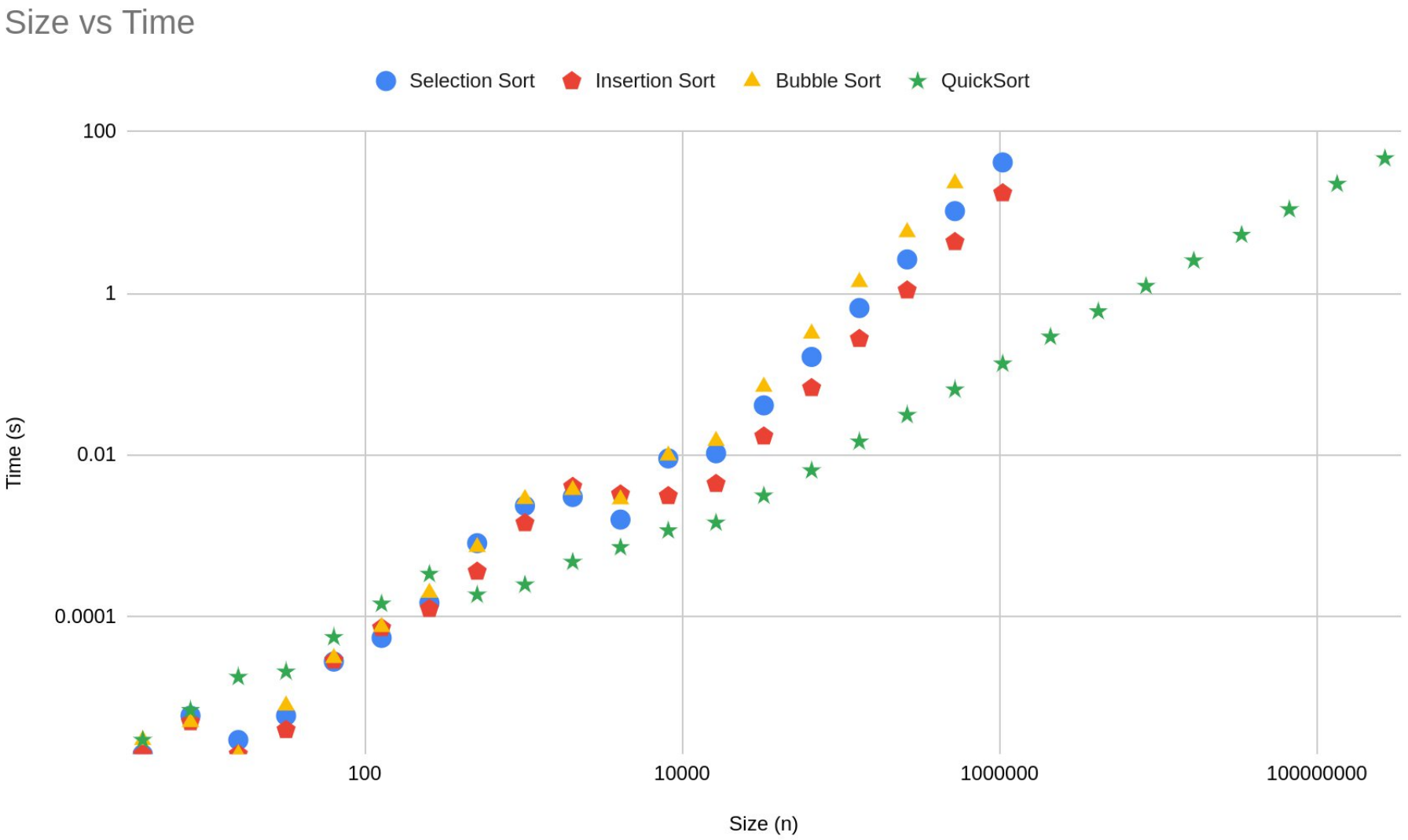
# Assignment 01 - All Sorts of Sorts

Programming II / Tristan Goodell

# {graphs_&_analysis}

## [selection] / [insertion] / [bubble] / [quick]

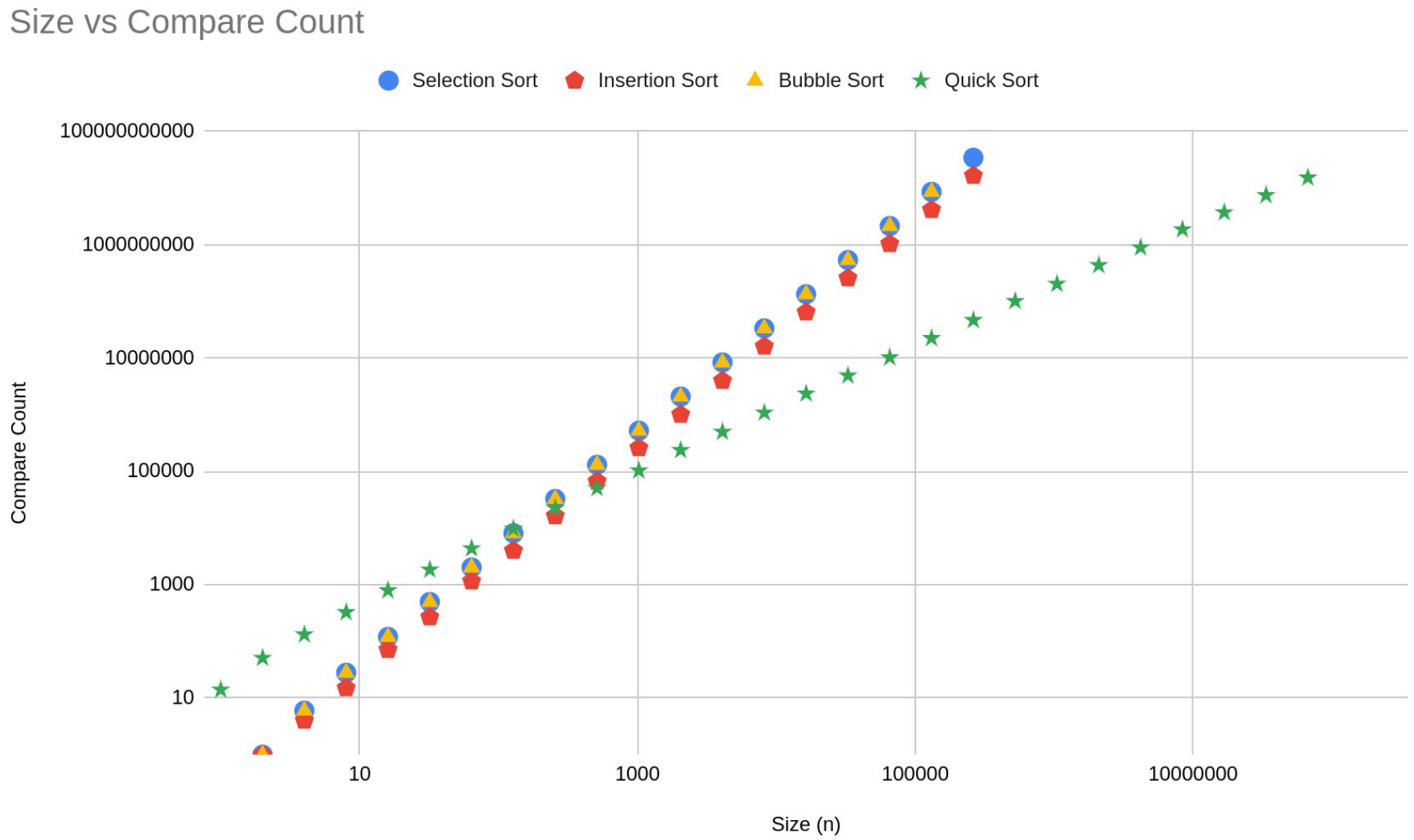
# {time}


Size vs Time
Selection Sort
Insertion Sort
Bubble Sort
QuickSort
100
1
0.01
0.0001
Time (s)
100
10000
1000000
100000000
Size (n)

# {time} / [discussion]

- Although QuickSort is slower than the other sorts for lists smaller than $n=512$, it outpaces the other sorting techniques for larger lists.
- Since QuickSort is faster than the other techniques, it can process lists up to size $n=268435456$ in less than a minute, compared to the other lists only being able to process lists up to size $n=262144$ in less than a minute.

