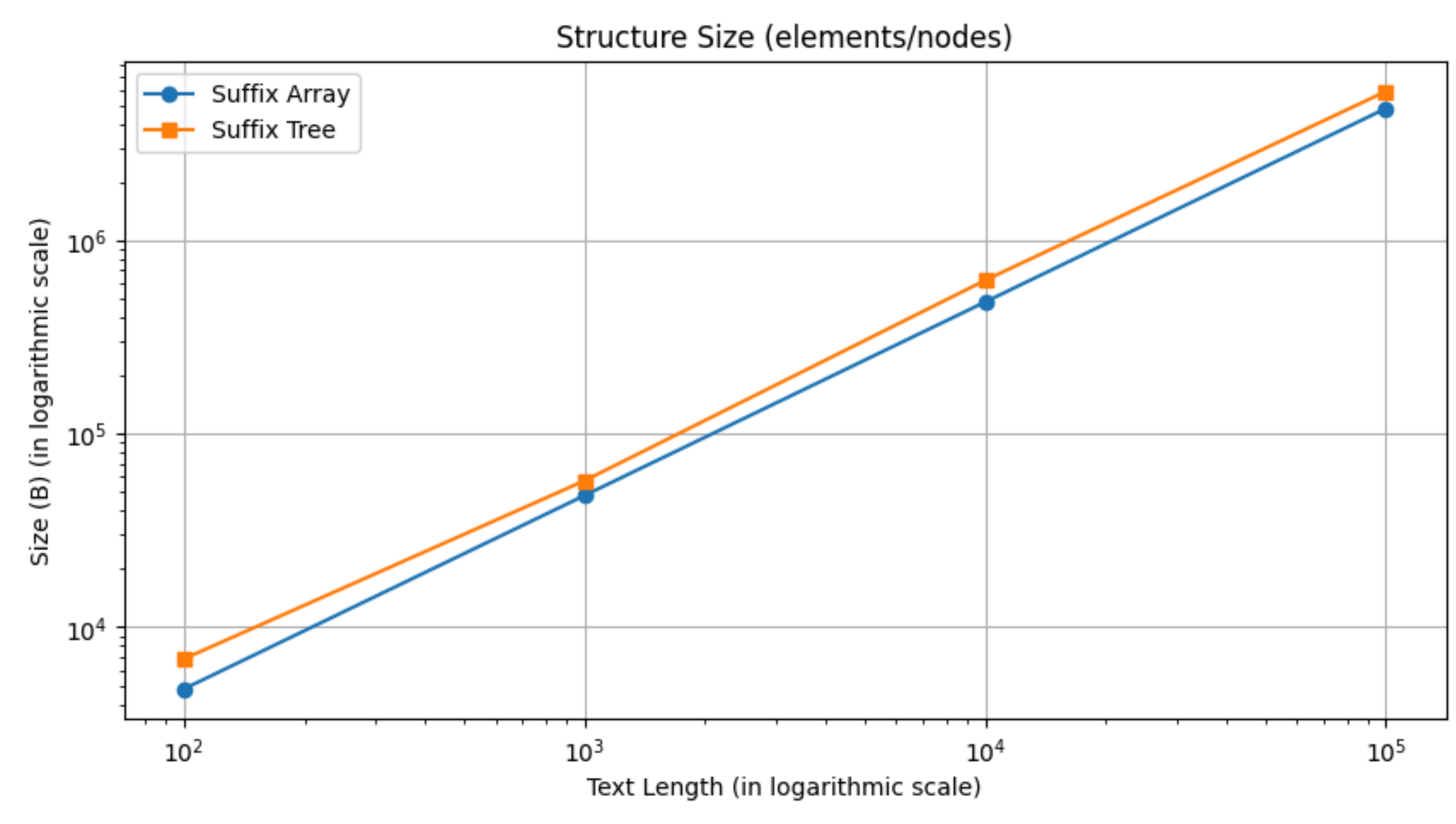
In [10]:
```python
# Construction Time
plt.figure(figsize=(10, 5))
plt.plot(text_sizes, sa_df.loc["construction_time_ms"], marker='o', label=sa_label)
plt.plot(text_sizes, st_df.loc["construction_time_ms"], marker='s', label=st_label)
plt.title("Construction Time (ms)")
plt.xlabel("Text Length (in logarithmic scale)")
plt.ylabel("Time (ms) (in logarithmic scale)")
plt.xscale('log')
plt.yscale('log')
plt.legend()
plt.grid(True)
plt.show()
```



