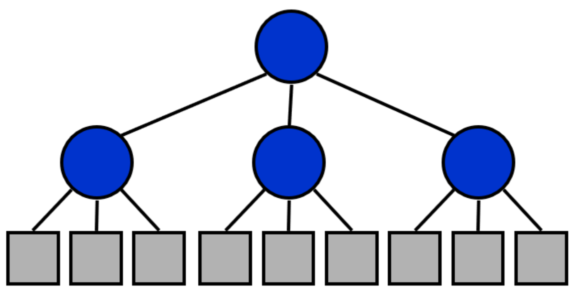
In computer science, divide and conquer is an algorithm design paradigm. A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

Divide and conquer steps:

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem
2. **Conquer** the subproblems by solving them recursively
3. **Combine** the solutions to the subproblems into the solution for the original problem
4. The **base case** for the recursion is subproblems of constant size



Practice problems:

Instructions:

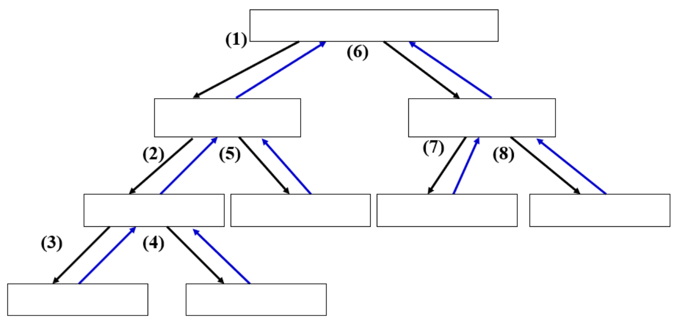
1. Do not adopt unfair means. **10 marks will be deducted from the final marks for adopting unfair means**.
2. No more than 40% marks for uncompilable codes.

1. Find the max and min element of an array.

```
Function MaxMin(A):
    fmax ← fmin ← A (1);
    for i ← 2 to n do
        if (A (i) > fmax) then fmax ← A (i);
        if (A (i) < fmin) then fmin ← A (i);
    return fmax, fmin
```

```
Function RMaxMin(A, i, j):
    if i==j:
        return A[i], A[i]
    else
        mid ← (i+j)/2
        gmax, gmin ← RMaxMin(A, i, mid)
        hmax, hmin ← RMaxMin(A, mid+1, j)
        fmax ← max(gmax, hmax)
        fmin ← min(gmin, hmin)
        return fmax, fmin
```

Index:	1	2	3	4	5	6	7	8	9
Array:	22	13	-5	-8	15	60	17	31	47



2. X^Y

3. Merge sort

```
MERGE-SORT(A, p, r)
if p < r
   then q ← ⌊(p + r)/2⌋
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1 . . n1 + 1] and R[1 . . n2 + 1]
for i ← 1 to n1
     do L[i] ← A[p + i − 1]
for j ← 1 to n2
     do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
     do if L[i] ≤ R[j]
           then A[k] ← L[i]
                i ← i + 1
           else A[k] ← R[j]
                j ← j + 1
```

Simulation in slide.

4. Count Inversion

https://www.cp.eng.chula.ac.th/~prabhas//teaching/algo/algo2008/count-inv.htm

The sequence 2, 4, 1, 3, 5 has three inversions (2,1), (4,1), (4,3).

The idea is similar to "merge" in merge-sort. Merge two sorted lists into one output list, but we also count the inversion.

- divide: size of sequence n to two lists of size n/2
- conquer: count recursively two lists
- combine: this is a trick part (to do it in linear time)

5. Quick Sort

Practice yourself.

6. Maximum-sum subarray

```
Function FIND-MAXIMUM-SUBARRAY (A, low, high):

    if high == low /// base case: only one element
        return (low, high, A[low])
    else
        mid = (low+high)/2

        left-low, left-high, left-sum = FIND-MAXIMUM-SUBARRAY(A, low, mid)
        right-low, right-high, right-sum = FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
        cross-low, cross-high, cross-sum = FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)

        if left-sum >= right-sum and left-sum >= cross-sum
            return (left-low, left-high, left-sum)
        else if right-sum >= left-sum and right-sum >= cross-sum
            return (right-low, right-high, right-sum)
        else
            return (cross-low, cross-high, cross-sum)
```

```
Function FIND-MAX-CROSSING-SUBARRAY (A, low, mid, high):

    ///Find a maximum subarray of the form A[i..mid]
    left-sum = -∞
    sum = 0
    for i = mid downto low
        sum = sum + A[i ]
        if sum > left-sum
            left-sum = sum
            max_left = i
    ///Find a maximum subarray of the form A[mid + 1 .. j ]
    right-sum = -∞
    sum =0
    for j = mid +1 to high
        sum = sum + A[j]
        if sum > right-sum
            right-sum = sum
            max_right = j
    ///Return the indices and the sum of the two subarrays
    return (max_left, max_right, left-sum + right-sum)
```

0 1 2 3 4 5 6 7

2 -3 4 2 || 1 -2 5 -1

7. Longest common prefix of n strings:

8. Closest pair of points

Ref: https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pearson/05DivideAndConquer.pdf

```
Closest-Pair(p1, …, pn) {
   Compute separation line L such that half the points        O(n log n)
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)                                2T(n / 2)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Delete all points further than δ from separation line L     O(n)

   Sort remaining points by y-coordinate.                      O(n log n)

   Scan points in y-order and compare distance between
   each point and next 11 neighbors. If any of these           O(n)
   distances is less than δ, update δ.

   return δ.
}
```

Running time:

$$T(n) \leq 2T\left(\frac{n}{2}\right) + O(n \log n) \Rightarrow T(n) \;=\; O(n \log^2 n)$$

Can we achieve $\boldsymbol{O(n\ log\ n)}$?

Yes. Don't sort points in strip from scratch each time.

- Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sort by merging two pre-sorted lists.
- $T(n) \leq 2T\left(\frac{n}{2}\right) + O(n) \Rightarrow T(n) = O(n \log n)$

9. More practice problems: https://leetcode.com/tag/divide-and-conquer/

Reference:

- Slides of Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET