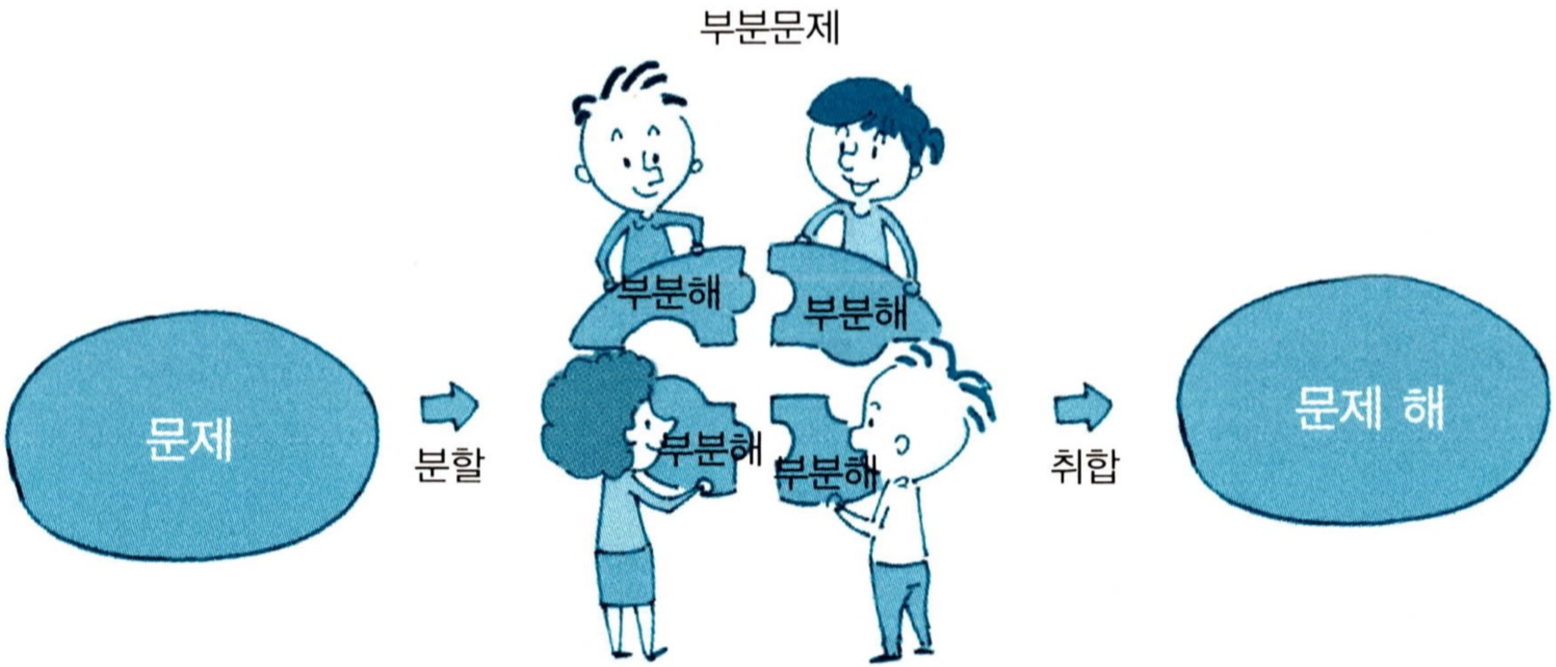
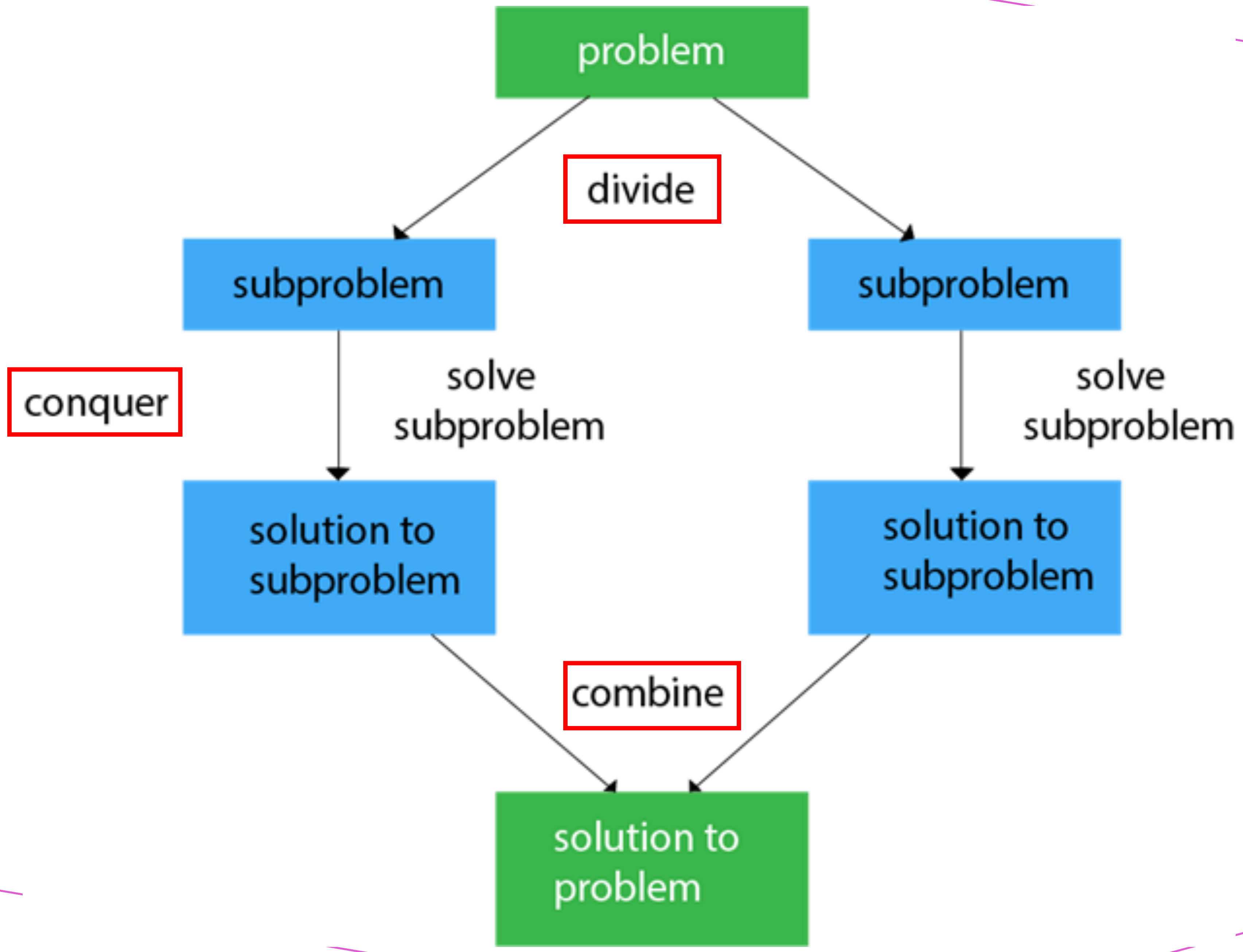
