Sorting Algorithms

Agenda

1. What are Algorithms?
2. Sorting algorithms
    a. Bubble sort
    b. Insertion sort
    c. Merge sort
3. Bubble Vs Insertion Vs Merge
4. To solve

What are Algorithms?

Algorithms are like step-by-step instructions that computers follow to solve different problems. They are the building blocks of computer programming and can be used to solve simple tasks like sorting or searching, as well as more complex ones like making an artificial intelligence program or machine learning system.

Sorting algorithms:

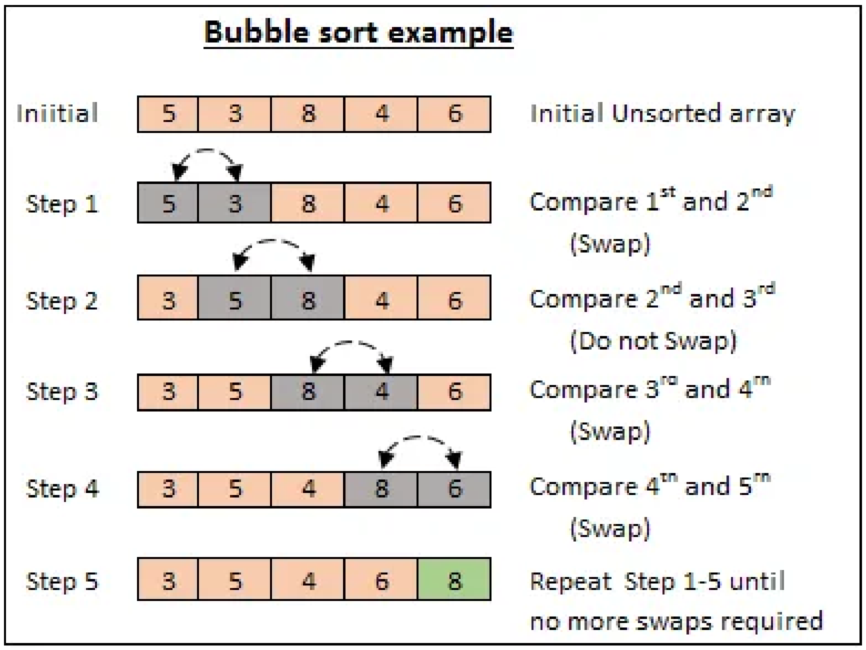
Sorting algorithms are step-by-step procedures that arrange a collection of items in a specific order

Some sorting algorithms:

1. **Bubble sort:**

   One of the simplest sorting algorithms. It works by exchanging adjacent elements if they are in the wrong order, and this process is repeated until the entire list is sorted.

   How it works:


Bubble sort example
Iniitial
5 3 8 4 6
Initial Unsorted array
Step 1
5 3 8 4 6
Compare 1st and 2nd (Swap)
Step 2
3 5 8 4 6
Compare 2nd and 3rd (Do not Swap)
Step 3
3 5 8 4 6
Compare 3rd and 4th (Swap)
Step 4
3 5 4 8 6
Compare 4th and 5th (Swap)
Step 5
3 5 4 6 8
Repeat Step 1-5 until no more swaps required

Code:

```
void bubbleSort(int array[], int sz) {
  for (int step = 0; step < sz; ++step) {
    for (int i = 0; i < sz-1; ++i) {
       // SORT ASCENDING
      if (array[i] > array[i + 1])
        swap(array[i], arr[i+1);
    }
  }
}
```

Advantages:

Bubble sort is a straightforward way to sort a list of items. It is easy to understand and use, and it doesn't need any extra memory to work.

