Searching & Sorting

1. Binary Search

- Divide and conquer technique is used
- Sorted array has to be used which is prerequisite for this algorithm. Complexity is 0(log n)

You have to implement it in two different ways:

A. Iterative
B. Recursive

Note: Consider the array is sorted in ascending order. Pseudo-code is given below:

Recursive Pseudocode:

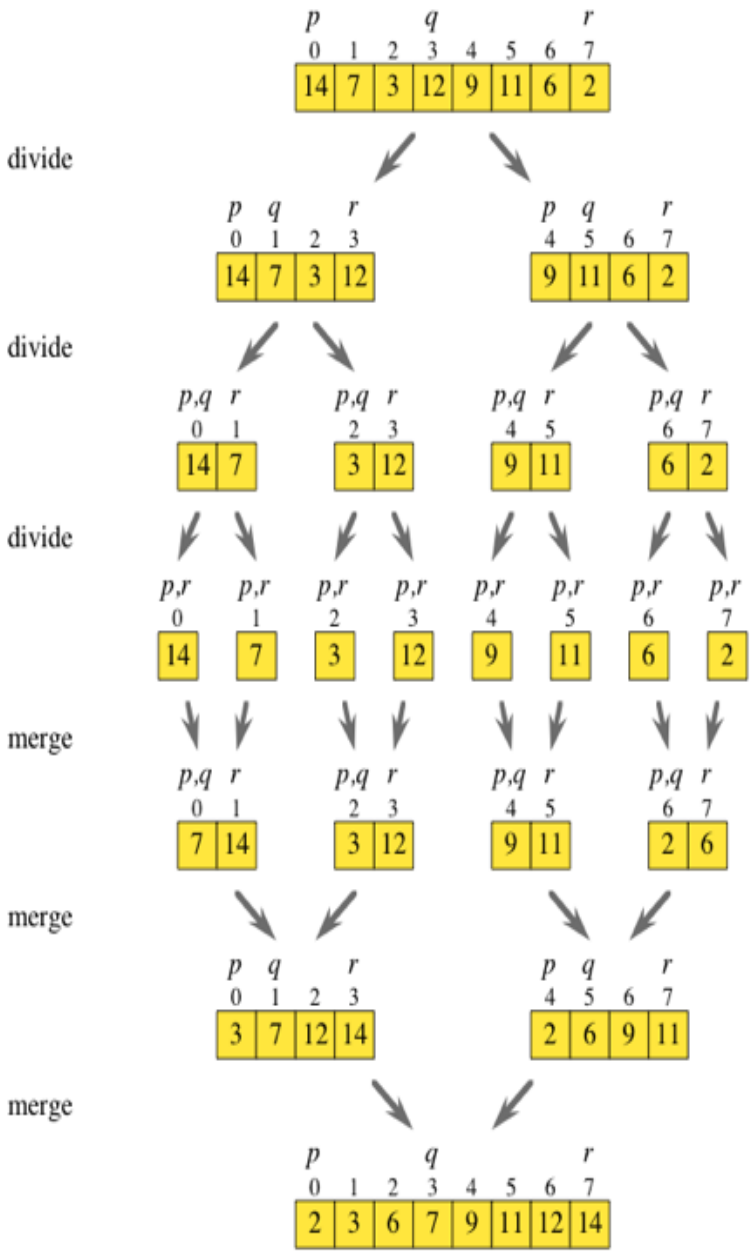
```
// initially called with low = 0, high = N-1
BinarySearch(A[0..N-1], value, low, high) {
    // invariants: value > A[i] for all i < low
                   value < A[i] for all i > high
    if (high < low)
        return not_found // value would be inserted at index "low"
    mid = (low + high) / 2
    if (A[mid] > value)
        return BinarySearch(A, value, low, mid-1)
    else if (A[mid] < value)
        return BinarySearch(A, value, mid+1, high)
    else
        return mid
}
```

Iterative Pseudocode:

```
BinarySearch(A[0..N-1], value) {
    low = 0
    high = N - 1
    while (low <= high) {
        // invariants: value > A[i] for all i < low
                       value < A[i] for all i > high
        mid = (low + high) / 2
        if (A[mid] > value)
            high = mid - 1
        else if (A[mid] < value)
            low = mid + 1
        else
            return mid
    }
    return not_found // value would be inserted at index "low"
}
```

2. Merge Sort

- Divide and conquer technique is used
- Time complexity is 0(n log (n))



A. Merge Sort pseudo-code is given below:

```
MergeSort(array A, int p, int r) {
    if (p < r) {                              // we have at least 2 items
        q = (p + r)/2
        MergeSort(A, p, q)                    // sort A[p..q]
        MergeSort(A, q+1, r)                  // sort A[q+1..r]
        Merge(A, p, q, r)                     // merge everything together
    }
}
```