In [11]:
```python
# Memory Usage
plt.figure(figsize=(10, 5))
plt.plot(text_sizes, sa_df.loc["memory_usage_kb"].replace(0, 1e-6), marker='o', label=sa_label)
plt.plot(text_sizes, st_df.loc["memory_usage_kb"].replace(0, 1e-6), marker='s', label=st_label)
plt.title("Memory Usage (KB)")
plt.xlabel("Text Length (in logarithmic scale)")
plt.ylabel("Memory (KB) (in logarithmic scale)")
plt.xscale('log')
plt.yscale('log')
plt.legend()
plt.grid(True)
plt.show()
```



In [12]:
```python
# Structure Size
plt.figure(figsize=(10, 5))
plt.plot(text_sizes, sa_df.loc["size"], marker='o', label=sa_label)
plt.plot(text_sizes, st_df.loc["size"], marker='s', label=st_label)
plt.title("Structure Size (elements/nodes)")
plt.xlabel("Text Length (in logarithmic scale)")
plt.ylabel("Size (B) (in logarithmic scale)")
plt.xscale('log')
plt.yscale('log')
plt.legend()
plt.grid(True)
plt.show()
```



Plotting results with more samples

In [13]:
```python
import random
import numpy as np
import timeit

def randtext(size: int) -> str:
    return "".join( (chr(ord('a') + random.randint(0,25))) for _ in range(size) )

def plt_approximate(ax, xs, ys, deg, label = None, title = None):
    poly = np.poly1d(np.polyfit(xs, ys, deg))
    domain = np.linspace(min(xs), max(xs), 100)

    ax.plot(domain, poly(domain), label = label)
    ax.scatter(xs, ys)
    if label: ax.legend()
    if title: ax.set_title(title)

def time_exec(fun, *data):
    return timeit.timeit(lambda: fun(*data), number=1)
```

In [14]:
```python
sizes = np.linspace(10, 5e4, 20, dtype=int)

sa_compare = dict()
st_compare = dict()

for size in sizes:
    text = randtext(size)
    di = compare_suffix_structures(text)
    sa_compare[f"{size}"] = di["suffix_array"]
    st_compare[f"{size}"] = di["suffix_tree"]

sa_df = pd.DataFrame(sa_compare)
st_df = pd.DataFrame(st_compare)
sa_df.columns = sa_df.columns.astype(int)
st_df.columns = st_df.columns.astype(int)

text_sizes = sa_df.columns.tolist()
sa_label, st_label = 'Suffix Array', 'Suffix Tree'
```

In [15]:
```python
sa_df
```

Out[15]:

	10	2641	5272	7903	10534	13165	15796	18427	21058	23689	26320	28951	31582	34213	36844	
construction_time_ms	0.021219	3.736019	11.663675	16.232729	43.435812	58.992386	65.556526	169.90757	8.585525e+01	8.626056e+01	1.067567e+02	1.280739e+02	1.485925e+02	2.681975e+02	1.951265e+02	3.33633
memory_usage_kb	0.000000	128.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
size	480.000000	126768.000000	253056.000000	379344.000000	505632.000000	631920.000000	758208.000000	884496.00000	1.010784e+06	1.137072e+06	1.263360e+06	1.389648e+06	1.515936e+06	1.642224e+06	1.768512e+06	1.89480

