Merge Insertion

Hybrid Sort

Nathan, Shun Kah, Huai Zhi

A

Implementation

Implementation of hybrid sort

B

Generation of input data

Generation of random data sets

C

Analysis of time complexity

Analysing of time complexity with different inputs

D

Comparison with original mergesort

Comparing hybrid sort with original merge sort

Part A

Algorithm Implementation

Pseudo-code for Merge Sort

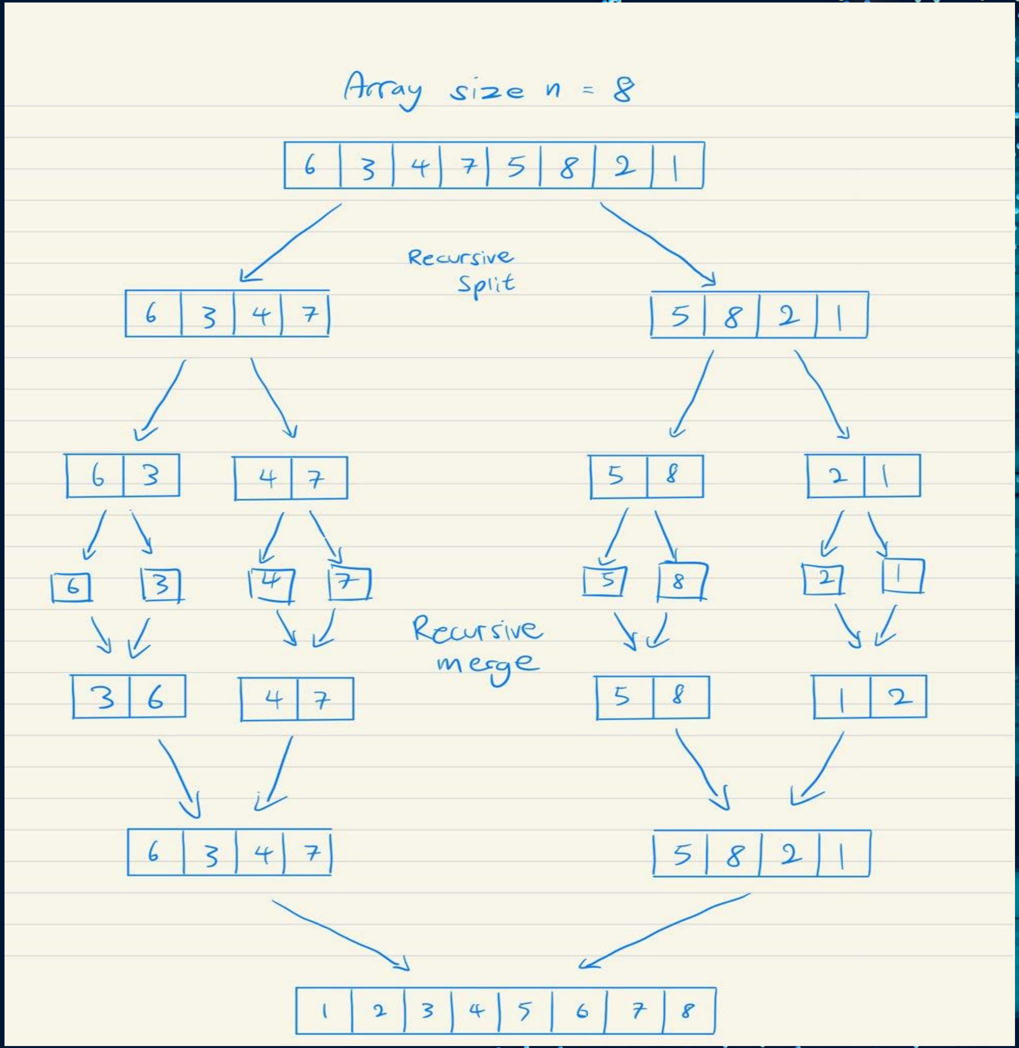
```
merge_sort(list):
        find length of list

        if length of list is equals to 1:
                return the list
        else:
                find mid-point of list
                merge_sort(left subarray, S)
                merge_sort(right subarray, S)

        return merge(left subarray,right subarray)
```

Pseudo-code for Hybrid Sort

```
merge_sort_hybrid(list, S):
        find length of list

        if length of list is smaller than or equals to S:
                perform insertion sort and return the list
        else:
                find mid-point of list
                merge_sort_hybrid(left subarray, S)
                merge_sort_hybrid(right subarray, S)

        return merge(left subarray,right subarray)
```

