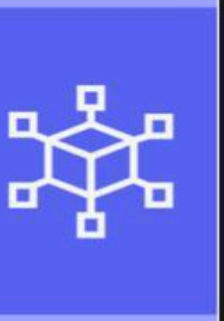
III-I



DATA STRUCTURES

Introduction to DS, Searching & Sorting

Module 1 QB Solutions Handbook

Ujjwal | Vishal

Module 1

Part A

Q1) If there are 22,049 data elements being searched, what is the maximum number of ”looks” it will take with binary search to find the data element being searched for.

A. At most it will take 15 "looks". The log2 (22049) is approximately 14.4.

Q2) Explain the importance of data structures and discuss typical algorithm complexities of different problems? Write the best, average, and worst-case analysis of linear search and binary search algorithms.

A. Data structure provides the right way to organise information in the digital space. The data structure is a key component of Computer Science and is largely used in the areas of Artificial Intelligence, operating systems, graphics, etc. The performances of algorithms can be measured on the scales of Time and Space.

● The Time Complexity of an algorithm or a program is a function of the running time of the algorithm or a program.

● The Space Complexity of an algorithm or a program is a function of the space needed by the algorithm or program to run to completion.

In linear search, best-case complexity is O(1) where the element is found at the first index. Worst-case complexity is O(n) where the element found at the last index of the element is not present in the array and average-case complexity is O(n). In binary search, best-case complexity is O(1) where the element is found at the middle index. The worst-case complexity is O(logn). The average case is O(logn).

Q3) Suppose an array A with elements indexed 1 to n is to be searched for a value x. Write pseudo code that performs a forward search, returning n + 1 if the value is not found.

A. Pseudo Code

```
int forwardSearch(int[] a, int x) {
   int n = a.length;
```

```
    for (int i = 1; i <= n; i++) {
        if (a[i] == x) {
            return i;
        }
    }
    return n + 1;
}
```

Q4) 4. Searching in a phone book: A phone book is stored in a text file, containing names of people, their city names and phone numbers. Choose an appropriate data structure to search a person's phone number based on his / her first name and city.

A. Didn't get it. Please share in #we-code

Q5) Sorting a phone book: Given a text file containing people's names, their city and phone numbers. Write a program which prints all the city names in an alphabetical order and for each one of them print their names in alphabetical order and their corresponding phone number.

A.

```
import java.io.*;
import java.util.*;

public class PhoneBookSorter {
    public static void main(String[] args) {
        // Map to store city names as keys and a list of people as values
        Map<String, List<Person>> cityMap = new TreeMap<>();

        try (BufferedReader br = new BufferedReader(new FileReader("phonebook.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] details = line.split(",");
                String name = details[0].trim();
                String city = details[1].trim();
                String phoneNumber = details[2].trim();

                cityMap.putIfAbsent(city, new ArrayList<>());
```

```
                cityMap.get(city).add(new Person(name, phoneNumber));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Print cities and their corresponding people in alphabetical order
        for (Map.Entry<String, List<Person>> entry : cityMap.entrySet()) {
            System.out.println("City: " + entry.getKey());
            List<Person> people = entry.getValue();
            people.sort(Comparator.comparing(Person::getName));
            for (Person person : people) {
                System.out.println("Name: " + person.getName() + ", Phone Number: " +
person.getPhoneNumber());
            }
            System.out.println();
        }
    }
}

class Person {
    private String name;
    private String phoneNumber;

    public Person(String name, String phoneNumber) {
        this.name = name;
        this.phoneNumber = phoneNumber;
    }

    public String getName() {
        return name;
    }

    public String getPhoneNumber() {
        return phoneNumber;
```

```
    }
}
```

Q6) What is a binary search and write the pseudocode for binary search.

A. Binary search is a fast search algorithm with run-time complexity of O(log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in sorted form.

Procedure of Binary search
Binary search is implemented using the following steps...

```
function binarySearch(arr, target):
    left = 0
    right = len(arr) - 1

    while left <= right:
        mid = (left + right) / 2

        if arr[mid] == target:
            return mid
        else if arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1
```

Q7) . Given an array A of non-negative integers of size m. Your task is to sort the array in non-decreasing order and print out the original indices of the new sorted array.