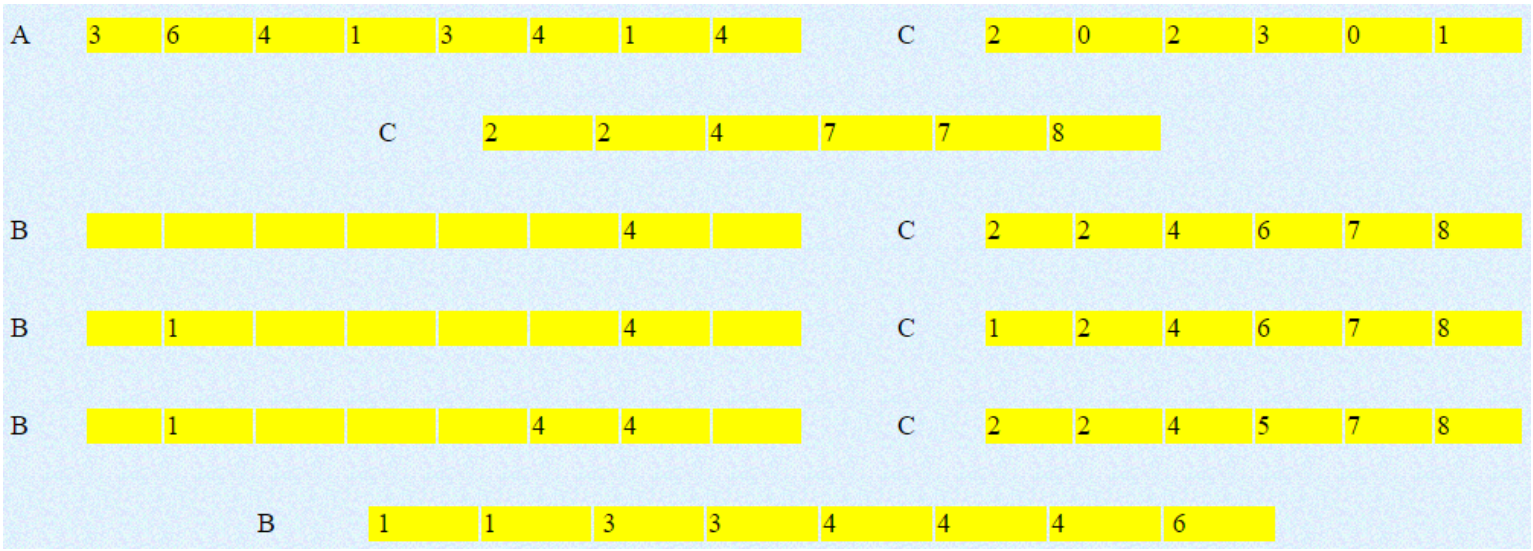
```
Merge(array A, int p, int q, int r) {       // merges A[p..q] with A[q+1..r]

    array B[p..r]
    i = k = p                                // initialize pointers
    j = q+1
    while (i <= q and j <= r) {              // while both subarrays are nonempty
        if (A[i] <= A[j]) B[k++] = A[i++]    // copy from left subarray
        else              B[k++] = A[j++]    // copy from right subarray
    }
    while (i <= q) B[k++] = A[i++]           // copy any leftover to B
    while (j <= r) B[k++] = A[j++]
    for i = p to r do A[i] = B[i]            // copy B back to A
}
```

4. Counting sort

- Array A[ ] stores the initial data to be sorted.
- Array C[ ] is used to count the occurrences of the data values
- Array B[ ] is used to store the final, sorted, list.
- Time complexity is O(*n*)
- https://www.cs.usfca.edu/~galles/visualization/CountingSort.html

```
COUNTING_SORT (A, B, k)
1. for i ← 1 to k do
2.     c[i] ← 0
3. for j ← 1 to n do
4.     c[A[j]] ← c[A[j]] + 1
5. //c[i] now contains the number of elements equal to i
6. for i ← 2 to k do
7.     c[i] ← c[i] + c[i-1]
8. // c[i] now contains the number of elements ≤ i
9. for j ← n downto 1 do
10.    B[c[A[j]]] ← A[j]
11.    c[A[j]] ← c[A[j]] - 1
```



Selection Sort

Suppose A is an array of N values. We want to sort A in ascending order. That is, A[1] should be the smallest and A[N] should be the largest.

The idea of Selection Sort is that we repeatedly find the smallest element in the unsorted part of the array and swap it with the first element in the unsorted part of the array.

```
For I = 1 to N-1 do:
  Smallsub = I
  For J = I + 1 to N-1 do:
    If A(J) < A(Smallsub)
      Smallsub = J
    End-If
  End-For
  Temp = A(I)
  A(I) = A(Smallsub)
  A(Smallsub) = Temp
End-For
```

Insertion Sort

Suppose A is an array of N values. We want to sort A in ascending order.

Insertion Sort is an algorithm to do this as follows: We traverse the array and insert each element into the sorted part of the list where it belongs. This usually involves pushing down the larger elements in the sorted part.

```
For I = 2 to N
  J = I
  Do while (J > 1) and (A(J) < A(J - 1)
    Temp = A(J)
    A(J) = A(J - 1)
    A(J - 1) = Temp
    J = J - 1
  End-Do
End-For
```

Bubble Sort

Suppose A is an array of N values. We want to sort A in ascending order.

Bubble Sort is a simple-minded algorithm based on the idea that we look at the list, and wherever we find two consecutive elements out of order, we swap them. We do this as follows: We repeatedly traverse the unsorted part of the array, comparing consecutive elements, and we interchange them when they are out of order. The name of the algorithm refers to the fact that the largest element "sinks" to the bottom and the smaller elements "float" to the top.

```
For I = 1 to N - 1
  For J = 1 to N - 1
    If (A(J) > A(J + 1)
      Temp = A(J)
      A(J) = A(J + 1)
      A(J + 1) = Temp
    End-If
  End-For
End-For
```

5. Implement and simulate quick sort on your own. You can take help from online or discuss with your class mates.

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```