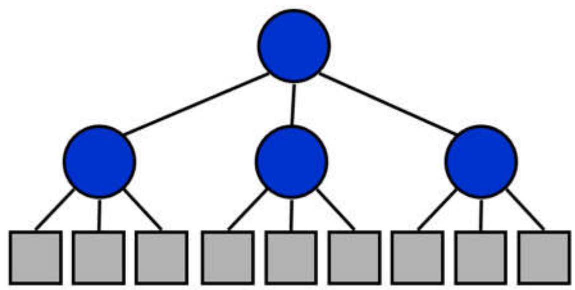
In computer science, divide and conquer is an algorithm design paradigm. A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined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

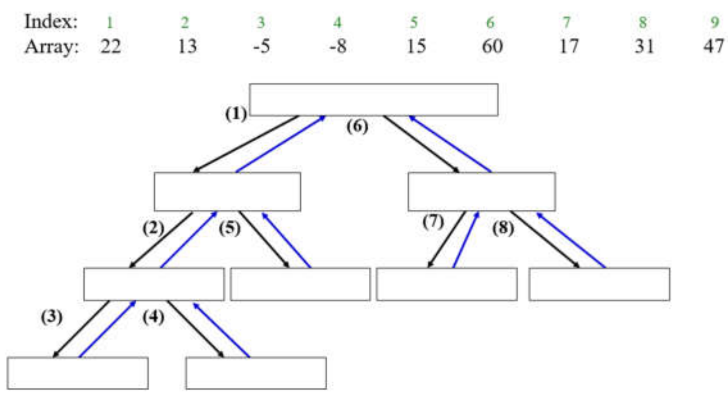
For loop version:

```
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
```

Divide and Conquer version:

```
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
```

Recursion Tree:



2. X^Y

Hint:

$$3^{100} = 3^{50} . 3^{50} = (3^{\frac{100}{2}}).(3^{\frac{100}{2}})$$

$$3^{50} = 3^{25} . 3^{25}$$

$$3^{25} = 3^{12} . 3^{12} . 3$$

$$3^{12} = 3^{6} . 3^{6}$$

$$3^{6} = 3^{3} . 3^{3}$$

$$3^{3} = 3^{1} . 3^{1} . 3$$

$$3^{1} = 3^{0} . 3^{0} . 3$$

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
```

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

The quick sort uses divide and conquer just like merge sort but without using additional storage.

The steps are:

1. Select an element q, called a pivot, from the array. In this algorithm we have chosen the last index as the pivot.

2. The PARTITION function finds the location of the pivot in such a way that all the elements smaller than the pivot is on the left side and all the element on the right-hand side of the pivot is greater in value. (Items with equal values can go either way).
3. Recursively call the QUICKSORT function which perform quicksort on the array on the left side of the pivot and then on the array on the right side, thus dividing the task into sub tasks. This is carried out until the arrays can no longer be split.

Implement Quick sort algorithm. The pseudo code is given below:

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

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

6. Binary Search

```
/**
 the following function returns
     the index of x in A;  if x exists,
     NOT_FOUND;            if x does not exist
*/
function Binary_Search (A, start, _end, x)
{
    if start <= _end then
        mid = (start+_end)/2

        if A[mid] == x then
            return mid

        if A[mid] > x then
            return Binary_Search(A, start, mid-1, x);

        if A[mid] < x
            return Binary_Search(A, start, mid-1, x);

        return NOT_FOUND;
}
```

Sample Input	Sample Output
Number of Elements: 5 Enter elements: 3 4 5 7 2 Key: 4	4 found in index 1
Number of Elements: 5 Enter elements: 3 4 5 7 2 Key: 14	14 not found

7.

Write a function `print_odd` using divide-and-conquer algorithm to print the odd numbers of an array of n integers.

8.

Write a function `calc_sum` using divide-and-conquer algorithm to calculate the sum of an array of n integers.

9.

Write a function `calc_sum` using divide-and-conquer algorithm to calculate the sum of the even numbers of an array of n integers.

10. 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)
```

11. Longest common prefix of n strings

Sample Input	Sample Output
3 Algolab Algorithms Algeria	Alg
4 Algolab Algorithms Algeria UIU	No common prefix

12. 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 $O(n\ log\ n)$?

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

13. More practice problems

https://leetcode.com/tag/divide-and-conquer/

Reference:

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