A. Program

```
 import java.util.Arrays;
import java.util.Comparator;

public class SortWithIndices {
```

```
    public static void main(String[] args) {
        int[] A = {4, 5, 3, 7, 1};
        Integer[] indices = new Integer[A.length];

        // Initialize the indices array
        for (int i = 0; i < A.length; i++) {
            indices[i] = i;
        }

        // Sort the indices array based on the values in A
        Arrays.sort(indices, new Comparator<Integer>() {
            @Override
            public int compare(Integer i1, Integer i2) {
                return Integer.compare(A[i1], A[i2]);
            }
        });

        // Print the sorted indices
        for (int index : indices) {
            System.out.print(index + " ");
        }
    }
}
```

Q8) Consider the following list of integers: [12,9,3,14,5,66,7,80,9,10] and arrange the elements in descending order using insertion sort.

A. Program

```
public class InsertionSortDescending {
    public static void main(String[] args) {
        int[] arr = {12, 9, 3, 14, 5, 66, 7, 80, 9, 10};
        insertionSortDescending(arr);
        for (int num : arr) {
            System.out.print(num + " ");
        }
```

```
    }

    public static void insertionSortDescending(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] < key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
    }
}
```

Q9) Consider the following list of integers: [1,9,33,47,5,6,7,80,9,10] and write the procedure for finding the element '7' using binary search.

A. Procedure of Binary search

Binary search is implemented using the following steps…

Step 1 - Read the search element from the user

. Step 2 - Find the middle element in the sorted list.

Step 3 - Compare the search element with the middle element in the sorted list.

Step 4 - If both are matched, then display "Given element is found!!!" and terminate the function.

Step 5 - If both are not matched, then check whether the search element is smaller or larger than the middle element.

Step 6 - If the search element is smaller than the middle element, repeat steps 2, 3, 4, and 5 for the left sublist of the middle element.

Step 7 - If the search element is larger than the middle element, repeat steps 2, 3, 4, and 5 for the right sublist of the middle element.

Step 8 - Repeat the same process until we find the search element in the list or until the sublist contains only one element.

Step 9 - If that element also doesn't match with the search element, then display "Element is not found in the list!!!" and terminate the function.

Q10) Define insertion sort and write the pseudocode for insertion sort.

A. Insertion sort is the sorting mechanism where the sorted array is built having one item at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order. The analogy can be understood from the style of arranging a deck of cards.

Pseudo Code:

```
// Pseudocode for Insertion Sort
void insertionSort(int[] array) {
    int n = array.length;
    for (int i = 1; i < n; ++i) {
        int key = array[i];
        int j = i - 1;

        while (j >= 0 && array[j] > key) {
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = key;
    }
}
```

PART B

Q1) What are internal and external sorting techniques? Given an unsorted array arr[0..n-1] of size n, find the minimum length subarray arr[s..e] such that sorting this subarray makes the whole array sorted.

A. Internal Sorting

- Internal sorting is a type of sorting which is used when the entire collection of data is small enough that sorting can take place within the main memory. There is no need for external memory for the execution of sorting programs.
- It is used when the size of the input is small