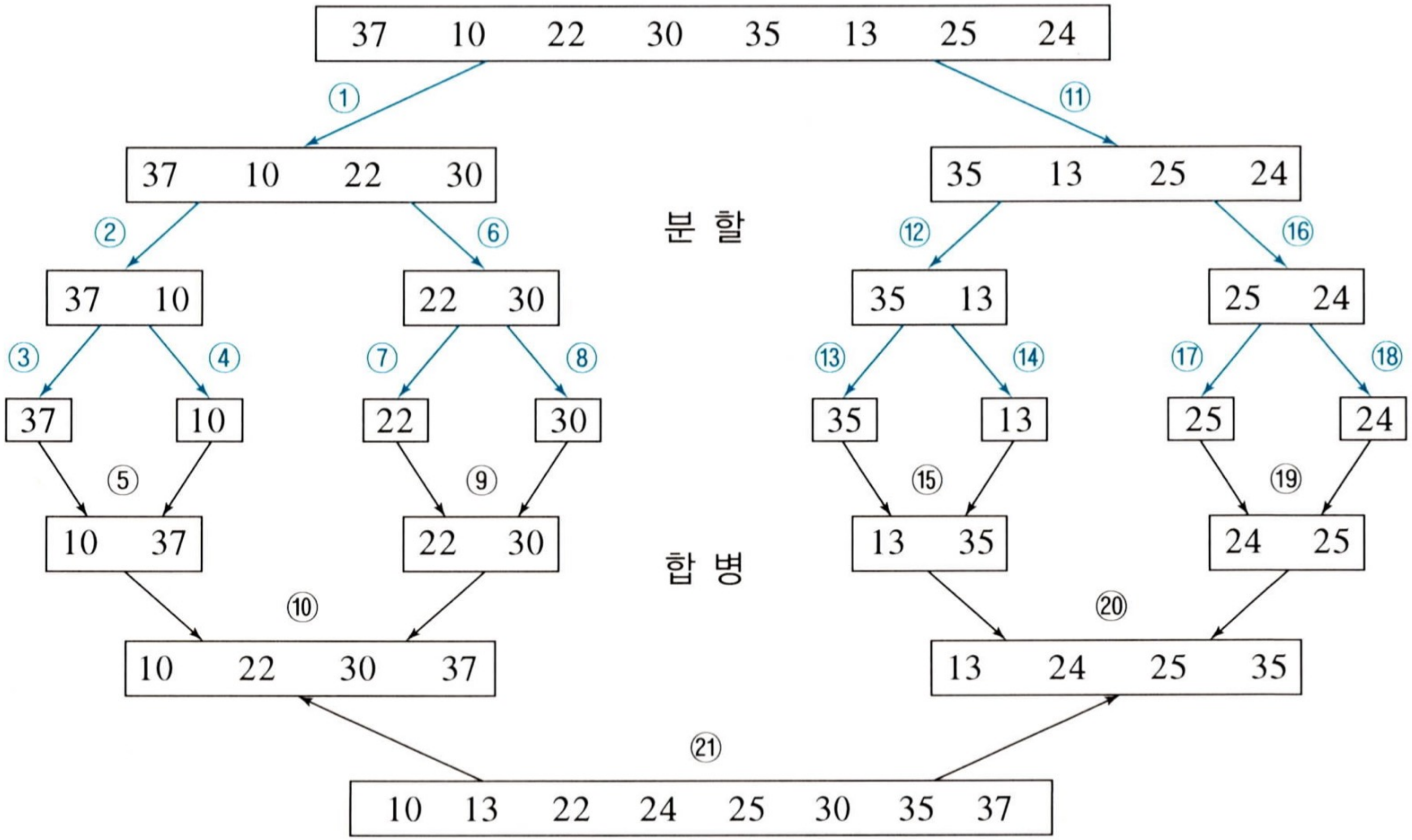
Divide And Conquer