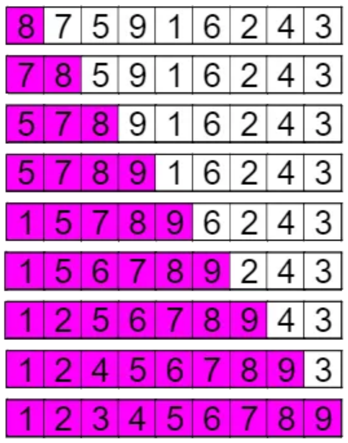
Disadvantages:

Bubble sort's time complexity is $O(n^2)$, which means that it can be very inefficient for sorting large inputs due to the significant amount of time it takes to execute.

2. Insertion sort:

A basic sorting algorithm that works by inserting each element into its correct position in the already-sorted section of the list. It compares each element to the ones before it and inserts it into the proper place.

How it works:



Code:

```
void insertionSort (int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++){
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Advantages:

Insertion sort is a simple sorting algorithm that is easy to learn and apply. Moreover, it does not require any additional memory space to carry out the sorting process.

Disadvantages:

Due to its time complexity of $O(n^2)$, Insertion sort can be slow for large inputs, which means that it may not be the best choice for sorting large amounts of data.

3. Merge sort:

Merge sort is a sorting technique that involves breaking an array into smaller subarrays, sorting each subarray separately, and then merging them together to create the final sorted array.

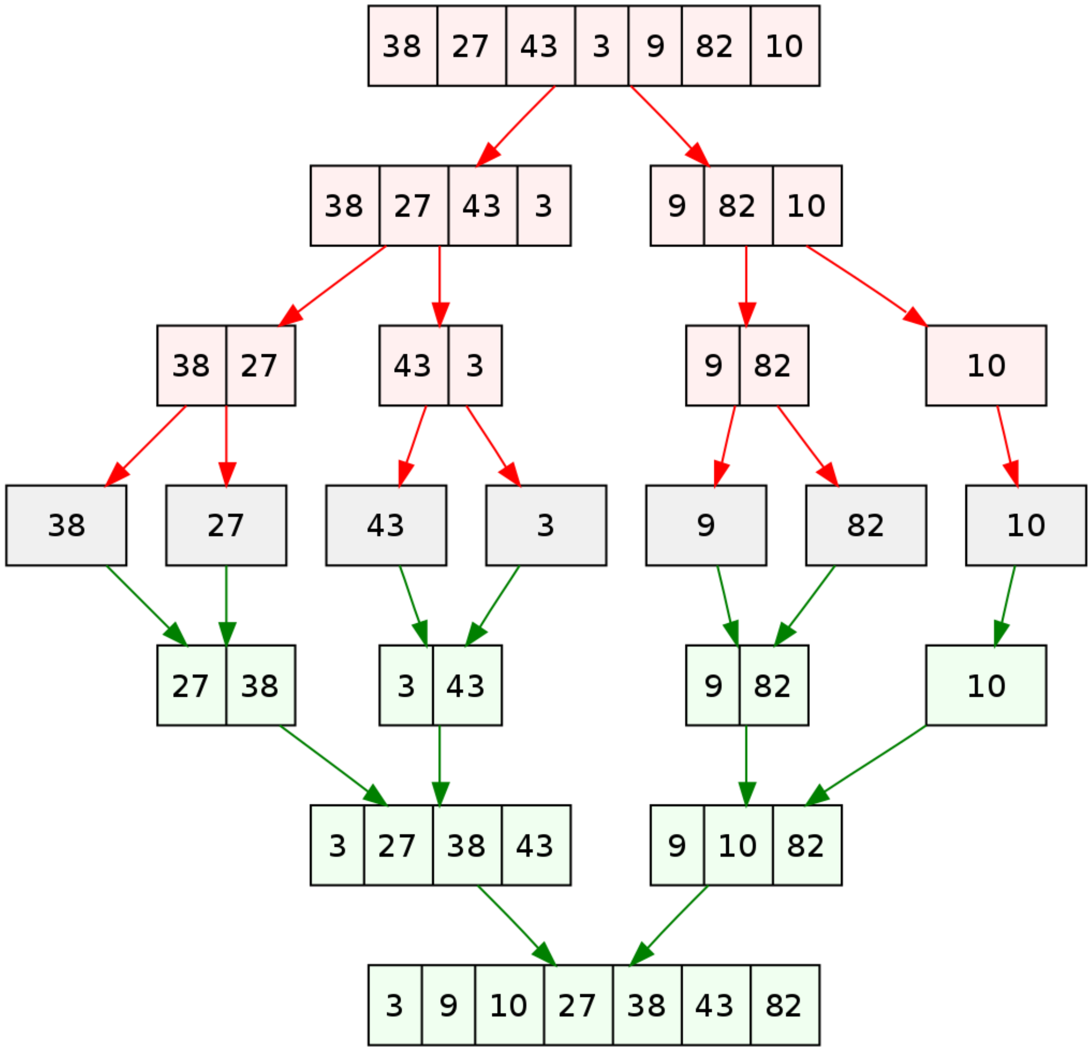
Merge sort is a highly popular sorting algorithm that is based on the **divide-and-conquer** principle.

Divide-and-conquer is a problem-solving method that involves breaking a big problem into smaller problems that are easier to solve. These smaller problems are similar to the big problem, and they are solved one by one. When all the smaller problems are solved, their solutions are combined to solve the big problem.

We can view a divide-and-conquer algorithm as consisting of three main parts:

1. **Divide:** Break the problem into smaller subproblems that can be solved independently.
2. **Conquer:** Solve each subproblem recursively using the same divide-and-conquer algorithm.
3. **Combine:** Combine the solutions of the subproblems to solve the original problem.

How it works:


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Code:

```
void merge(int l, int md, int r, vector<int>& arr) {
   vector<int> L, R;
   for (int i = l; i <= md; i++)
       L.push_back(arr[i]);
   for (int i = md + 1; i <= r; i++)
       R.push_back(arr[i]);
   L.push_back(INT_MAX);
   R.push_back(INT_MAX);
   int lp = 0, rp = 0;
   for (int i = l; i <= r; i++) {
       if (L[lp] <= R[rp])
          arr[i] = L[lp++];
       else
          arr[i] = R[rp++];
   }
}

void mergeSort(int l, int r, vector<int>& arr) {
   if (l >= r) return;
   int mid = (l + r) / 2;
   mergeSort(l, mid, arr);
   mergeSort(mid + 1, r, arr);
   merge(l, mid, r, arr);
}
```

Divide:

If 'mid' represents the middle point between 'l' and 'r', then we can divide the subarray A[l..r] into two separate subarrays, namely A[l..mid] and A[mid+1..r].

Conquer:

During this step, we make an attempt to sort both subarrays A[l..mid] and A[mid+1..r]. If we have not yet reached the base case, we continue to recursively divide both these subarrays and attempt to sort them.

Combine:

When the conquer step reaches the base level and two sorted subarrays A[l..mid] and A[mid+1, r] are obtained for array A[l..r], the sorted subarrays are merged by creating a new sorted array A[l..r] from A[l..mid] and A[mid+1, r]. The algorithm then moves up to the next higher level of recursion and proceeds to merge every adjacent pair of sorted subarrays into one larger sorted subarray. This process is repeated until the entire array is sorted.

Advantages:

- Merge sort is a fast and efficient sorting algorithm that works well for large data inputs. It has a time complexity of O(n log n), which means it can sort an array quickly regardless of its size.

Disadvantages:

- Requires auxiliary space of O(n).

Bubble Vs Insertion Vs Merge:

Sorting Algorithm	Time Complexity	Space Complexity
Bubble Sort	$O(n^2)$	$O(1)$
Insertion Sort	$O(n^2)$	$O(1)$
Merge Sort	$O(n * log(n))$	$O(n)$

To solve:

- 75. Sort Colors
- Sorting: Bubble Sort
- Insertion Sort - Part 2
- Merge Sort: Counting Inversions