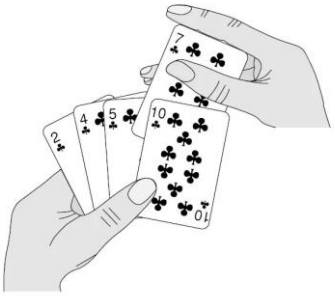
CS 311 Sorting Algorithm Review

Insertion Sort



Overview:

Insertion Sort builds the final sorted array one element at a time. It works like how you might sort playing cards in your hand:

1. Start with the second element (index 1).
2. Compare it to the elements before it.
3. Shift larger elements from one position to the right.
4. Insert the current element into its correct place.
5. Repeat for all elements.

Code:

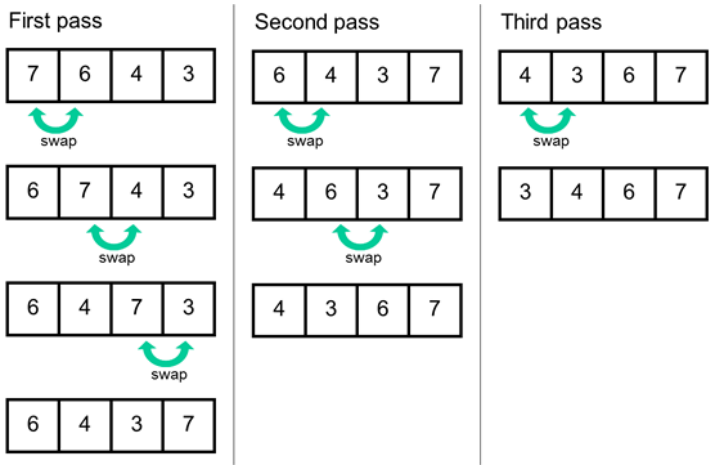
```cpp
#include <iostream>
using namespace std;

void insertionSort(int arr[], int size) {
    for (int i = 1; i < size; i++) {
        int key = arr[i];       // Element to insert
        int j = i - 1;

        // Shift elements greater than key to the right
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        // Insert key at correct position
        arr[j + 1] = key;
    }
}
```

Bubble Sort



Overview:

Bubble Sort repeatedly steps through the array, compares adjacent pairs, and swaps them if they are in the wrong order. After each full pass, the largest remaining unsorted element "bubbles up" to its correct position at the end. This process repeats until no more swaps are needed.

Code:

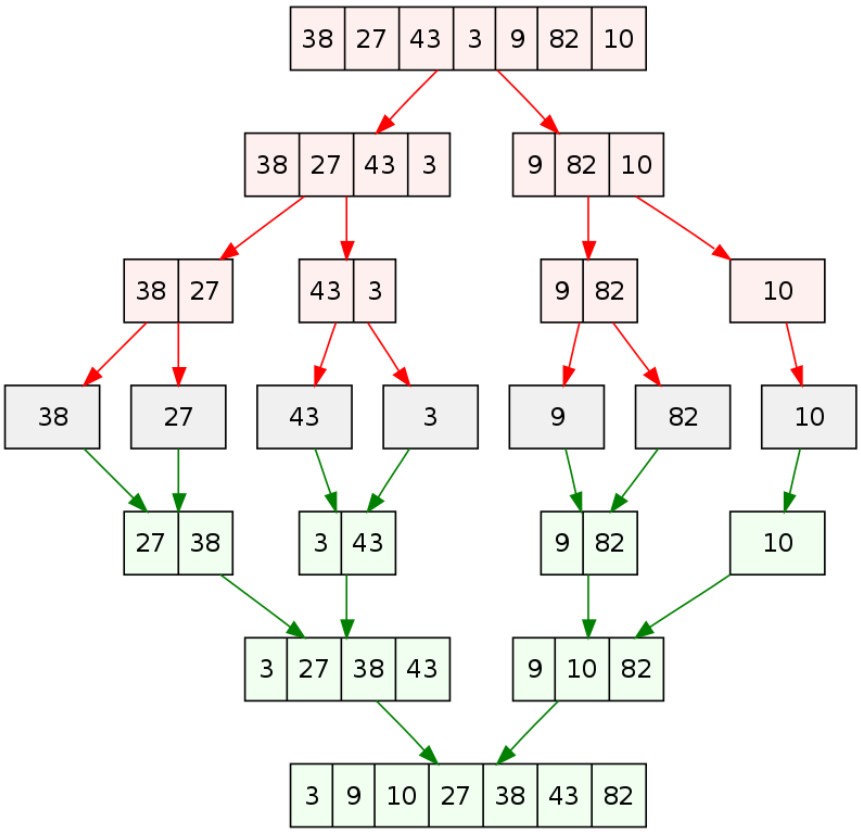
```cpp
#include <iostream>
using namespace std;

void bubbleSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        // Flag to check if any swap happened
        bool swapped = false;

        for (int j = 0; j < size - i - 1; j++) {
            // Swap if elements are in wrong order
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }

        // Optimization: Stop if no swaps occurred (already sorted)
        if (!swapped)
            break;
    }
}
```

