1 Insertion sort algorithm and proof

1.1 Algorithm

Algorithm 1 Insertion Sort

```
Require: array of comparable elements
function INSERTION_SORT(array)                    ▷ 0 indexed array assumed
    for $j = 1$ to $array.length - 1$ do
        $key = array[j]$;
        $i = j - 1$;
        while $i \geq 0$ and $array[i] > key$ do
            $array[i + 1] = array[i]$;
            $i = i - 1$;
        end while
        $array[i + 1] = key$;
    end for
end function
```

1.2 Proof

Invariant: The subarray array[0..j-1], is always in sorted order.

Initialization: Initially we have j = 1, so the array[0] (a single element) is always sorted.

Maintenance: For loop works by moving forward from j = 1, in each iteration, it searches for the position of current key then places it there while shifting every value down the line by one.

Termination: when the loop terminates, j = array.length = n, so array[0..n-1] is in sorted order, which is the entire space value of array. So we conclude that entier array is sorted.

Time Complexity	**Best Case**	**Worst Case**
	$\Theta(n)$	$\Theta(n^2)$

2 Linear search algorithm and proof

2.1 Algorithm

Algorithm 2 Linear Search Algorithm

```
Require: Array to search given element in and the value to search for
1: function LINEAR_SEARCH($array, value$)
2:     for $index = 0$ to $array.length - 1$ do
3:         if $array[index] == value$ then
4:             return $index$;
5:         end if
6:     end for
7:     return $index$;
8: end function
```

2.2 Proof

Invariant: The value is not within the range array[0..index-1]

Initialization: Initially we have index = 0, so the value isn't within the array range array[0..-1] which is empty arrray.

Maintenance: Within each for loop if the value pointed by index isn't equal to the value, we increment otherwise we return the index and terminate the function. In either case, the value will not be contained within the range array[0..index-1].

Termination: The return value of index could either point to the value itself, or point to the value one after the size limit of array. In former case, since the value was first found at that index, there is no possibility for the same value to exist within the sub-array, array[0..index-1]. In later case, the value of index is one plus the total size of array. This implies, that the value isn't contained within the array at all. So, we conclude that linear search will always aware us about the presence or absence of the value in the given array.

Time Complexity	**Best Case**	**Worst Case**
	$\Theta(1)$	$\Theta(n)$

3 Selection sort

3.1 Algorithm

Algorithm 3 Selection sort algorithm

```
function SELECTION_SORT($array$)
    for $i = 0$ to $array.length - 2$ do
        index = i;
        for $j = i + 1$ to $array.length - 1$ do
            if $array[index] > array[j]$ then
                index = j;
            end if
        end for
        SWAP($array[index], array[i]$);
    end for
end function
```

3.2 Proof

Invariant: The sub array array[0..i-1] is always in sorted order.

Initialization: Initially we have i = 1, so the array[0] (a single element) is always sorted.

Maintenance: In each iteration, we move in the array by one position. At each position we find the smallest value since that position then swap the value at given position with found smallest value. This ensures that the array[0..i-1] is always sorted in ascending order. We only need to move by n-1 position, as the final will be sorted implicitly.

Termination: When the loop ends, we have i = array.length, so array[0..i-1] which comprises of the entire array is sorted. Hence we can conclude that the entier array is sorted.

Time Complexity	Best Case	Worst Case
	$\Theta\left(n^2\right)$	$\Theta\left(n^2\right)$