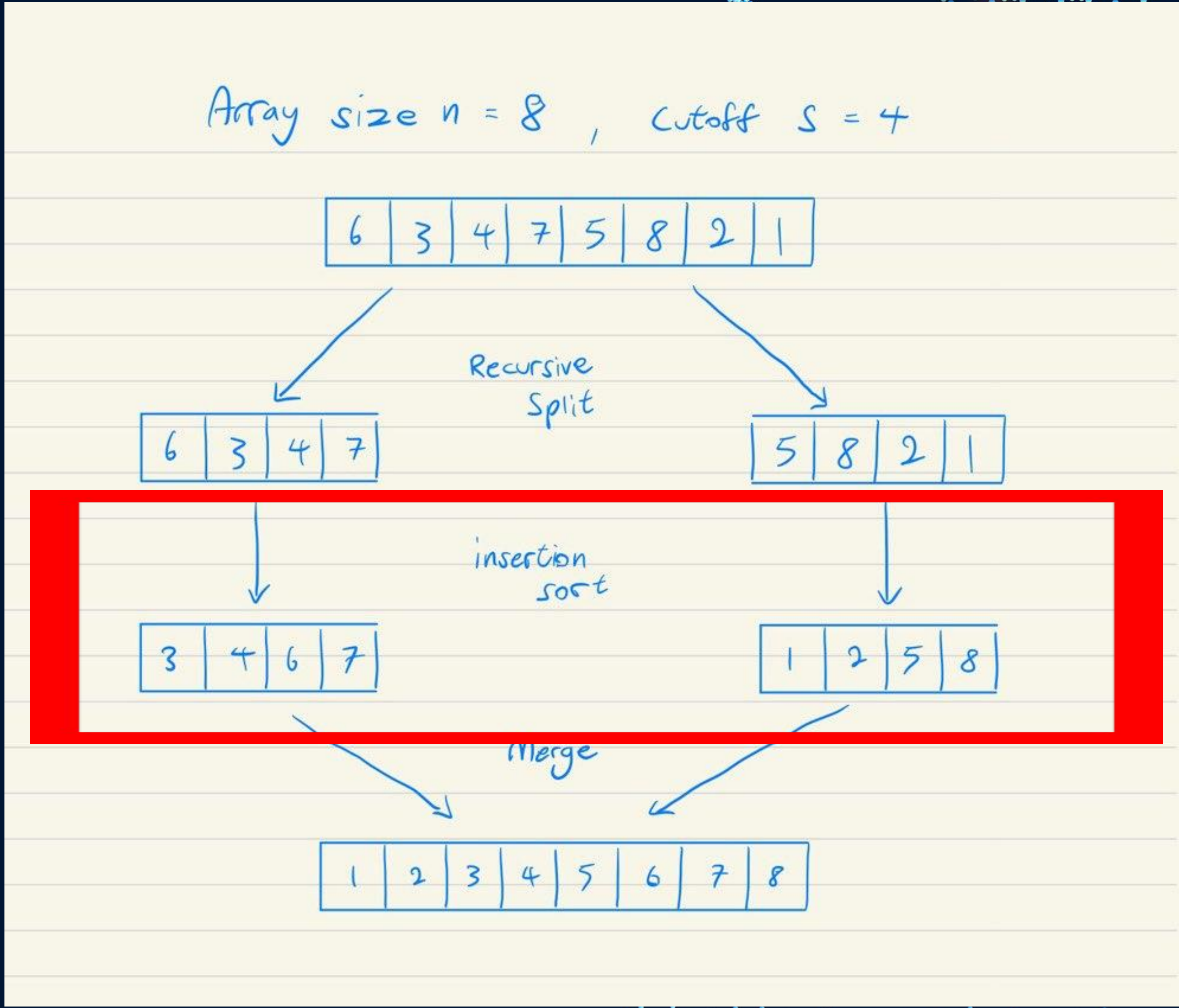
Different base case

Visualisation (Merge Sort)



Visualisation (Hybrid Sort)

Insertion Sort instead of splitting the subarray further, when subarray size = S



Code Implementation for Hybrid Sort

Array : [0]

```
def merge_sort_hybrid(list, S, key_comparisons):
    length = len(list)

    if length <= S:   # base case
        return insertion_sort(list, key_comparisons)

    mid = length // 2

    left = merge_sort_hybrid(list[:mid], S, key_comparisons)[0]
    right = merge_sort_hybrid(list[mid:], S, key_comparisons)[0]

    return merge(left, right, key_comparisons)
```

Code Implementation for Hybrid Sort

```python
def insertion_sort(list, key_comparisons):
    for i in range(1, len(list)):
        current = list[i]
        for j in range(i, 0, -1):
            key_comparisons[0] += 1
            if list[j] < list[j-1]:
                list[j] = list[j-1]
                list[j-1] = current
            else:
                break

    return list, key_comparisons
```

```python
def merge(left, right, key_comparisons):
    output = []
    i = j = 0

    while i < len(left) and j < len(right):
        key_comparisons[0] += 1
        if left[i] < right[j]:
            output.append(left[i])
            i += 1
        else:
            output.append(right[j])
            j += 1

    output.extend(left[i:])
    output.extend(right[j:])

    return output, key_comparisons
```

Sample Output

```
def main():
    unsorted = [99, 0, 5, 20, 123, 0, -1, 72, 21, 22, 13, 8, 7, 67, 29, 1, 2, 4]
    S = 5
    comparisons = merge_sort_hybrid(unsorted, S,[0])
    print(unsorted)
    print(comparisons)

main()
```

```
[99, 0, 5, 20, 123, 0, -1, 72, 21, 22, 13, 8, 7, 67, 29, 1, 2, 4]
([-1, 0, 0, 1, 2, 4, 5, 7, 8, 13, 20, 21, 22, 29, 67, 72, 99, 123], [58])
```

Tuple: (Sorted Array , Number of Key Comps)

Part B

Generating input datasets

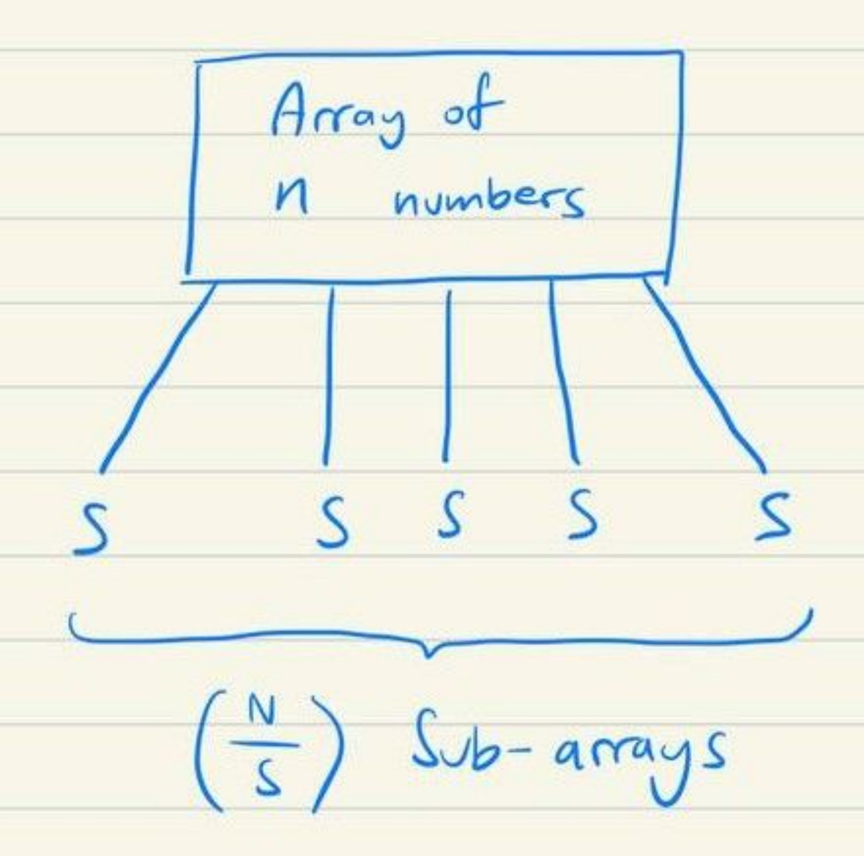
```python
from numpy import random

def rng():
    inputs = []
    inputs.append(random.randint(1000, size=1000))
    for i in range(1, 5):
        size = 1000 * pow(10, i)
        inputs.append(random.randint(size, size=int(size/2)))
        inputs.append(random.randint(size, size=size))

    return inputs

inputs = rng()
for i in inputs:
    print(len(i))
```

