Algorithms

Selection Sort

Steps:

1. **Find the Minimum**: Start with the first element, search the entire array to find the smallest element, and swap it with the first element.
2. **Move to Next Position**: Then move to the second element, find the smallest element from the remaining array, and swap it with the second element.
3. **Repeat**: Continue this process until the array is completely sorted.

Code:

```
import java.util.Scanner;

public class SelectionSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Ask the user for the number of elements in the array
        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        // Create an array to hold the elements
        int[] arr = new int[n];

        // Ask the user to enter the elements
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        // Selection sort algorithm
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
```

```
        // Print the sorted array
        System.out.println("Sorted array:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

How It Works:

1. **Initial Array**: {64, 25, 12, 22, 11}
2. **Step 1**: Find the smallest element (11) and swap it with the first element (64). New array: {11, 25, 12, 22, 64}
3. **Step 2**: Find the smallest element in the rest of the array (12) and swap it with the second element (25). New array: {11, 12, 25, 22, 64}
4. **Step 3**: Find the smallest element in the rest of the array (22) and swap it with the third element (25). New array: {11, 12, 22, 25, 64}
5. **Step 4**: The array is now sorted.

Bubble Sort

Steps:

1. **Compare Adjacent Elements**:
- Start from the beginning of the array.
- Compare each pair of adjacent elements and swap them if the first element is greater than the second.
2. **Move Through the Array**:
- Continue comparing and swapping adjacent elements until you reach the end of the array.
- This process will "bubble up" the largest element to its correct position at the end of the array.
3. **Repeat the Process**:
- Move to the next pass and repeat the process for the remaining unsorted portion of the array.
- In each pass, the next largest element is bubbled up to its correct position.
4. **Continue Until Sorted**:
- Continue the passes until no more swaps are needed, indicating that the array is sorted.

Code:

```
import java.util.Scanner;
```

```
public class BubbleSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Ask the user for the number of elements in the array
        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        // Create an array to hold the elements
        int[] arr = new int[n];

        // Ask the user to enter the elements
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        // Bubble sort algorithm
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - 1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }

        // Print the sorted array
        System.out.println("Sorted array:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

How It Works:

Initial Array:

{64, 25, 12, 22, 11}

Pass 1:

1. Compare 64 and 25, swap. Array: {25, 64, 12, 22, 11}
2. Compare 64 and 12, swap. Array: {25, 12, 64, 22, 11}
3. Compare 64 and 22, swap. Array: {25, 12, 22, 64, 11}
4. Compare 64 and 11, swap. Array: {25, 12, 22, 11, 64}

Pass 2:

1. Compare 25 and 12, swap. Array: {12, 25, 22, 11, 64}
2. Compare 25 and 22, swap. Array: {12, 22, 25, 11, 64}
3. Compare 25 and 11, swap. Array: {12, 22, 11, 25, 64}

Pass 3:

1. Compare 12 and 22, no swap.
2. Compare 22 and 11, swap. Array: {12, 11, 22, 25, 64}

Pass 4:

1. Compare 12 and 11, swap. Array: {11, 12, 22, 25, 64}

Now, the array is sorted!

Linear Search

Steps:

1. **Start from the Beginning**: Begin at the first element of the array.
2. **Compare Each Element**: Compare each element with the target value.
3. **Find the Target**: If the element matches the target value, return its index.
4. **Continue Until End**: If the target is not found, continue the comparison until the end of the array.
5. **Return Result**: If the target is found, return its index; otherwise, return -1.

Code:

```
import java.util.Scanner;


public class LinearSearch {
```

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // Ask the user for the number of elements in the array
    System.out.print("Enter the number of elements: ");
    int n = scanner.nextInt();

    // Create an array to hold the elements
    int[] arr = new int[n];

    // Ask the user to enter the elements
    System.out.println("Enter the elements of the array:");
    for (int i = 0; i < n; i++) {
        arr[i] = scanner.nextInt();
    }

    // Ask the user to enter the target element
    System.out.print("Enter the target element: ");
    int target = scanner.nextInt();

    // Linear search algorithm
    int result = -1;
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
```

```
            result = i;
            break;
        }
    }

    // Print the result
    if (result == -1) {
        System.out.println("Element not found in the array.");
    } else {
        System.out.println("Element found at index: " + result);
    }
  }
}
```

How It Works:

- **Initial Array**: {64, 25, 12, 22, 11}, **Target**: 22
- **Step 1**: Compare 64 with 22 - not a match
- **Step 2**: Compare 25 with 22 - not a match
- **Step 3**: Compare 12 with 22 - not a match
- **Step 4**: Compare 22 with 22 - match found
- **Result**: Target 22 is found at index 3

Binary Search

Steps:

1. **Sort the Array**: Ensure the array is sorted.
2. **Initialize Pointers**: Set two pointers, `low` at the start and `high` at the end of the array.
3. **Calculate Midpoint**: Find the middle element using the formula `mid = (low + high) / 2`.
4. **Compare Target with Midpoint**:
    - If the target is equal to the mid element, return the mid index.

- If the target is less than the mid element, move the `high` pointer to `mid - 1`.
- If the target is greater than the mid element, move the `low` pointer to `mid + 1`.

5. **Repeat**: Continue the process until `low` exceeds `high`.

Code:

```
import java.util.Scanner;

import java.util.Arrays;


public class BinarySearch {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);


        // Ask the user for the number of elements in the array

        System.out.print("Enter the number of elements: ");

        int n = scanner.nextInt();


        // Create an array to hold the elements

        int[] arr = new int[n];


        // Ask the user to enter the elements

        System.out.println("Enter the elements of the array:");

        for (int i = 0; i < n; i++) {

            arr[i] = scanner.nextInt();

        }


        // Sort the array
```

```
Arrays.sort(arr);

// Ask the user to enter the target element
System.out.print("Enter the target element: ");
int target = scanner.nextInt();

// Binary search algorithm
int low = 0;
int high = arr.length - 1;
int result = -1;

while (low <= high) {
    int mid = (low + high) / 2;
    if (arr[mid] == target) {
        result = mid;
        break;
    } else if (arr[mid] < target) {
        low = mid + 1;
    } else {
        high = mid - 1;
    }
}

// Print the result
```

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

How It Works:

- **Initial Array** (sorted): {11, 12, 22, 25, 64}, **Target**: 22
- **Step 1**: Calculate the midpoint: `mid = (0 + 4) / 2 = 2`. Compare 22 with 22 - match found.
- **Result**: Target 22 is found at index 2.