Christian Winwood and Kosta Sergakis

Assignment 24

GO GO GO

The project that we are doing is evaluating the efficiency of the programming language Go. The reason we chose this is Go seems to be a mix between Python and Java. It has types and brackets like Java but writes fairly similar to Python. To compare the programs, we are creating a merge sort in all of them and timing them. This will show us the performance and efficiency of how they compare. To do this we wrote a merge sort in each of the programming languages.

Motivation and Purpose

We both have knowledge of Java and Python so we just needed to learn Go. Go has some very interesting aspects to it. We were curious to see how readability and writability compared to Java and Python. Go is interesting because it can be written in a way that is very readable more like Java using types or it can also be written in a way like python that gives it better writability but less readability. For example, variables in Go can be declared with or without types and in the following formats: “var testInt int = 1” or “testInt := 1” both of these achieve the same thing in assigning 1 to the variable testInt. Once we had learned a little bit more of the syntax we were able to start out development and testing. We decided we were going to write a mergeSort in all 3 languages and then test them against increasing sequences of numbers. We wanted to see which language was the fastest in completing these tests.

Results

The results we got were quite interesting. We timed each of the different languages as they ran a merge sort on an array that gets longer each time. Or merge sort functions are implemented as follows:

Python

```python
def mergeSort(a):
    if len(a) > 1:
        size = len(a)//2
        left = mergeSort(a[:size])
        right = mergeSort(a[size:])

        arr = []

        while len(left) > 0 and len(right) > 0:
            if left[0] <= right[0]:
                arr.append(left[0])
                left = left[1:]
            else:
                arr.append(right[0])
                right = right[1:]

        if len(left) > 0:
            for i in left:
                arr.append(i)
        if len(right) > 0:
            for i in right:
                arr.append(i)
        return arr
    return a
```

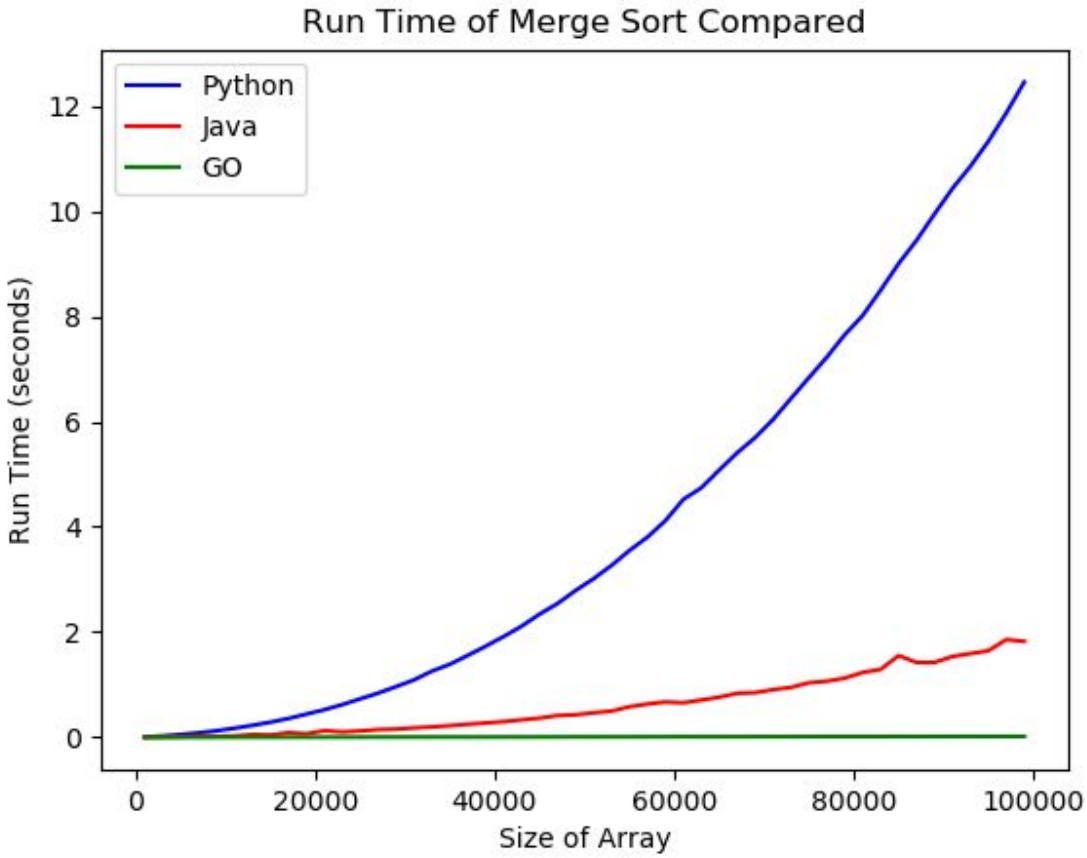
Java

```java
private static int[] sort(int[] sequence){
    if (sequence.length == 1){
        return sequence;
    }
    int mid = (int)(sequence.length / 2);
    int[] leftSeq = Arrays.copyOfRange(sequence, 0, mid);
    int[] rightSeq = Arrays.copyOfRange(sequence, mid, sequence.length);

    return merge(sort(leftSeq), sort(rightSeq));
}

private static int[] merge(int[] l, int[] r){
    int[] result = new int[l.length + r.length];

    int i = 0;

    while (l.length > 0 && r.length > 0){
        if (l[0] < r[0]){
            result[i] = l[0];
            l = Arrays.copyOfRange(l, 1, l.length);
        }
        else {
            result[i] = r[0];
            r = Arrays.copyOfRange(r, 1, r.length);
        }
        i++;
    }
    for(int j = 0; j < l.length; j++){
        result[i] = l[j];
        i++;
    }
    for(int j = 0; j < r.length; j++){
        result[i] = r[j];
        i++;
    }

    return result;
}
```

Go

```go
func merge(l, r []int) (result []int) {
    result = make([]int, len(l)+len(r))

    i := 0

    for len(l) > 0 && len(r) > 0 {
        if l[0] < r[0] {
            result[i] = l[0]
            l = l[1:]
        } else {
            result[i] = r[0]
            r = r[1:]
        }
        i++
    }
    for j := 0; j < len(l); j++ {
        result[i] = l[j]
        i++
    }
    for j := 0; j < len(r); j++ {
        result[i] = r[j]
        i++
    }

    return
}

func sort(sequence []int) []int {
    if len(sequence) == 1 {
        return sequence
    }

    mid := int(len(sequence) / 2)
    var leftSeq = make([]int, mid)
    var rightSeq = make([]int, len(sequence)-mid)

    for i := 0; i < len(sequence); i++ {
        if i < mid {
            leftSeq[i] = sequence[i]
        } else {
            rightSeq[i-mid] = sequence[i]
        }
    }

    return merge(sort(leftSeq), sort(rightSeq))
}
```

They are all implemented fairly similar. However, the results we got were very interesting. We initially guessed that Java would take the longest with Go and Python being fairly close. After timing each program and then using a small python script to graph all the results we were very surprised. The results are as follows:



As you can tell, the python script took much longer to run than either Go or Java. Go was actually surprisingly fast and showed almost no signs of increasing in speed. I have no idea what was causing the Python Merge Sort to run so inefficiently but it was all over the place. It seems like Go really does take the best of Python and Java and puts them together into a really efficient and easy language.

Conclusion

In conclusion, we learned a lot from this experiment. First, we got to learn Go, an extremely efficient and fun language. It is very writeable, has the readability of Java,

and is quite a bit more efficient than Java. I would say probably close to C++. This was really fun to be able to see how the same code in different languages can run so drastically differently.