1k, 5k, 10k, 50k, 100k, 500k, 1m, 5m, 10m

Part C

Time complexity analysis

Theoretical Analysis of Time Complexity (Explanation)



Insertion Sort:

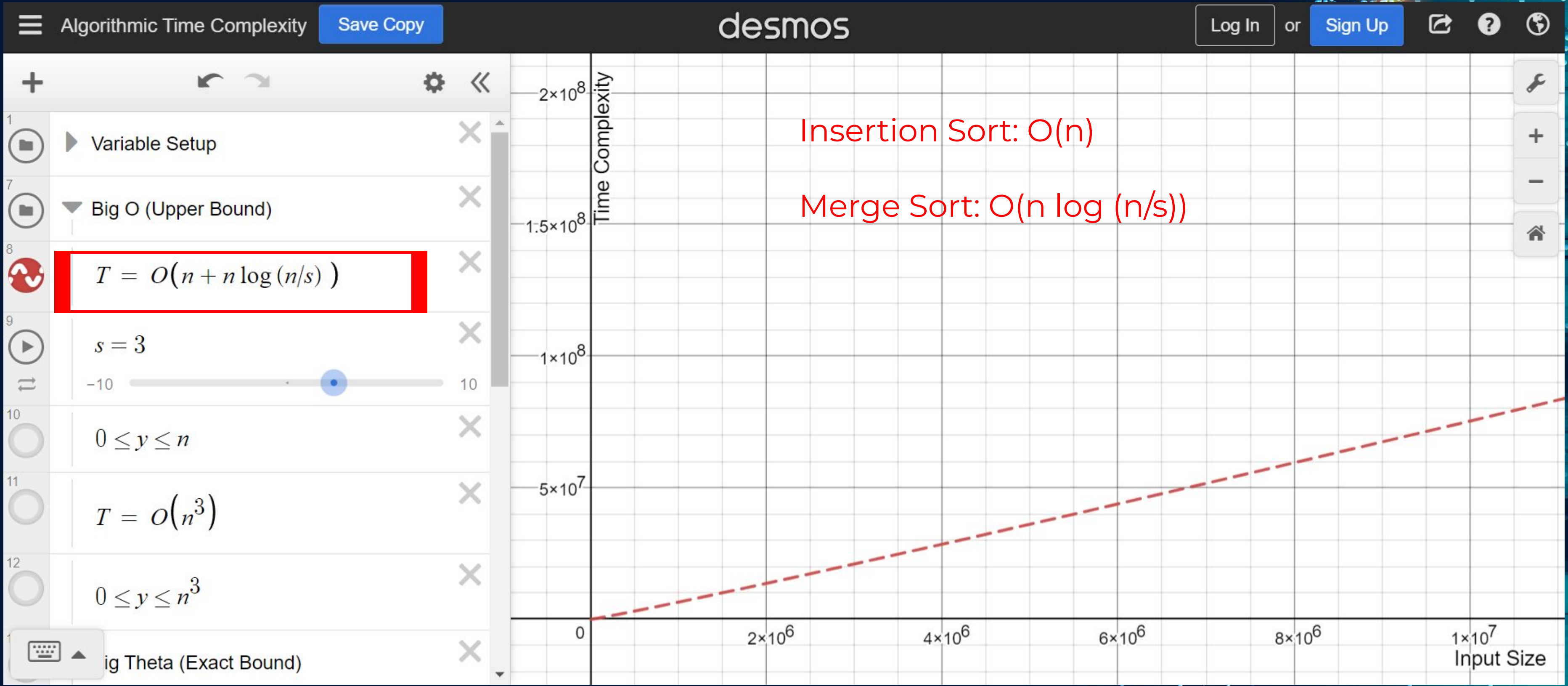
(N/S) * O(S) = O(N) [BEST CASE]

(N/S) * O(S^2) = O(NS) [WORST/AVG]

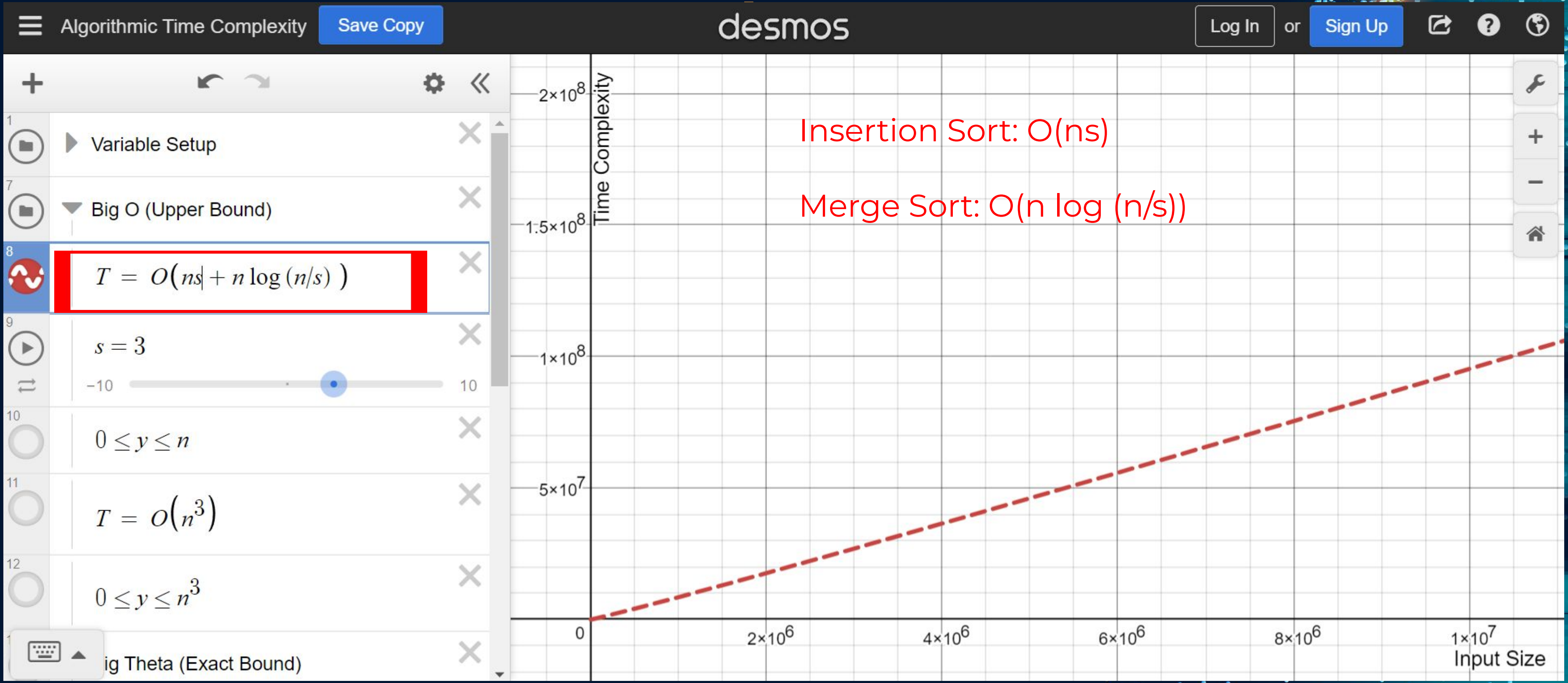
Merge Sort:

O(N log (N/S)) [ALL CASES]

C(i): Theoretical Analysis of Time Complexity (best case)



C(i): Theoretical Analysis of Time Complexity (worst/avg case)



C(ii): Theoretical Analysis of Time Complexity

As S ↑

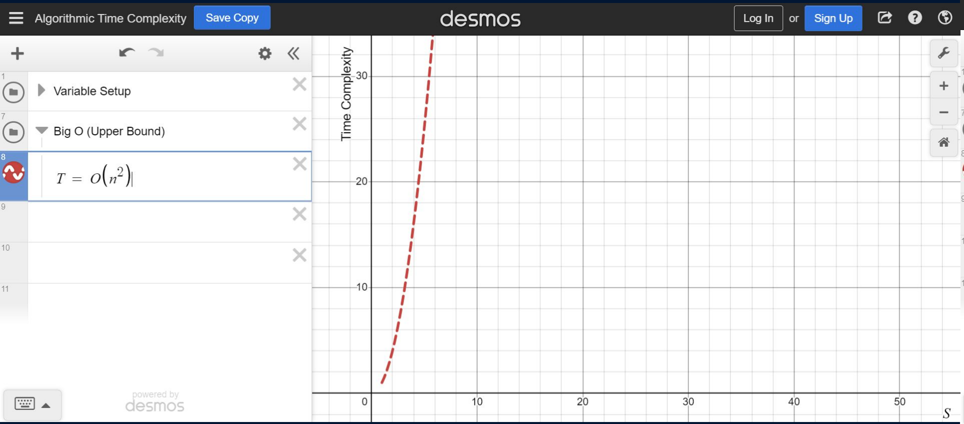
Insertion Sort ↑

Merge Sort ↓

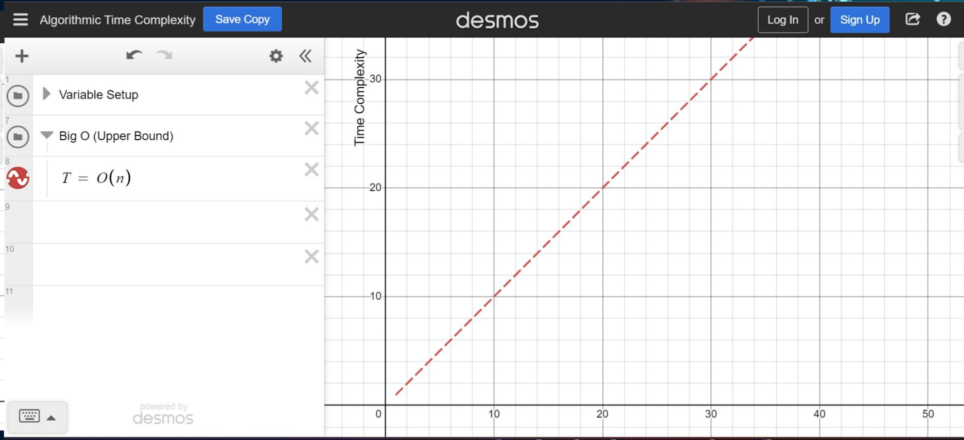
Time complexity will resemble Insertion Sort