● Examples:- Bubble sort, insertion sort, quicksort, heapsort

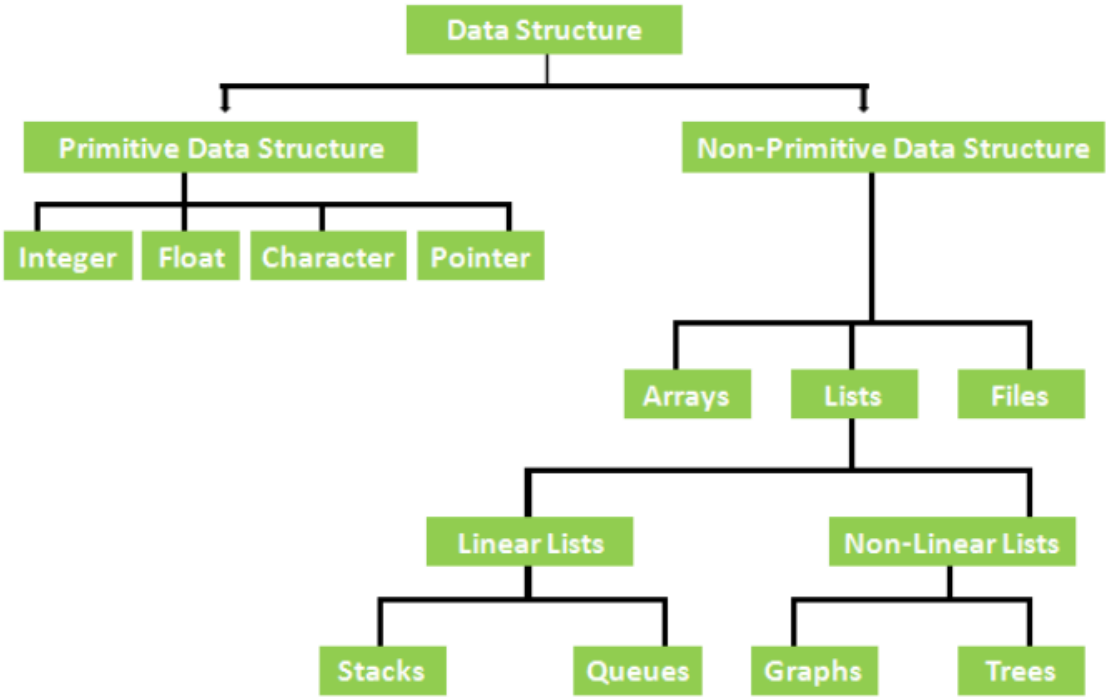
External Sorting
● External sorting is a term for a class of sorting algorithms that can handle massive amounts of data.
● External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually RAM) and instead, they must reside in the slower external memory (usually a hard drive).
● Examples:- Merge Sort

Finding Unsorted Subarray

```
import java.util.Arrays;

public class ShortestUnsortedSubarray {
    public static int findUnsortedSubarray(int[] nums) {
        int[] sorted = nums.clone();
        Arrays.sort(sorted);
        int start = nums.length, end = 0;

        for (int i = 0; i < nums.length; i++) {
            if (nums[i] != sorted[i]) {
                start = Math.min(start, i);
                end = Math.max(end, i);
            }
        }

        return (end - start >= 0) ? end - start + 1 : 0;
    }

    public static void main(String[] args) {
        int[] nums = {2, 6, 4, 8, 10, 9, 15};
        System.out.println("Length of shortest unsorted subarray: " + findUnsortedSubarray(nums));
    }
}
```

Q2) Define a data structure? Draw and explain the classification of data structures.

.A. ● Data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.
● It is a special format for organizing and storing data.
● Arrays, linked lists, trees, graphs, etc. are all data structures.



Q3) The Fibonacci numbers are the numbers in the following integer sequence. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89. . . . . Write a function that generates first N Fibonacci numbers

A.
```
public class Fibonacci {
    public static void main(String[] args) {
        int N = 10;
        generateFibonacci(N);
    }

    public static void generateFibonacci(int N) {
        int num1 = 0, num2 = 1;
```

```
        for (int i = 0; i < N; i++) {
            System.out.print(num1 + " ");
            int num3 = num1 + num2;
            num1 = num2;
            num2 = num3;
        }
    }
}
```

Q4) Explain the linear search procedure for the following list of elements and assume the key element is 96. List: 12, 23, 34, 45, 55, 62, 71, 85, 96

A.Procedure for Linear Search:

1. **start from the first element** of the list.
2. **Compare** the current element with the key element (96).
3. **If a match is found**, return the index of the element.
4. **If no match is found**, move to the next element.
5. **Repeat steps 2-4** until the end of the list is reached.
6. **If the key element is not found**, return -1.