부분문제
문제
분할
부분해
부분해
부분해
부분해
취합
문제 해


problem
divide
subproblem
subproblem
conquer
solve
subproblem
solve
subproblem
solution to
subproblem
solution to
subproblem
combine
solution to
problem

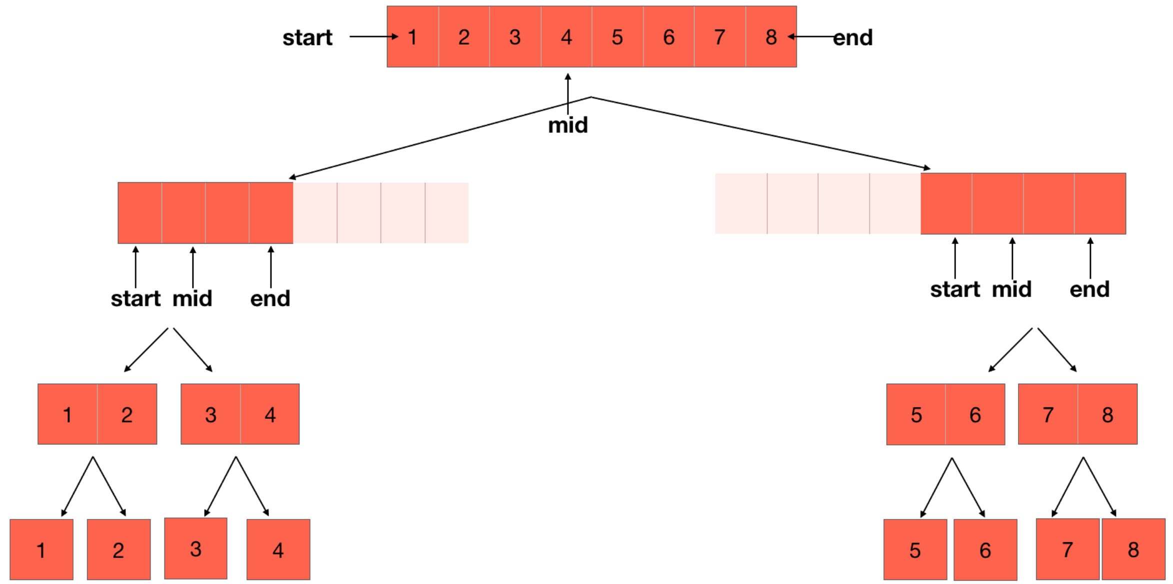
Example (merge sort)