C(ii): Theoretical Analysis of Time Complexity

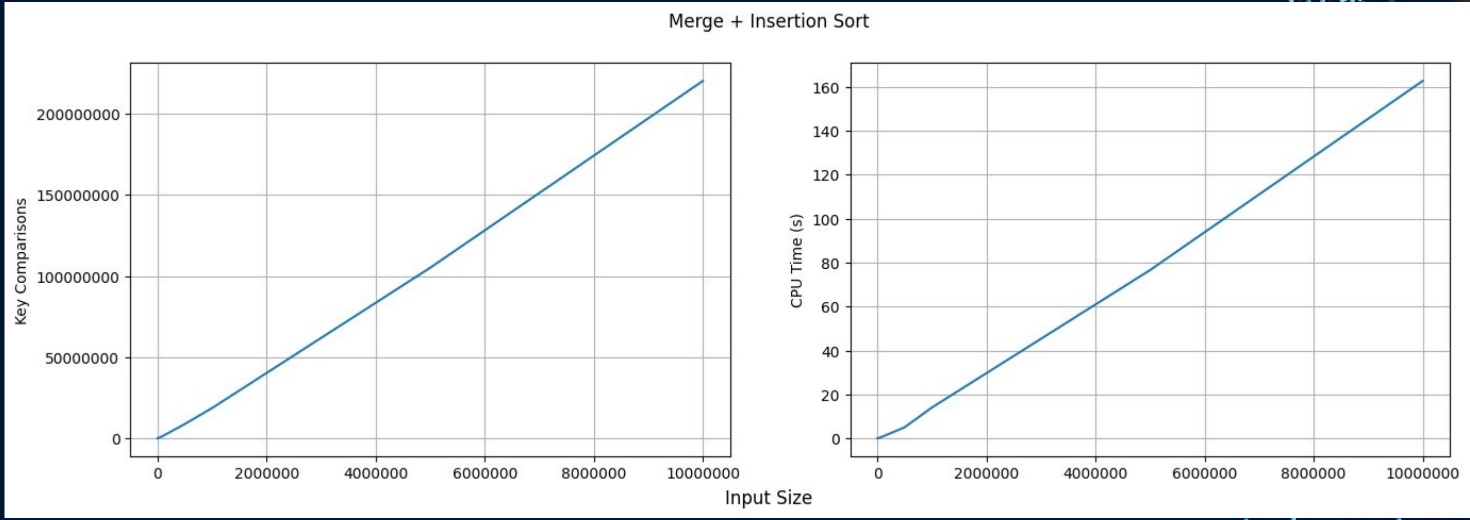
Worst/ Avg case

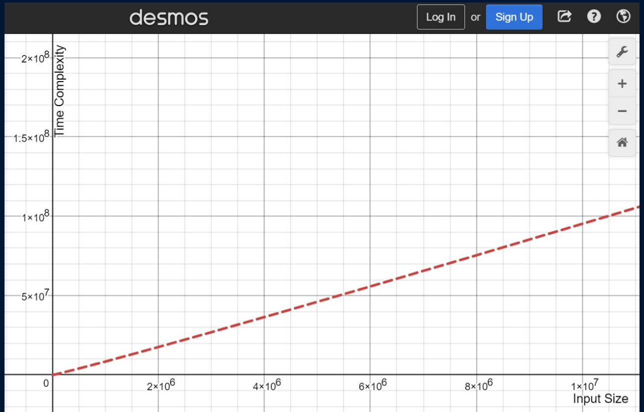


Best case

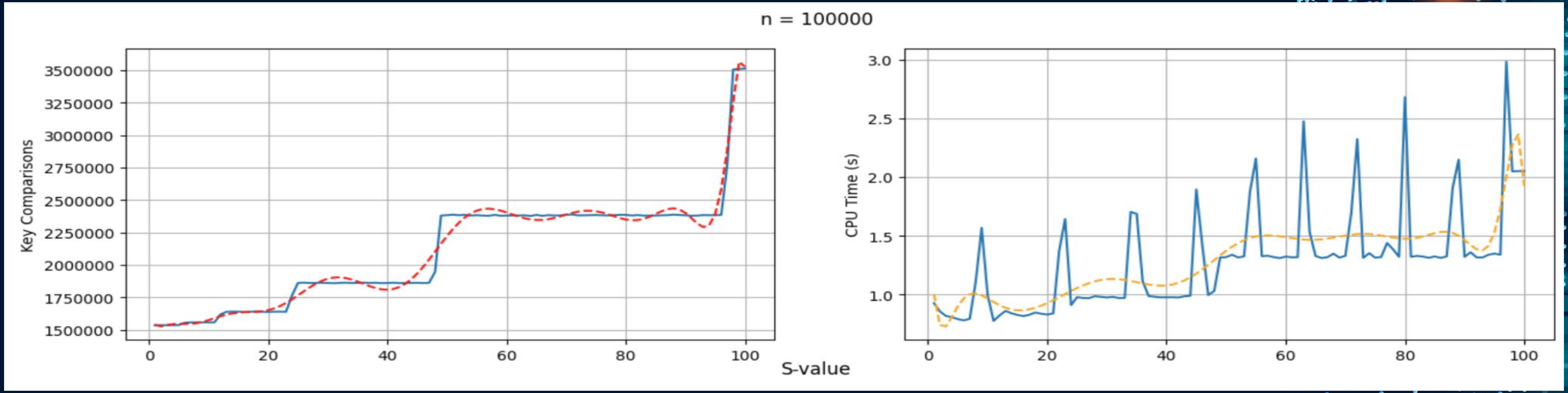


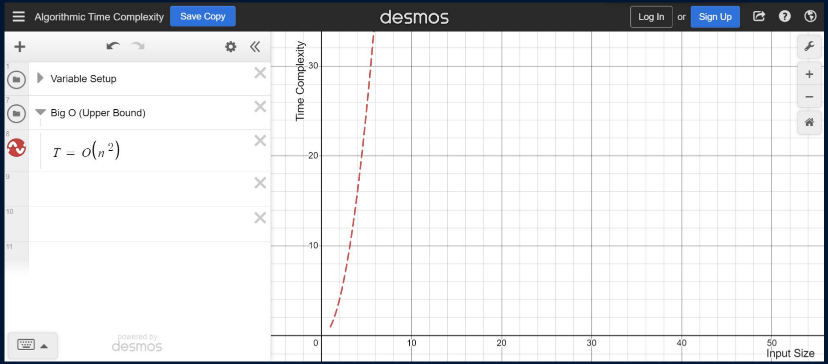
C(i): Empirical vs theoretical

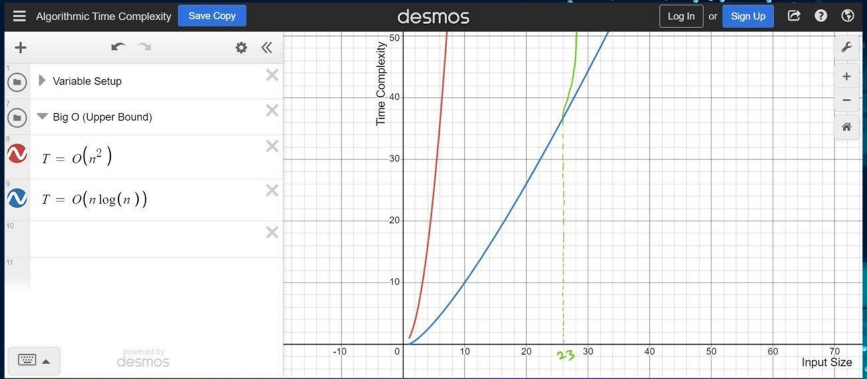




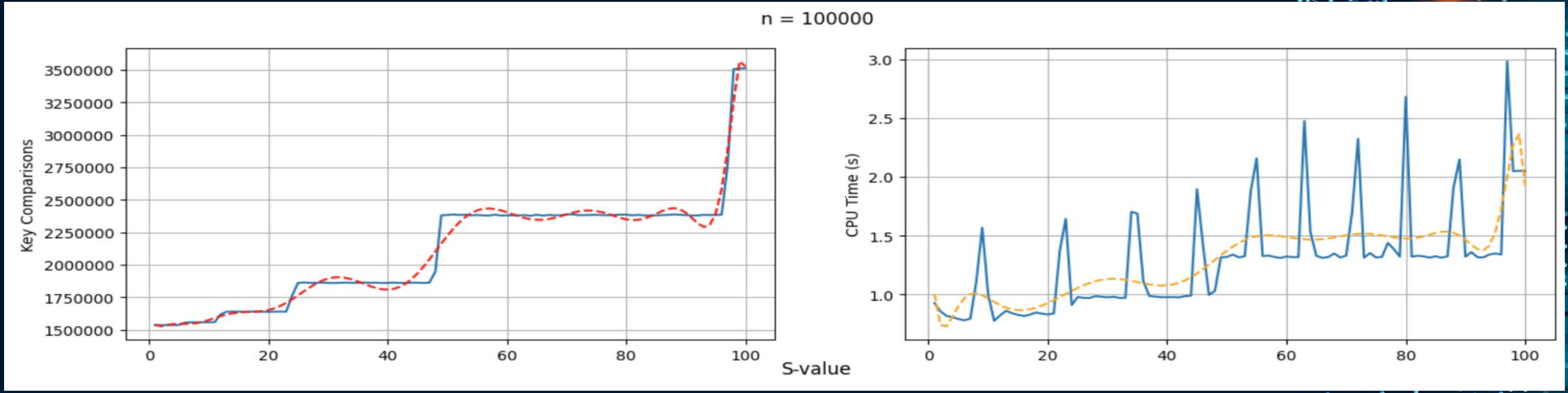
C(ii): Empirical vs theoretical

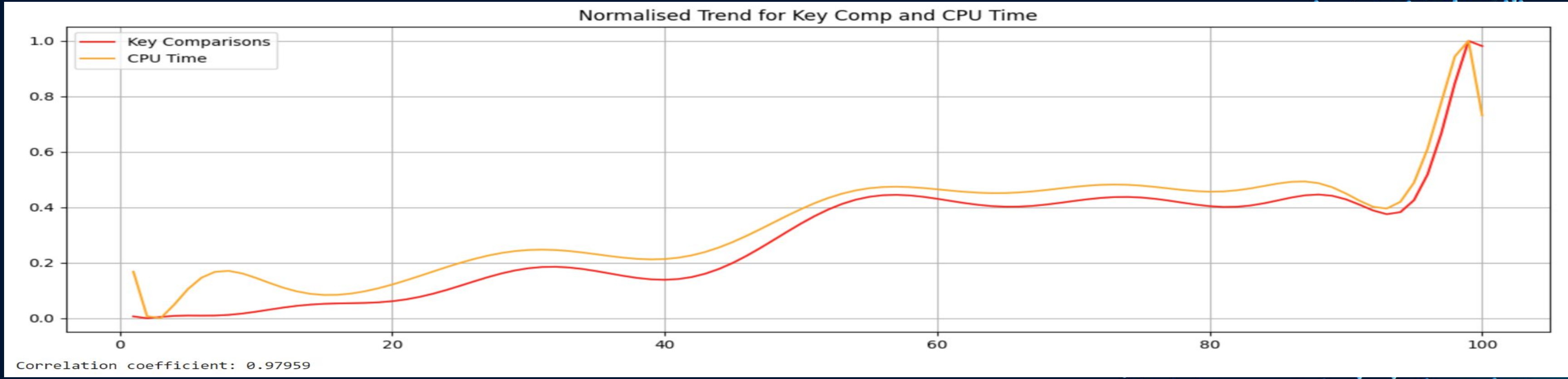