Program

```
public class LinearSearch {
    public static int search(int[] arr, int key) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == key) {
                return i;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {12, 23, 34, 45, 55, 62, 71, 85, 96};
        int key = 96;
        int result = search(arr, key);
        if (result != -1) {
```

```
            System.out.println("Element found at index: " + result);
        } else {
            System.out.println("Element not found in the list.");
        }
    }
}
```

Q5) A Pancake Sorting Problem: Given an unsorted array, sort the given array. One can do only the following operation on an array. flip(arr, i): Reverse array from 0 to i Write an efficient program for sorting a given array in O(nLogn) time on the given machine

A. Program

```
import java.util.Arrays;

public class PancakeSort {
    // Function to flip the array from 0 to i
    static void flip(int[] arr, int i) {
        int start = 0;
        while (start < i) {
            int temp = arr[start];
            arr[start] = arr[i];
            arr[i] = temp;
            start++;
            i--;
        }
    }

    // Function to find the index of the maximum element in arr[0..n-1]
    static int findMax(int[] arr, int n) {
        int maxIdx = 0;
        for (int i = 1; i < n; i++) {
            if (arr[i] > arr[maxIdx]) {
                maxIdx = i;
            }
        }
```

```
        return maxIdx;
    }

    // Function to sort the array using pancake sort
    static void pancakeSort(int[] arr) {
        for (int currSize = arr.length; currSize > 1; currSize--) {
            int maxIdx = findMax(arr, currSize);
            if (maxIdx != currSize - 1) {
                flip(arr, maxIdx);
                flip(arr, currSize - 1);
            }
        }
    }

    public static void main(String[] args) {
        int[] arr = {23, 10, 20, 11, 12, 6, 7};
        pancakeSort(arr);
        System.out.println("Sorted Array: " + Arrays.toString(arr));
    }
}
```

Q6) Define sorting? Write the procedure for bubble sort using a suitable example?

A. Sorting is a process of ordering or placing a list of elements from a collection in some kind of order. It is nothing but storage of data in sorted order. Sorting can be done in ascending and descending order. It arranges the data in a sequence which makes searching easier.

Procedure for Bubble Sort

1. **Start** with an array of elements.
2. **Compare** each pair of adjacent elements from the beginning of the array to the end.
3. **Swap** the elements if they are in the wrong order.
4. **Repeat** the process for all elements except the last one.
5. **Continue** the process for the remaining elements, excluding the last sorted elements each time.
6. **Stop** when no more swaps are needed.

Program

```
public class BubbleSort {
    // Method to perform Bubble Sort on an array
    public static void bubbleSort(int[] array) {
        int n = array.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                // Compare adjacent elements and swap if needed
                if (array[j] > array[j + 1]) {
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
    }

    // Method to print the sorted array
    public static void printArray(int[] array) {
        int n = array.length;
        for (int i = 0; i < n; i++) {
            System.out.print(array[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int[] array = {64, 34, 25, 12, 22, 11, 90};
        System.out.println("Original Array:");
        printArray(array);

        // Apply Bubble Sort
        bubbleSort(array);

        System.out.println("Sorted Array:");
        printArray(array);
```

```
  }
}
```

Q7) Explain Binary Search procedure for the following list of elements and assume the key element is 85. 12, 23, 34, 45, 55, 62, 71, 85, 96

A. . Procedure of Binary search

Binary search is implemented using the following steps...

Step 1 - Read the search element from the user.

Step 2 - Find the middle element in the sorted list.

Step 3 - Compare the search element with the middle element in the sorted list.

Step 4 - If both are matched, then display "Given element is found!!!" and terminate the function.

Step 5 - If both are not matched, then check whether the search element is smaller or larger than the middle element.

Step 6 - If the search element is smaller than the middle element, repeat steps 2, 3, 4 and 5 for the left sublist of the middle element.

Step 7 - If the search element is larger than the middle element, repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.

Step 8 - Repeat the same process until we find the search element in the list or until the sublist contains only one element.

Step 9 - If that element also doesn't match with the search element, then display "Element is not found in the list!!!" and terminate the function.

