Q6
Please write a modified insertion sort algorithm to reduce the total number of comparisons (the total number of other operations may increase). Explain how your approach reduces the total number of comparisons and discuss its drawbacks compared to the original insertion sort.

Solution

To reduce total number of comparisons, we can use binary search to find where we should insert the current number in the sorted part of the array, instead of previously used linear search.

Linear search iterates over the sorted list so in the worst case it will run n comparisons. On the other hand, binary search checks the middle element. It splits the list into two, decides whether search element is bigger than or equal to middle element. If it is bigger or equal, it continues right list if it is not, it continues left. Since we can split a list with n elements log(n) times until we reach a list with single element, we will make log(n) comparisons in the worst case.

```
procedure binary_insertion_sort(arr[0:n-1])
        for i <- 1 to (length(arr) - 1) do
                target <- arr[i]
                low <- 0
                high <- i - 1

                while low <= high do
                        mid <- (low + high) // 2
                        if target < arr[mid] do
                                high <- mid - 1
                        else do
                                low <- mid + 1
                        end if
                end while
                for j <- i down to (low + 1) by -1 do
                        arr[j] <- arr[j-1]
                end for
                arr[low] <- target
        end for
end procedure
```

Drawbacks

- If data is already sorted or nearly sorted, we will be doing more comparisons than linear search since linear search starts comparing elements from the end, so it will terminate early. At the best-case (sorted array) linear search implementation will have $O(n)$, however binary search implementation will have $O(n * log(n))$ complexity (there will be only 1 swap in shifting but we will still be doing log(n) comparisons for each element).

- When we implement insertion sort with binary search, the basic operation becomes shifting the elements instead of the comparison. Comparison has $O(log(n))$ complexity but shifting has $O(n)$ so our overall worst-case complexity is still $O(n^2)$.