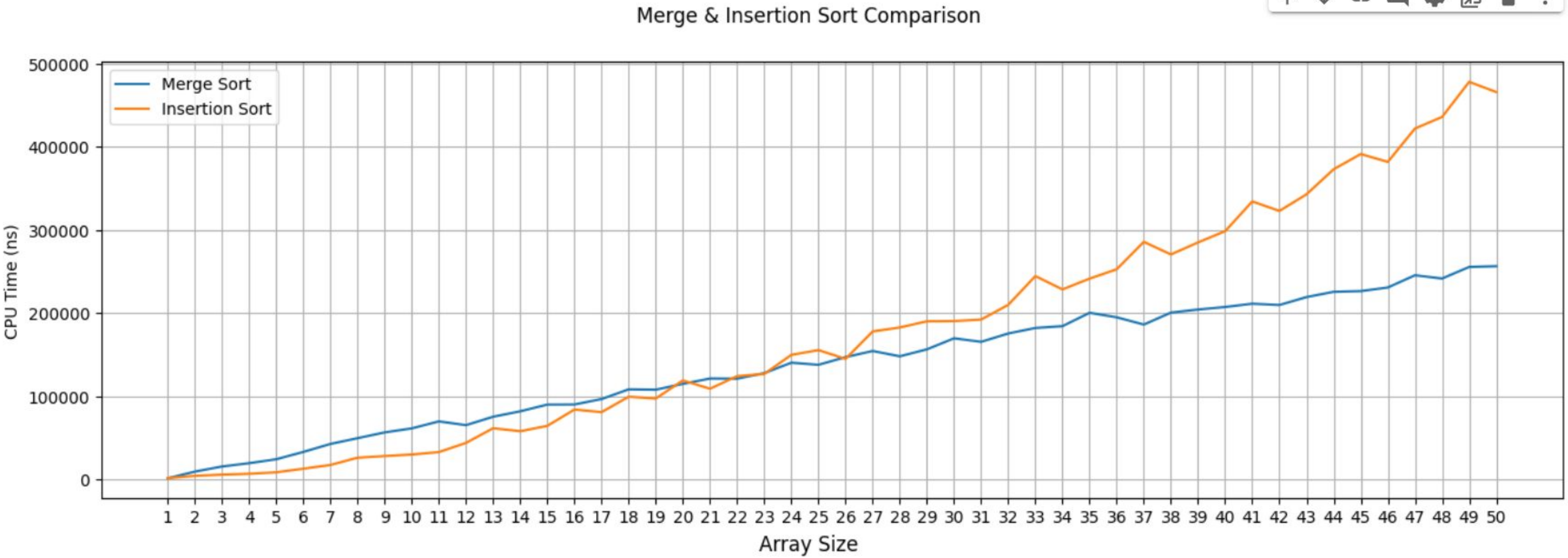


C(ii): Empirical vs theoretical





C(iii): Finding Optimal S Value



C(iii): Finding Optimal S-value

Based on the intersection between the insertion and merge sort algorithm, where the hybrid sort will work best as insertion sort before the optimal S-value, and will work best as merge sort after the optimal S-value. The optimal S-value should be the intersection, which is 23.