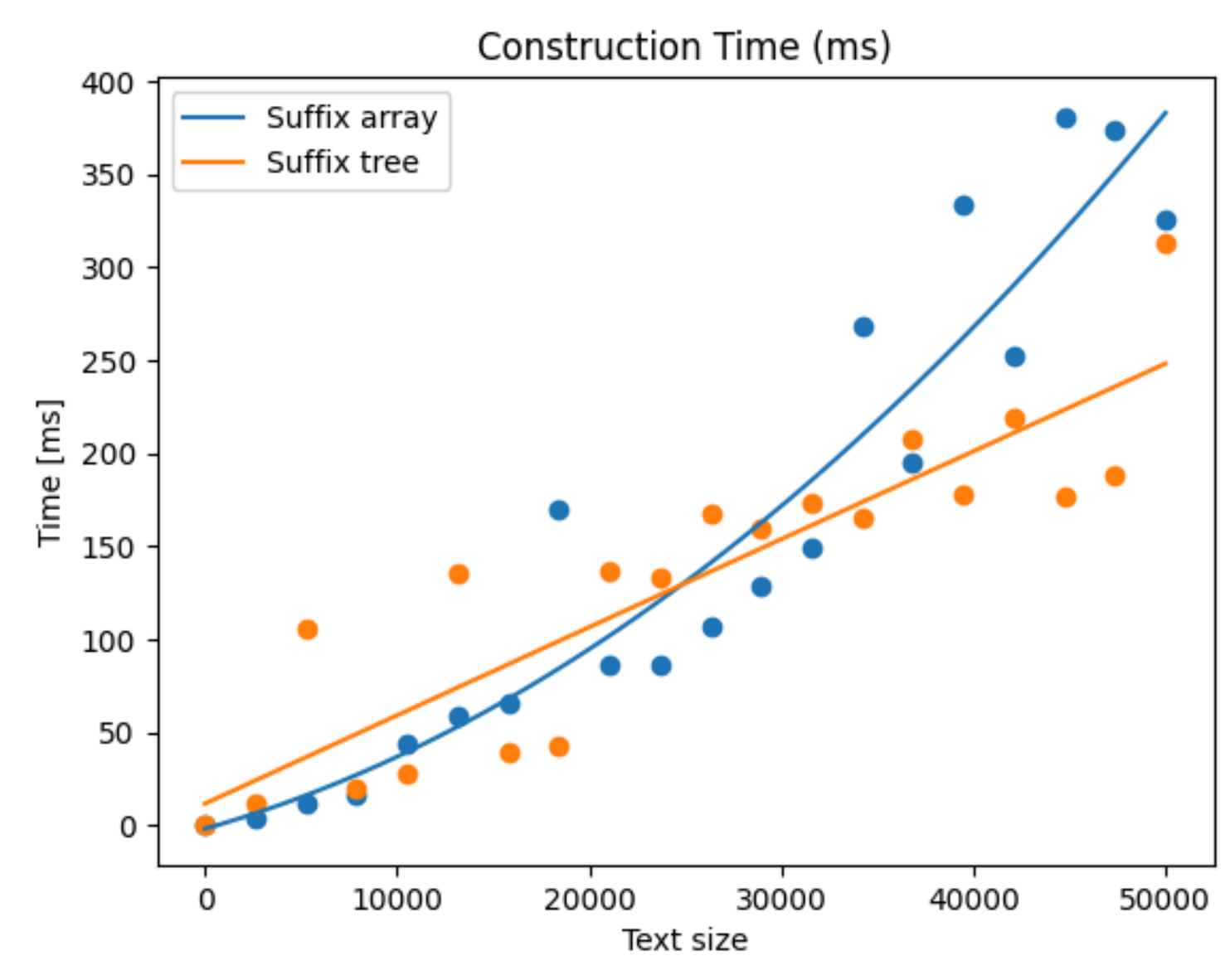
In [16]:
```python
st_df
```

Out[16]:

	10	2641	5272	7903	10534	13165	15796	18427	21058	23689	26320	28951	31582	34213	36844	
construction_time_ms	0.033379	11.235237	105.648756	19.277334	27.210474	135.437012	39.147377	4.316616e+01	1.364088e+02	1.328652e+02	1.679342e+02	1.592736e+02	1.732605e+02	1.651087e+02	2.079289e+02	1.78103
memory_usage_kb	0.000000	0.000000	-3068.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
size	672.000000	165024.000000	317952.000000	476256.000000	645120.000000	817680.000000	991296.000000	1.167408e+06	1.344192e+06	1.521792e+06	1.698288e+06	1.871472e+06	2.045664e+06	2.214768e+06	2.384064e+06	2.55763