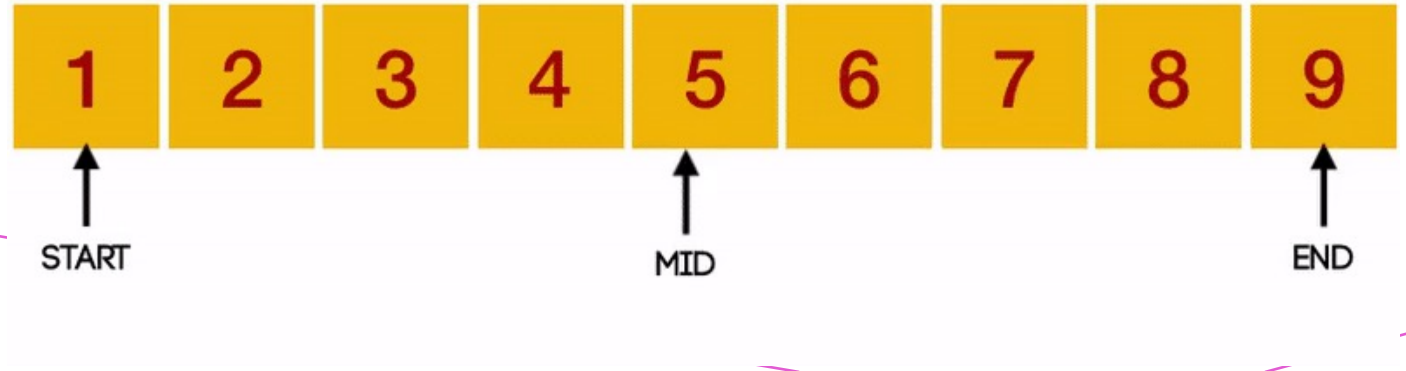


Example (merge sort)

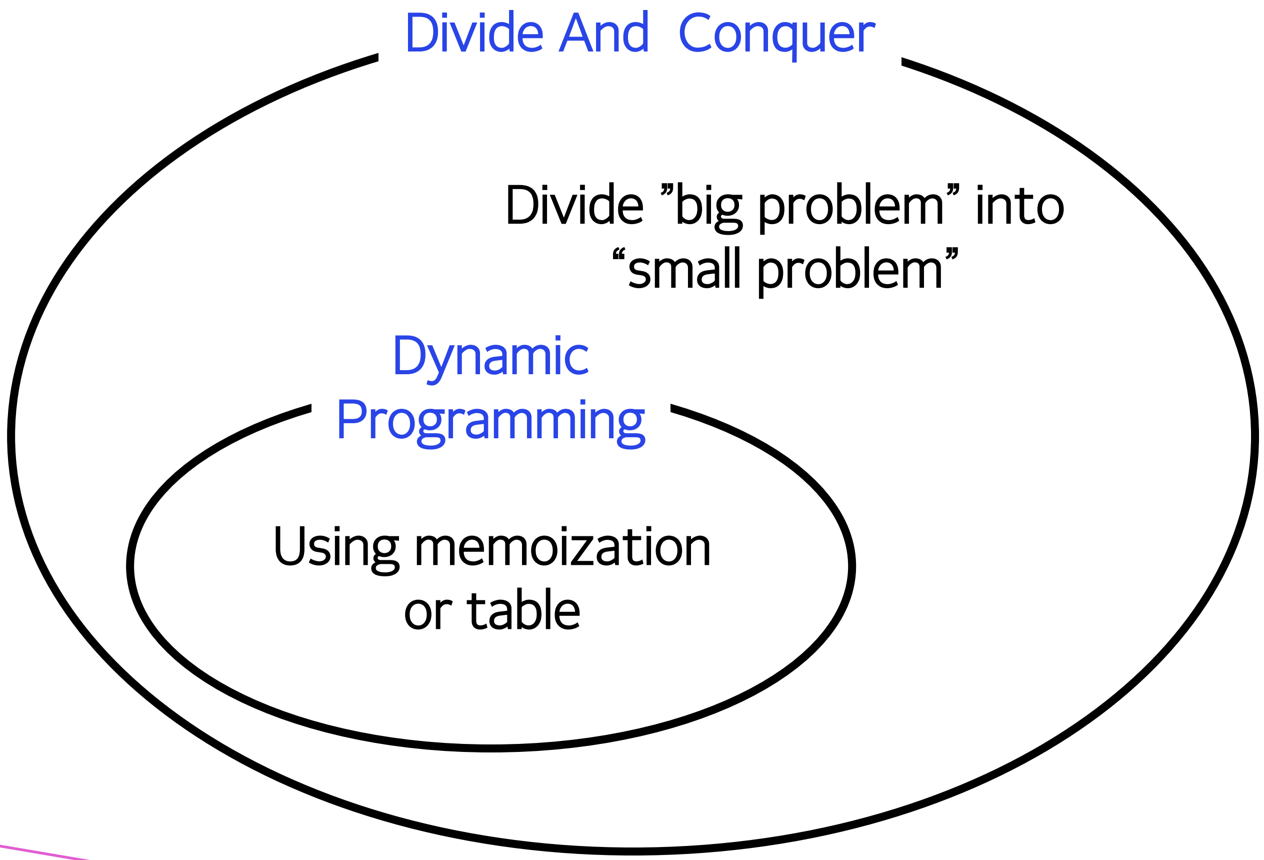
6 5 3 1 8 7 2 4

Example(Binary Search)