Part D

Comparison with original merge sort

Pseudo-code for Merge Sort

```
merge_sort(list):
        find length of list

        if length of list is equals to 1:
                return the list
        else:
                find mid-point of list
                merge_sort(left subarray, S)
                merge_sort(right subarray, S)

        return merge(left subarray,right subarray)
```

Merge Sort Implementation

```python
def merge_sort(list, key_comparisons):
    length = len(list)

    if length == 1:    # base case
        return list, key_comparisons

    mid = length // 2

    left = merge_sort(list[:mid], key_comparisons)[0]
    right = merge_sort(list[mid:], key_comparisons)[0]

    return merge1(left, right, key_comparisons)
```

```python
def merge1(left, right, key_comparisons):
    output = []
    i = j = 0

    while i < len(left) and j < len(right):
        key_comparisons[0] += 1
        if left[i] < right[j]:
            output.append(left[i])
            i += 1
        else:
            output.append(right[j])
            j += 1

    output.extend(left[i:])
    output.extend(right[j:])

    return output, key_comparisons
```

Program for comparison

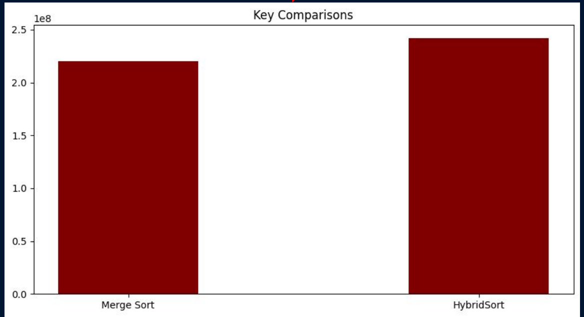
```
import numpy as np
import matplotlib.pyplot as plt
from time import process_time, process_time_ns

# unsorted = [99, 0, 5, 20, 123, 0, -1, 72, 21, 22, 13, 8, 7, 67, 29, 1, 2, 4]
input = random.randint(1000, size=10000000)
# print(input)
merge_start = process_time_ns()
comparisons = merge_sort(input,[0])
merge_stop = process_time_ns()
# print(merge_stop,merge_start)
merge_time = merge_stop - merge_start
print("Merge Sort Time:", merge_time, "ns")

S = 23
hybrid_start = process_time_ns()
sortResult = merge_sort_hybrid(input, S, [0])
hybrid_stop = process_time_ns()
hybrid_time = hybrid_stop - hybrid_start
print("Hybrid Sort Time:", hybrid_time, "ns")
```