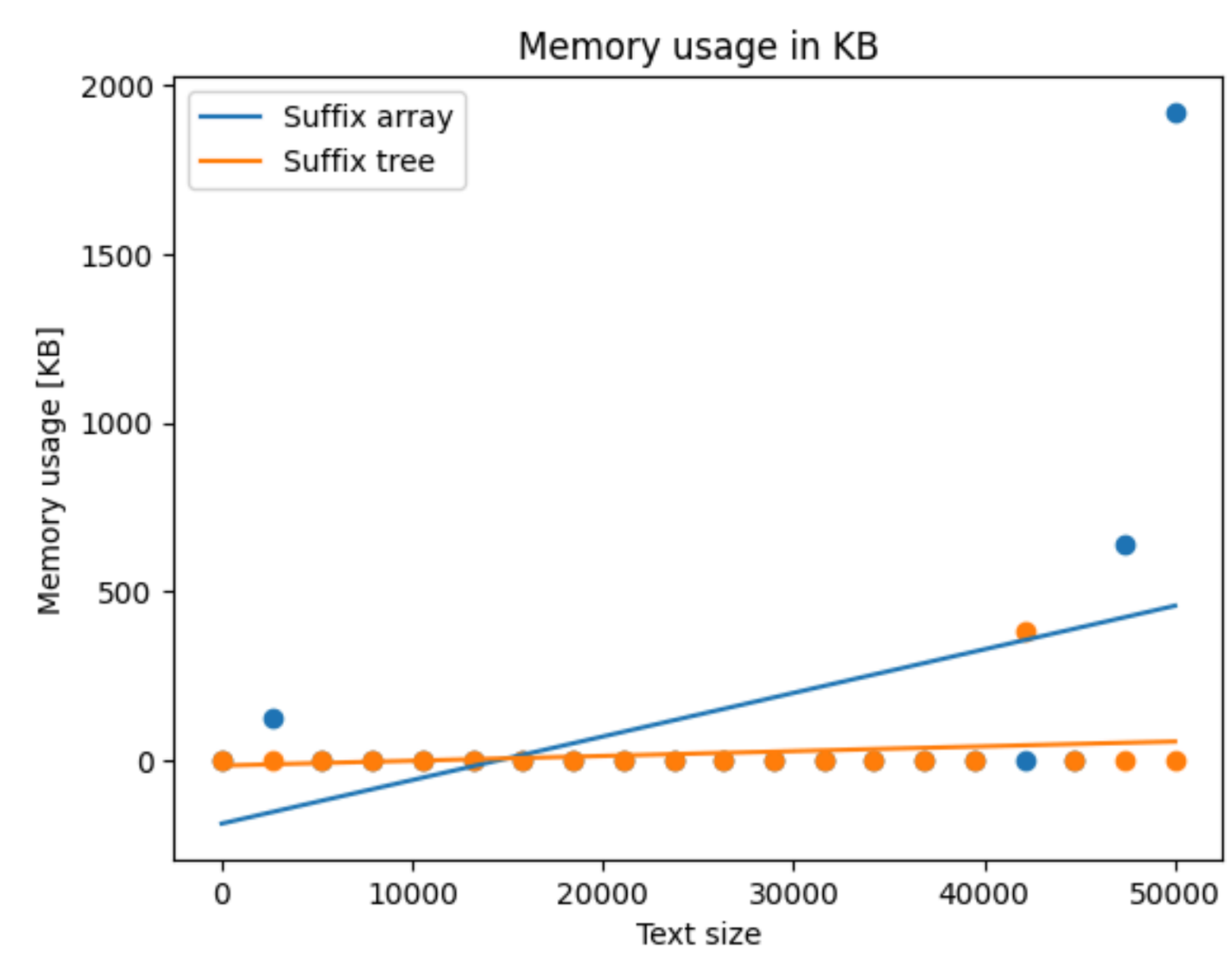
In [17]:
```python
ax = plt.gca()
ax.set_title("Construction Time (ms)")
ax.set_xlabel("Text size")
ax.set_ylabel("Time [ms]")

plt_approximate(ax, sizes, sa_df.loc['construction_time_ms'], deg=2, label="Suffix array")
plt_approximate(ax, sizes, st_df.loc['construction_time_ms'], deg=2, label="Suffix tree")
plt.show()
```