Q8) Explain the following sorting algorithms with an example and write their time complexities?

• Radix Sort
• Shell Sort

A. Radix Sort is a non-comparative sorting algorithm that sorts numbers by processing individual digits. It works by sorting the numbers digit by digit, starting from the least significant digit (LSD) to the most significant digit (MSD).

Example

Consider the array: `[170, 45, 75, 90, 802, 24, 2, 66]`.

1. Sort by the least significant digit (units place):

    - `[170, 90, 802, 2, 24, 45, 75, 66]`
2. Sort by the tens place:
    - `[802, 2, 24, 45, 66, 170, 75, 90]`
3. Sort by the hundreds place:
    - `[2, 24, 45, 66, 75, 90, 170, 802]`

The array is now sorted.

Radix Sort example

```
int n = arr.length;
    int[] output = new int[n];
    int[] count = new int[10];
    Arrays.fill(count, 0);

    for (int i = 0; i < n; i++) {
        count[(arr[i] / exp) % 10]++;
    }

    for (int i = 1; i < 10; i++) {
        count[i] += count[i - 1];
    }

    for (int i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }

    System.arraycopy(output, 0, arr, 0, n);
```

me Complexity

- **Best Case:**
  O(nk)

- **Average Case:**
  O(nk)

- **Worst Case:**
  O(nk)

- **Space Complexity:**
  O(n+k)

Here, (n) is the number of elements and (k) is the number of digits in the largest number.

Shell Sort

Shell Sort is an optimization over Insertion Sort. It allows the exchange of far-apart elements to reduce the total number of movements required.

Example

Consider the array: `[12, 34, 54, 2, 3]`.

1. Initial gap (n/2): 2
   - Compare and swap elements at intervals of 2: `[12, 3, 54, 2, 34]`
2. Next gap (gap/2): 1
   - Perform insertion sort: `[2, 3, 12, 34, 54]`

The array is now sorted.

```
nt n = arr.length;
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
}

public static void main(String[] args) {
    int[] arr = {12, 34, 54, 2, 3};
    shellSort(arr);
```

```
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

Time Complexity

- **Best Case:**
  O(nlogn)
- **Average Case:**
  O(n1.25)
- **Worst Case:**
  O(n2)
- **Space Complexity:**
  O(1)

Q9) Explain Binary Search procedure for the following list of elements and assume the key element is 49. 12, 23, 34, 45, 55, 62, 71, 85, 96

A. **Binary Search Procedure**

1. Initial Setup:
   - Start with two pointers: `low` at the beginning (index 0) and `high` at the end (index 8) of the list.
2. First Iteration:
   - Calculate the middle index: `mid = (low + high) / 2 = (0 + 8) / 2 = 4`.
   - Compare the middle element (55) with the key (49):
     - Since 49 < 55, update `high` to `mid - 1 = 3`.
3. Second Iteration:
   - Calculate the new middle index: `mid = (low + high) / 2 = (0 + 3) / 2 = 1`.
   - Compare the middle element (23) with the key (49):
     - Since 49 > 23, update `low` to `mid + 1 = 2`.
4. Third Iteration:

- Calculate the new middle index: `mid = (low + high) / 2 = (2 + 3) / 2 = 2`.
- Compare the middle element (34) with the key (49):
  - Since 49 > 34, update `low` to `mid + 1 = 3`

Q10) Sort the given list of elements using insertion sort.14, 33,27,10,35,19,42,44.

A.
```
public class InsertionSort {
  void sort(int arr[]) {
    int n = arr.length;
    for (int i = 1; i < n; ++i) {
      int key = arr[i];
      int j = i - 1;

      while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j];
        j = j - 1;
      }
      arr[j + 1] = key;
    }
  }

  static void printArray(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n; ++i)
      System.out.print(arr[i] + " ");
    System.out.println();
  }

  public static void main(String args[]) {
    int arr[] = {14, 33, 27, 10, 35, 19, 42, 44};
```

