ALGORITHMS AND DATA STRUCTURES

CS106.3

1. **Using an appropriate example, explain how main searching algorithms can be performed.**
   - Linear Search: Linear search is a simple and straightforward searching algorithm. It works by sequentially checking each element in a list or array until the target element is found or the end of the list is reached.

   - Binary Search: Binary search is a more efficient searching algorithm that requires the input array to be sorted in ascending order. It works by repeatedly dividing the search space in half until the target element is found or the search space is empty. Binary search is particularly useful when dealing with large, sorted datasets.

2. **Compare (look for similarities) and contrast (look for differences) linear search and algorithms.**

   Similarities:

   - Both algorithms are used for searching elements in a list or array.
   - They can be implemented in various programming languages.
   - They return the index of the found element or a specific value (e.g., -1) to indicate that the element is not present.

   Differences:

   - Linear search can be performed on both sorted and unsorted arrays, while binary search requires a sorted array.
   - Linear search has a time complexity of O(n), where n is the number of elements in the array, as it checks each element in sequence. Binary search, on the other hand, has a time complexity of O(log n), providing a faster search for sorted arrays.
   - Linear search does not require any pre-processing steps, while binary search requires the array to be sorted beforehand.

3. **Write a function using pseudo or source codes for searching an integer variable called an item using linear search in an array called an unordered array.**

```
function linear_search_unordered_array(arr, item):
    for i = 0 to length(arr) - 1 do
        if arr[i] == item then
            return i
    return -1
```

4. **Write the C program for Binary search.**

```
#include <stdio.h>

int binary_search(int arr[], int low, int high, int item) {
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == item) {
            return mid;
        }
        else if (arr[mid] < item) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }
    return -1;
}

int main() {
    int sorted_array[] = {1, 2, 5, 7, 9};
    int size = sizeof(sorted_array) / sizeof(sorted_array[0]);
    int item_to_find = 7;
    int index = binary_search(sorted_array, 0, size - 1, item_to_find);
    if (index != -1) {
        printf("Element %d found at index %d\n", item_to_find, index);
    }
    else {
        printf("Element not found in the array\n");
    }

    return 0;
}
```

5. **Briefly explain bubble sort, selection sort, and insertion sort.**

➢ Bubble Sort: Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the entire list is sorted.

➢ Selection Sort: Selection sort works by dividing the input into a sorted and an unsorted region. The algorithm repeatedly finds the smallest (or largest) element from the unsorted region and swaps it with the first element of the unsorted region. This process continues until the entire array is sorted.

➢ Insertion Sort: Insertion sort builds the final sorted array one element at a time by iterating through the input array. It maintains a sorted region in the beginning and considers each element from the unsorted region, placing it in its correct position within the sorted region by shifting elements as necessary.