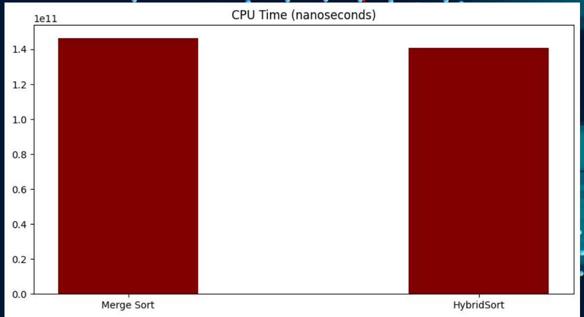
Logs
1. Key Comparisons
2. CPU Time

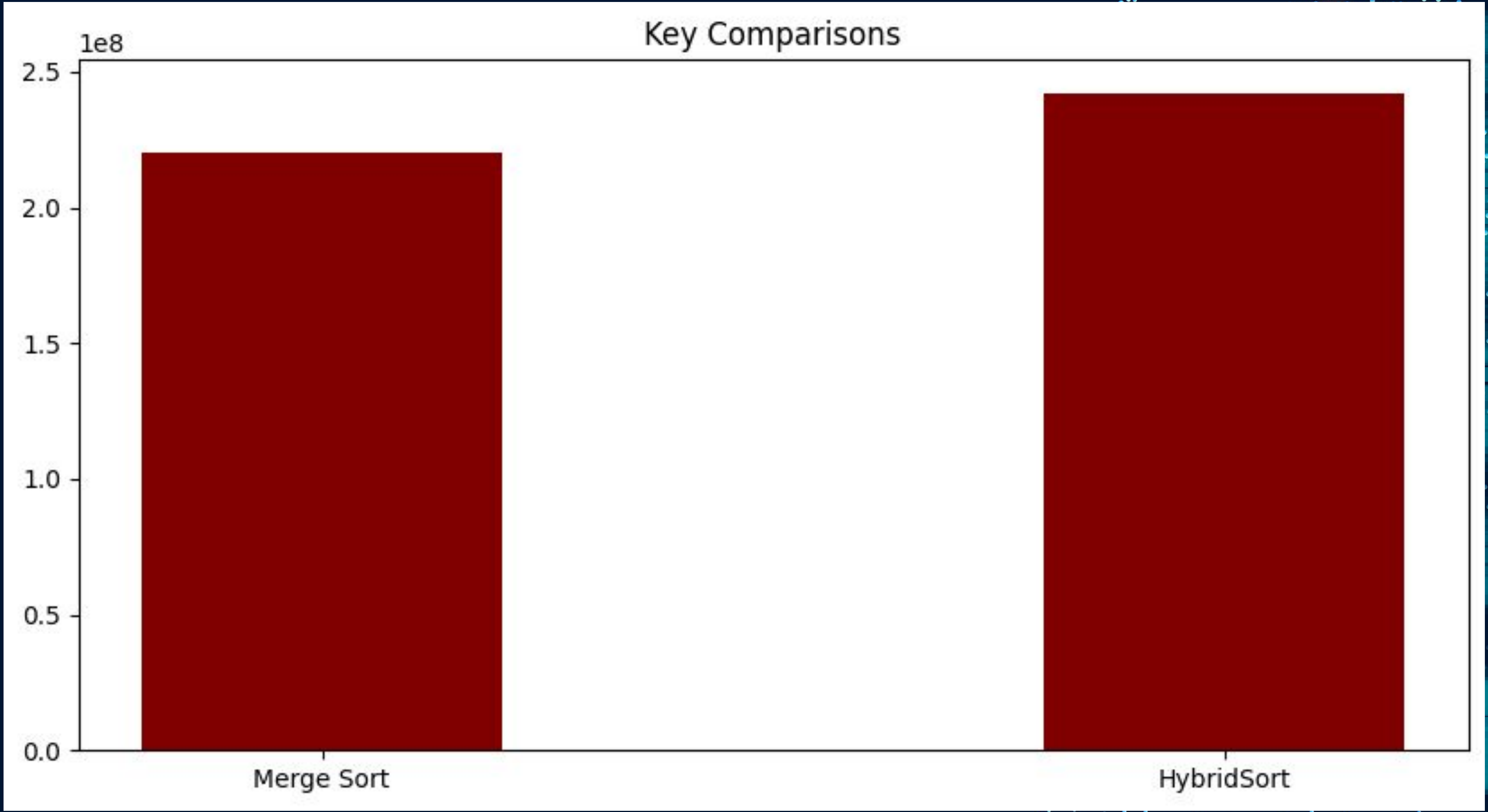
Bar Graph code

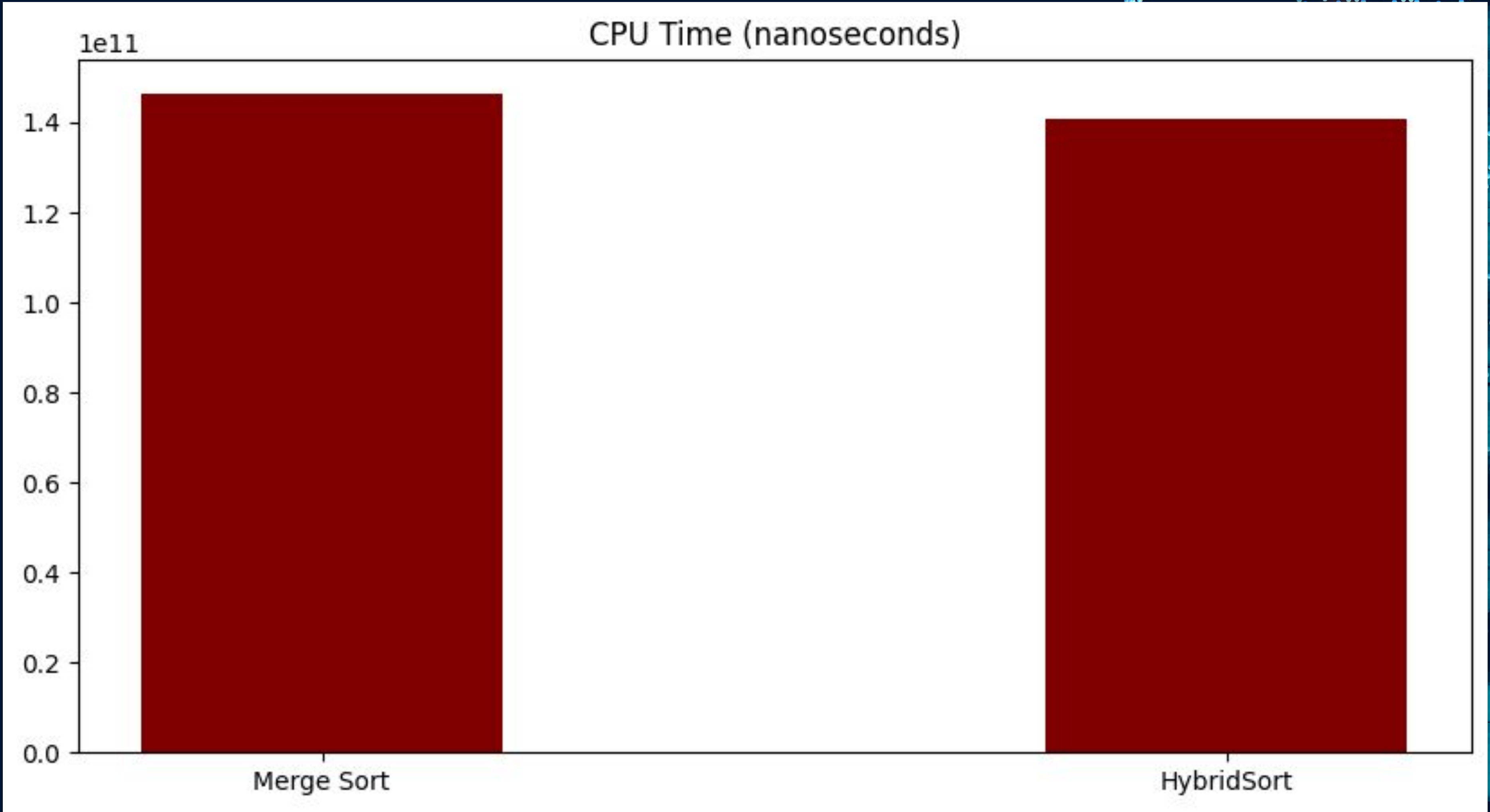
```python
# creating the dataset
data = {'Merge Sort':comparisons[1][0], 'HybridSort':sortResult[1][0]}
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color ='maroon',
    width = 0.4)

plt.xlabel("")
plt.ylabel("")
plt.title("Key Comparisons")
plt.show()
```

```python
# creating the dataset
data = {'Merge Sort':merge_time, 'HybridSort':hybrid_time}
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color ='maroon',
    width = 0.4)

plt.xlabel("")
plt.ylabel("")
plt.title("CPU Time (nanoseconds)")
plt.show()
```






Key Comparisons
1e8
2.5
2.0
1.5
1.0
0.5
0.0
Merge Sort
HybridSort



CPU Time (nanoseconds)
1e11
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Merge Sort
HybridSort

Interpretation

Since insertion sort typically uses more key comparison than merge sort, when the hybrid sort starts to follow the pattern of insertion sort, its key comparison will be more than if merge sort is used.

Insertion sort worst key comparison

$$\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

Merge sort worst key comparison

at most (worst case) $n - 1$

Interpretation

Insertion sort is faster on smaller array of numbers as it skips the sorted values. Thus when the hybrid model is used on the smaller array of numbers and insertion sort is used, the cpu time will be faster than the merge sort model as merge sort will still go through the sorted values.

Thank you!