```
        InsertionSort ob = new InsertionSort();
        ob.sort(arr);
        printArray(arr);
    }
}
```

Q11) Write the name of the sorting technique which is used in playing cards game? Write a procedure for sorting a given list of numbers using that technique? 14, 25, 36, 74, 85, 6, 53, 62, 41

A. **Insertion Sort Procedure**

1. **Start with the second element** (index 1) of the array.
2. **Compare it with the elements before it** (index 0 to current index - 1).
3. **Shift elements** that are greater than the current element to the right.
4. **Insert the current element** into its correct position.
5. **Repeat** the process for all elements in the array.

Program

```
public class InsertionSort {
    public static void main(String[] args) {
        int[] numbers = {14, 25, 36, 74, 85, 6, 53, 62, 41};
        insertionSort(numbers);
        for (int number : numbers) {
            System.out.print(number + " ");
        }
    }

    public static void insertionSort(int[] array) {
        for (int i = 1; i < array.length; i++) {
            int key = array[i];
            int j = i - 1
            while (j >= 0 && array[j] > key) {
```

```
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = key;
    }
  }
}
```

Q12) Write the algorithm for bubble sort and then explain with an example

A.

Bubble Sort Algorithm

1. **Start** with an array of `n` elements.
2. **Repeat** the following steps for `i` from `0` to `n-1`:
    - For `j` from `0` to `n-i-1`:
        - If `arr[j]` is greater than `arr[j+1]`, **swap** them.
3. **End** when no more swaps are needed.

```
public class BubbleSort {
 void bubbleSort(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n-1; i++) {
       for (int j = 0; j < n-i-1; j++) {
          if (arr[j] > arr[j+1]) {
             // Swap arr[j] and arr[j+1]
             int temp = arr[j];
             arr[j] = arr[j+1];
             arr[j+1] = temp;
          }
```

```
        }
      }
   }
void printArray(int arr[]) {
      int n = arr.length;
      for (int i = 0; i < n; ++i)
         System.out.print(arr[i] + " ");
      System.out.println();
   }
```

Q13) Explain the procedure, advantages and disadvantages of linear and binary search with a suitable example?

A. . Procedure of Linear Search

- **Begins search at first item in list, continues searching sequentially(item by item) through list, until desired item(key) is found, or until end of list is reached.**

  Also called sequential or serial search.
- **Obviously not an efficient method for searching ordered lists like phone directory(which is ordered alphabetically).**
- **Advantages**

  1. **Algorithm is simple.**
  2. **List need not be ordered in any particular way.**

The main disadvantage of a linear search is the fact that it's time consuming for the enormous arrays.

```
public class LinearSearch {
    public static int linearSearch(int[] arr, int key) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == key) {
                return i;
```

```
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {3, 4, 1, 7, 5};
        int key = 4;
        int result = linearSearch(arr, key);
        if (result == -1) {
            System.out.println("Element not found in the array");
        } else {
            System.out.println("Element found at index: " + result);
        }
    }
}
```

Procedure of Binary search

Binary search is implemented using the following steps...

Step 1 - Read the search element from the user.

Step 2 - Find the middle element in the sorted list.

Step 3 - Compare the search element with the middle element in the sorted list.

Step 4 - If both are matched, then display "Given element is found!!!" and terminate the function.

Step 5 - If both are not matched, then check whether the search element is smaller or larger than the middle element.

Step 6 - If the search element is smaller than the middle element, repeat steps 2, 3, 4 and 5 for the left sublist of the middle element.

Step 7 - If the search element is larger than the middle element, repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.

Step 8 - Repeat the same process until we find the search element in the list or until the sublist contains only one element.
Step 9 - If that element also doesn't match with the search element, then display "Element is not found in the list!!!" and terminate the function.