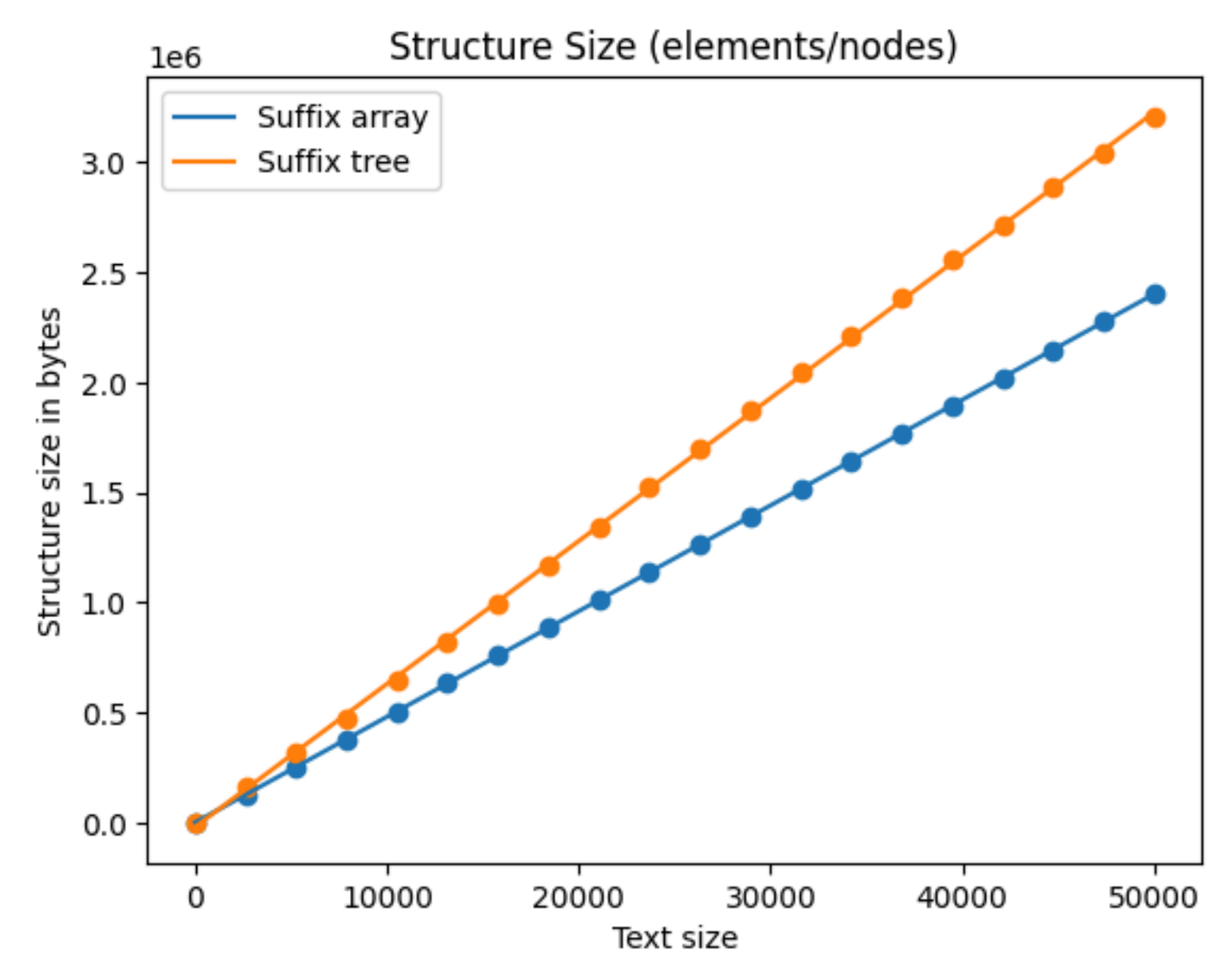
In [18]:
```python
ax = plt.gca()
ax.set_title("Memory usage in KB")
ax.set_xlabel("Text size")
ax.set_ylabel("Memory usage [KB]")

plt_approximate(ax, sizes, sa_df.loc['memory_usage_kb'].clip(lower=0), deg=1, label="Suffix array")
plt_approximate(ax, sizes, st_df.loc['memory_usage_kb'].clip(lower=0), deg=1, label="Suffix tree")
plt.show()
```



In [19]:
```python
ax = plt.gca()
ax.set_title("Structure Size (elements/nodes)")
ax.set_xlabel("Text size")
ax.set_ylabel("Structure size in bytes")

plt_approximate(ax, sizes, sa_df.loc['size'].clip(lower=0), deg=1, label="Suffix array")
plt_approximate(ax, sizes, st_df.loc['size'].clip(lower=0), deg=1, label="Suffix tree")
plt.show()
```



Conclusions

We can see that both structures are linear in terms of structure sizes in respect to text size, however suffix tree takes more memory as expected. In terms of time construction, for larger texts suffix trees are a better choice that suffix arrays given that time complexity for constructing suffix tree is linear using Ukkonen's algorithm giving $O(n)$ and used simple method of constructing suffix array is giving us $O(n^2 logn)$ time complexity. However suffix array can be a better choice for smaller texts.