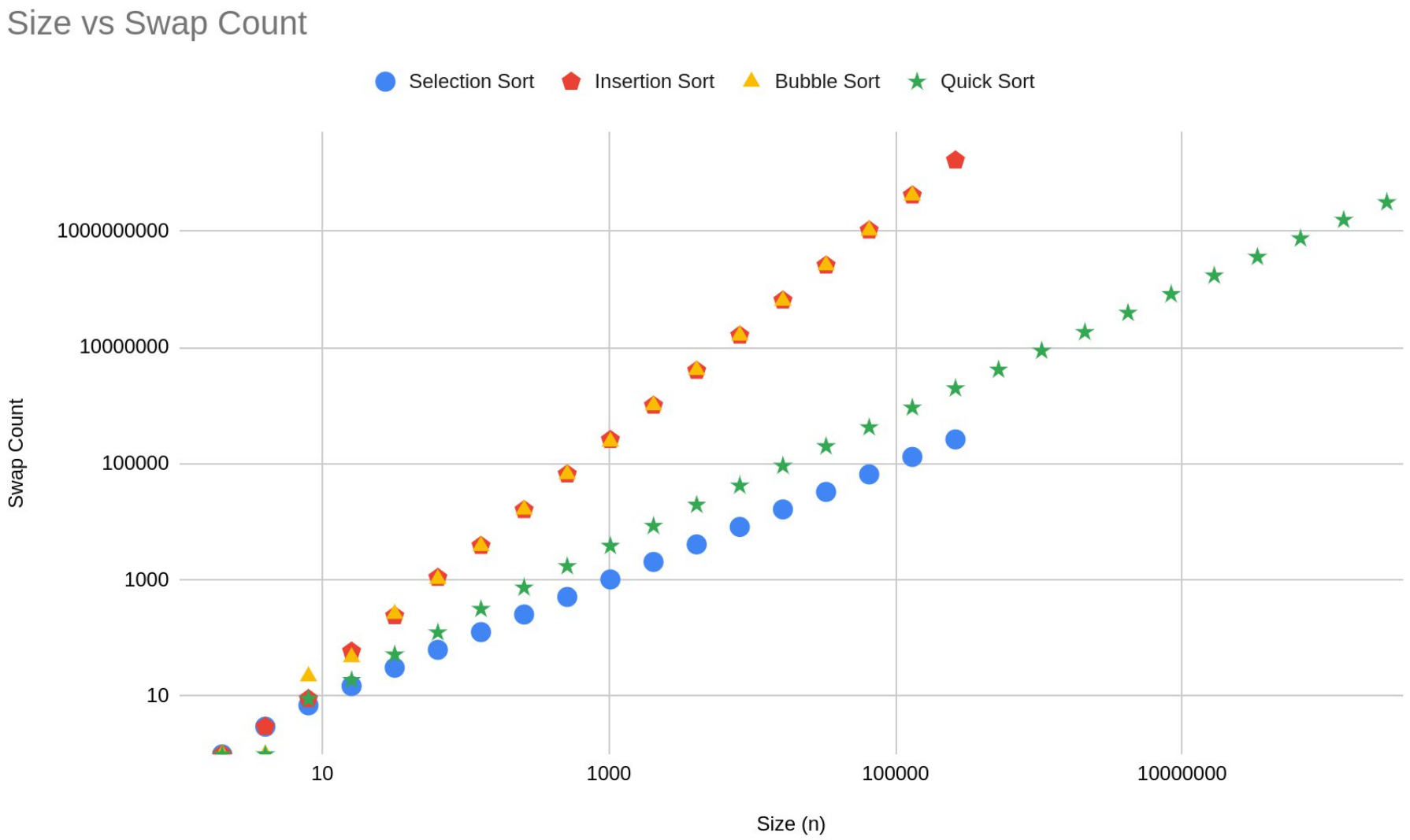
# {compare_count}



# {compare_count} / [discussion]

- Once again, Quick Sort is less efficient than the other sorting techniques with lists with a length less than n=512. After that, Quick Sort becomes much more efficient and requires fewer compares to successfully sort, relative to other sorts.
- The number of Compare Counts for Insertion, Selection, & Bubble sorting is incredibly similar. This is explained by all three sorts comparing values just once per loop.

# {swap_count}


Size vs Swap Count
Selection Sort
Insertion Sort
Bubble Sort
Quick Sort
1000000000
10000000
100000
1000
10
Swap Count
10
1000
100000
10000000
Size (n)

# {swap_count} / [discussion]

- Selection Sort is the superior sorting technique when looking at swap count. This efficiency is caused by Selection Sort only swapping n-1 times.
- Insertion and Bubble Sort swap items approximately the same number of times.
- It took approximately the same number of swaps for Quick Sort to sort a list $n=268435456$ as it took Insertion and Bubble Sort to sort a list $n=262144$.
- Additionally, it took approximately the same number of swaps for Selection Sort to sort a list $n=262144$ as it took Quick Sort to sort a list $n=32768$.

# {code}

## [selection] / [insertion] / [bubble] / [quick]

# {selection_sort} / [code]

```
public void selectionSort()
{
        for (int i=0;i<size-1;i++)
        {
                int minimum = i;
                for(int x=i+1;x<size;x++)
                {
                        if(inOrder(get(x),get(minimum)))
                        {
                                minimum=x;
                        }
                }
                swap(minimum,i);
        }
}
```

# {insertion_sort} / [code]

```
public void insertionSort()
{
        for(int i=1;i<=size-1;i++)
        {
                int j=i;
                while(j>0 && inOrder(get(j-1),get(j)))
                {
                        swap(j,j-1);
                        j--;
                }

        }
}
```

# {bubble_sort} / [code]

```
public void bubbleSort()
{
        for(int i=0;i<size;i++)
        {
                for(int j=1;j<=size-1-i;j++)
                {
                        if(!inOrder(get(j),get(j-1)))swap(j-1,j);
                }
        }
}
```

# {quick_sort} / [code]

```
public int partition(int lo, int hi){
		int pivot=get(lo);
		int i=lo;
		int j=hi;
		while(true) {
			if(get(i)<pivot){
				while(get(i)<pivot){
					i++;
					compareCount++;
				}
			}
			else{
				compareCount++;
			}
			if(get(j)>pivot){
				while(get(j)>pivot){
					j--;
					compareCount++;
				}
			}
			else{
				compareCount++;
			}
			if(i>=j){
				compareCount++;
				return j;
			}
			else{
				compareCount++;
			}
			swap(i,j);
		}
}
```

```
public void quickSort()
{
		quickerSort(0,size-1);
}

public void quickerSort(int lo, int hi)
{
		if (lo>=hi)
		{
			return;
		}
		int pivot=partition(lo, hi);
		quickerSort(lo, pivot);
		quickerSort(pivot+1, hi);
}
```