```
public class BinarySearch {
    public static int binarySearch(int[] arr, int target) {
        int left = 0;
        int right = arr.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (arr[mid] == target) {
                return mid;
            }

            if (arr[mid] < target) {
                left = mid + 1;
            }
            else {
                right = mid - 1;
            }
        }

        return -1;
    }

    public static void main(String[] args) {
        int[] sortedArray = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
```

```
        int target = 23;
        int result = binarySearch(sortedArray, target);

        if (result != -1) {
            System.out.println("Element found at index " + result);
        } else {
            System.out.println("Element not found");
        }
    }
}
```

Advantages and Disadvantages of Binary Search:

- Advantage:

1. Binary search is an optimal searching algorithm using which we can search desired element very efficiently.

- Disadvantage:

1. This algorithm requires the list to be sorted . Then only this method is applicable.

Q14) Compare the time complexities of various searching and sorting algorithms?

A. TABLE BELOW

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)

Q15) Write an algorithm to search for an employee ID in an array

A. 1. Read the employee ID to be searched
2. Set i = 0
3. Repeat step 4 until i > n or arr[i] = employee ID
4. Increment i by 1
5. If i > n:
Display “Not Found”
Else
Display “Found

Q16) Explain bubble sort by sorting the following list of elements: 5 ,1, 4, 2, 8.

A. Bubble Sort Algorithm is used to arrange N elements in ascending order, and for that, you have to begin with the 0th element and compare it with the first element. If the 0th element is found greater than the 1st element, then the swapping operation will be performed, i.e., the two values will get interchanged. In this way, all the elements of the array get compared.

```
public class BubbleSort {
    public static void main(String[] args) {
        int[] arr = {5, 1, 4, 2, 8};
        bubbleSort(arr);
        System.out.println("Sorted array:");
        printArray(arr);
    }
```

Q17) What is the idea behind Selection sort and sort the following list of elements using that idea. array A = [7 , 5 , 4 , 2] needs to be sorted in ascending order.

A. The Algorithm is mentioned below:

1. Get a list of unsorted numbers
2. Set a marker for the unsorted section at the front of the list
3. Repeat steps 4 - 6 until one number remains in the unsorted section
4. Compare all unsorted numbers in order to select the smallest one
5. Swap this number with the first number in the unsorted section
6. Advance the marker to the right one position
7. Stop

```
ublic class SelectionSort {
    public static void main(String[] args) {
        int[] A = {7, 5, 4, 2};
        selectionSort(A);
        System.out.println("Sorted array: ");
        for (int i : A) {
            System.out.print(i + " ");
        }
    }
```

Q18) . Sort the given list of elements using selection sort. 14, 33,27,10,35,19,42,4

A.

```
    public static void main(String[] args) {
        SelectionSort sorter = new SelectionSort();
        int[] arr = {14, 33, 27, 10, 35, 19, 42, 4};
        System.out.println("Before sorting:");
        sorter.printArray(arr);
        sorter.sort(arr);
        System.out.println("After sorting:");
        sorter.printArray(arr);
    }
}
```

Q19) . Define selection sort and write pseudo code for selection sort

A. . Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving the unsorted array boundary by one element to the right.

```
class SelectionSort {
    void sort(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            // Find the minimum element in the unsorted part
            int min_idx = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[min_idx]) {
                    min_idx = j;
                }
            }
            // Swap the found minimum element with the first element
            int temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }
```

Q20) Explain insertion sort with an example and compare time complexity of insertion sort with other sorting algorithms.
A.

Insertion Sort

Insertion Sort is a simple sorting algorithm that builds the final sorted array one item at a time. It is much like sorting playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed in the correct position in the sorted part.

Steps:

1. Start with the second element (index 1) as the first element is considered sorted.
2. Compare the current element with the elements in the sorted part.
3. Shift all elements in the sorted part that are greater than the current element to the right.
4. Insert the current element into its correct position.
5. Repeat until the entire array is sorted.

```
public class InsertionSort {
    void sort(int arr[]) {
        int n = arr.length;
        for (int i = 1; i < n; ++i) {
            int key = arr[i];
```

```
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
  }
  static void printArray(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
  }
  public static void main(String args[]) {
    int arr[] = {12, 11, 13, 5, 6};
    InsertionSort ob = new InsertionSort();
    ob.sort(arr);
    printArray(arr);
  }
}
```