Insertion Sort in C

INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY
HYDERABAD

What is Insertion Sort?

- A simple sorting algorithm
- Works like sorting playing cards in your hand
- Builds the final sorted array **one item at a time**

Algorithm Idea

1. Start from the **second element**
2. Compare with elements before it
3. Shift larger elements to the right
4. Insert the element into the correct position
5. Repeat until array is sorted

C Implementation

```c
#include <stdio.h>

void insertionSort(int arr[], int n) {
    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--;
        }
        arr[j + 1] = key;
    }
}
```

What is Binary Search?

- Efficient algorithm for finding an element in a **sorted array**
- Works by repeatedly dividing the search interval in half
- Compare target with the middle element:
    - If equal → found
    - If smaller → search left half
    - If larger → search right half

Algorithm Idea

1. Start with `low = 0`, `high = n-1`
2. Compute `mid = (low + high) / 2`
3. Compare `arr[mid]` with `key`
    - If `arr[mid] == key` → return `mid`
    - If `arr[mid] > key` → search left half
    - Else → search right half
4. Repeat until `low > high`

Recursive Implementation (C)

```c
int binarySearchRec(int arr[], int low, int high, int key) {
    if (low > high) return -1;

    int mid = (low + high) / 2;

    if (arr[mid] == key)
        return mid;
    else if (arr[mid] > key)
        return binarySearchRec(arr, low, mid - 1, key);
    else
        return binarySearchRec(arr, mid + 1, high, key);
}
```

Example Dry Run

Array: {1, 3, 5, 7, 9, 11, 13}

Search for 7

low=0, high=6 → mid=3 → arr[3]=7 → found at index 3

Search for 6

low=0, high=6 → mid=3 → arr[3]=7 (too big)

low=0, high=2 → mid=1 → arr[1]=3 (too small)

low=2, high=2 → mid=2 → arr[2]=5 (too small)

low=3, high=2 → stop → not found

What is the Fibonacci Sequence?

- Each number is the sum of the two preceding numbers
- Defined as:
  - $F(0) = 0$
  - $F(1) = 1$
  - $F(n) = F(n-1) + F(n-2)$, for $n \geq 2$

Example sequence:

0, 1, 1, 2, 3, 5, 8, 13, ...

Recursive Definition in C

```c
int fibonacci(int n) {
    if (n == 0) return 0;  // base case
    if (n == 1) return 1;  // base case
    return fibonacci(n-1) + fibonacci(n-2); // recursion
}
```

Example Trace: F(5)

```
fibonacci(5)
= fibonacci(4) + fibonacci(3)
= (fibonacci(3) + fibonacci(2)) + (fibonacci(2) + fibonacci(1))
= ...
= 5
```

Tree of calls (partial):

```
          F(5)
         /    \
     F(4)      F(3)
    /   \     /   \
F(3)  F(2) F(2)  F(1)
...
```

Towers of Hanoi

- You have **3 pegs**: Source (A), Auxiliary (B), Destination (C)
- **n disks** of different sizes are stacked on Source peg
- Goal: Move all disks to Destination peg
- Rules:
    i. Only one disk moved at a time
    ii. A disk can only be placed on an empty peg or on a larger disk
    iii. Disks must maintain order

Recursive Idea

To move `n` disks from A → C using B:

1. Move `n−1` disks from A → B (using C)
2. Move the largest disk from A → C
3. Move `n−1` disks from B → C (using A)

Example for 3 Disks

1. Move 2 disks from A → B
2. Move disk 3 from A → C
3. Move 2 disks from B → C

Moves:

A → C

A → B

C → B

A → C

B → A

B → C

C Implementation

```c
#include <stdio.h>

void hanoi(int n, char from, char to, char aux) {
    if (n == 1) {
        printf("Move disk 1 from %c to %c\n", from, to);
        return;
    }
    hanoi(n - 1, from, aux, to);
    printf("Move disk %d from %c to %c\n", n, from, to);
    hanoi(n - 1, aux, to, from);
}

int main() {
    int n = 3;
    hanoi(n, 'A', 'C', 'B');
    return 0;
}
```

What is Merge Sort?

- A **Divide and Conquer** sorting algorithm
- Steps:
  i. Divide array into two halves
  ii. Recursively sort both halves
  iii. Merge the two sorted halves
- **Always O(n log n)** time complexity

Algorithm Outline

1. If array has 0 or 1 elements → already sorted
2. Otherwise:
    - Divide array into left and right halves
    - Recursively sort each half
    - Merge two sorted halves into one sorted array

Merge Function

```c
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++) L[i] = arr[l + i];
    for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}
```

Merge Sort Function

```
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;

        // Recursively sort left and right halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        // Merge sorted halves
        merge(arr, l, m, r);
    }
}
```

Example Dry Run

Array = {12, 11, 13, 5, 6, 7}

Divide → {12, 11, 13} and {5, 6, 7}

Recursively sort:

{12, 11, 13} → {11, 12, 13}

{5, 6, 7} → {5, 6, 7}

Merge:

{11, 12, 13} and {5, 6, 7} → {5, 6, 7, 11, 12, 13}

Visualization of Recursive Calls

```
mergeSort([12,11,13,5,6,7])
 → mergeSort([12,11,13])
     → mergeSort([12,11])
        → mergeSort([12]) & mergeSort([11])
        → merge([12], [11]) → [11,12]
     → mergeSort([13]) → [13]
     → merge([11,12], [13]) → [11,12,13]
 → mergeSort([5,6,7]) → [5,6,7]
 → merge([11,12,13], [5,6,7]) → [5,6,7,11,12,13]
```