Data Structure and Algorithms

Lecture 4 & 5 : Sort & Search Algorithms on Array

Things to cover

- Linear Search
- Bubble Sort
- Binary Search
- Section Sort
- Insert Sort

Linear Search

```
#include <stdio.h>

// Function to perform linear search
int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i; // Return the index if the
target is found
        }
    }
    return -1; // Return -1 if the target is not
found
}
```

Bubble Sort

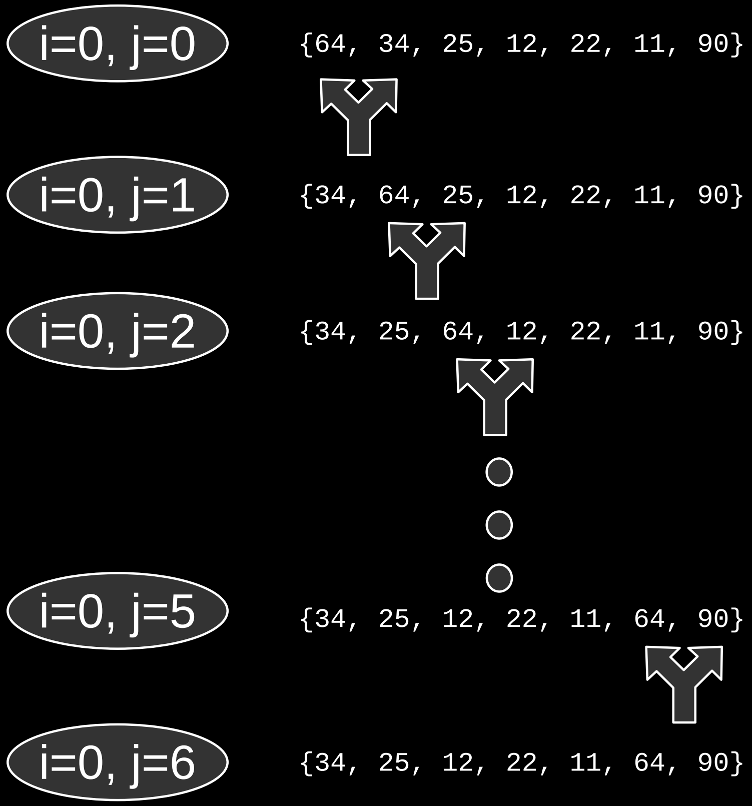
```
#include <stdio.h>

// Function to swap two elements
void swap(int *xp, int *yp) {
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
```

```
// Function to perform Bubble Sort on an array
void bubbleSort(int arr[], int n) {
    int i, j;
    for (i = 0; i < n - 1; i++) {
        // Last i elements are already in place
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j+1]) {
                swap(&arr[j], &arr[j+1]);
            }
        }
    }
}
```

Bubble Sort

```
int arr[] = {64, 34, 25, 12, 22, 11, 90};
```



i=0, j=0 {64, 34, 25, 12, 22, 11, 90}

i=0, j=1 {34, 64, 25, 12, 22, 11, 90}

i=0, j=2 {34, 25, 64, 12, 22, 11, 90}

i=0, j=5 {34, 25, 12, 22, 11, 64, 90}

i=0, j=6 {34, 25, 12, 22, 11, 64, 90}

Binary Search

```
#include <stdio.h>

    int binarySearch(int arr[], int low, int high, int key) {
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == key) {
                return mid; // Return the index if found
            } else if (arr[mid] < key) {
                low = mid + 1; // Search in the right half
                return binarySearch(arr, low, high, key);
            } else {
                high = mid - 1; // Search in the left half
                return binarySearch(arr, low, high, key);
            }
        }
        return -1; // Return -1 if not found
    }

    int main() {
        int arr[] = {10, 20, 30, 40, 50}; // Array must be sorted
        int n = sizeof(arr) / sizeof(arr[0]);
        int key = 40;
        int result = binarySearch(arr, 0, n - 1, key);

        if (result != -1) {
            printf("Element found at index: %d\n", result);
        } else {
            printf("Element not found\n");
        }
        return 0;
    }
```

$\rightarrow$ =0 + (4 – 0) / 2 = 2

$\rightarrow$ TRUE

$\rightarrow$ (arr, 0, 4, 40)

Binary Search

```
#include <stdio.h>

    int binarySearch(int arr[], int low, int high, int key) {
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == key) {
                return mid; // Return the index if found
            } else if (arr[mid] < key) {
                low = mid + 1; // Search in the right half
                return binarySearch(arr, low, high, key);
            } else {
                high = mid - 1; // Search in the left half
                return binarySearch(arr, low, high, key);
            }
        }
        return -1; // Return -1 if not found
    }
```

$\longrightarrow$ =3 + (4 – 3) / 2 = 3

low = 3, high = 4

Computational complexity

- Linear Search: Best O(1), Worst O(N)
- Bubble Sort: Best O(N), Worst O(N^2)
- Binary Search: Best O(1), Worst O(log_2(N))

Selection Sort

```
void selectionSort(int arr[], int n) {
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n - 1; i++) {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }

        // Swap the found minimum element with the first element
        // of the unsorted subarray
        if (min_idx != i) {
            int temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }
    }
}
```

Insertion Sort

```
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i]; // Store the current element as key
        j = i - 1;    // Start comparing with the element before key

        // Move elements of arr[0..i-1], that are greater than key,
        // to one position ahead of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j]; // Shift element to the right
            j = j - 1;
        }
        arr[j + 1] = key; // Place key in its correct position
    }
}
```