Merge Sort



Overview:

Merge Sort works by repeatedly splitting the array into halves until each piece has one element, then merging those pieces back together in sorted order.

Imagine you're sorting a stack of papers:

1. You divide the pile into two halves.
2. Then you split each half again and again until you have single sheets.
3. Next, you merge small, sorted stacks into larger sorted stacks—like putting two piles in order by comparing the top papers and picking the smaller one each time.
4. Eventually, all the sorted piles come back together into one fully sorted pile.

It’s like a team sorting big piles faster by breaking up the work and combining it neatly afterward.

Code:

```cpp
#include <iostream>
using namespace std;

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Temporary arrays
    int L[n1], R[n2];

    // Copy data to temp arrays
    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    // Merge the temp arrays back
    int i = 0, j = 0, k = left;

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

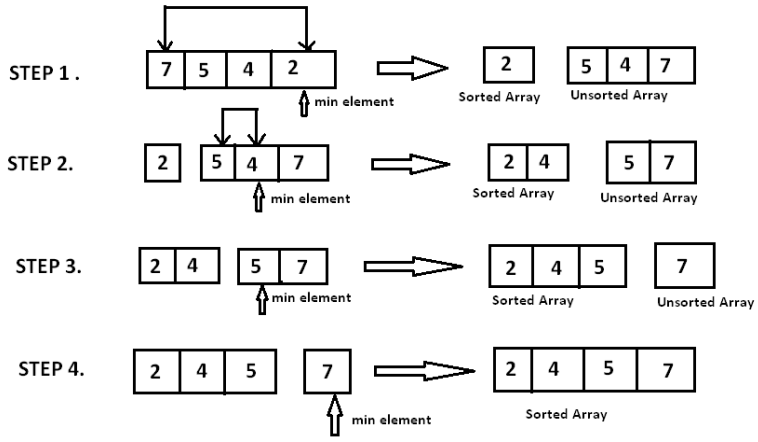
    // Copy remaining elements
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;

        mergeSort(arr, left, mid);        // Sort left half
        mergeSort(arr, mid + 1, right);   // Sort right half
        merge(arr, left, mid, right);     // Merge them
    }
}

```

Selection Sort



Overview:

Selection Sort repeatedly finds the smallest element from the unsorted part of the array and swaps it with the first unsorted element. It gradually grows a sorted portion at the beginning of the array until the whole array is sorted.

Think of selecting the shortest student from a line to form a new line ordered by height. You pick the shortest remaining student, put them at the front, and repeat with the rest until everyone is lined up.

Code:

```cpp
#include <iostream>
using namespace std;

void selectionSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int minIndex = i;

        // Find the minimum element in the unsorted part
        for (int j = i + 1; j < size; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }

        // Swap the found minimum element with the first element of unsorted part
        if (minIndex != i) {
            swap(arr[i], arr[minIndex]);
        }
    }
}
```