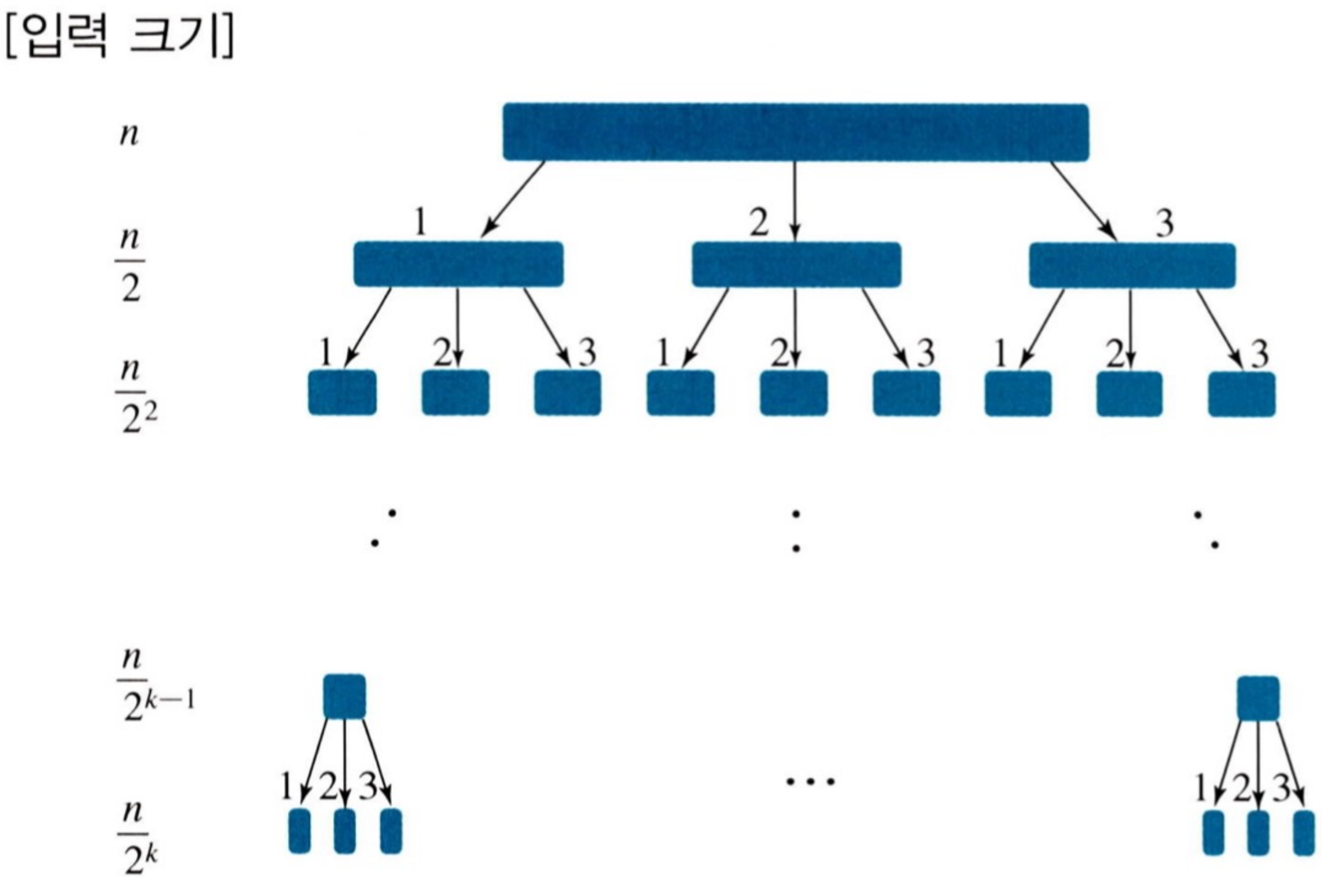




DP vs DC


Divide And Conquer
Divide "big problem" into "small problem"
Dynamic Programming
Using memoization or table

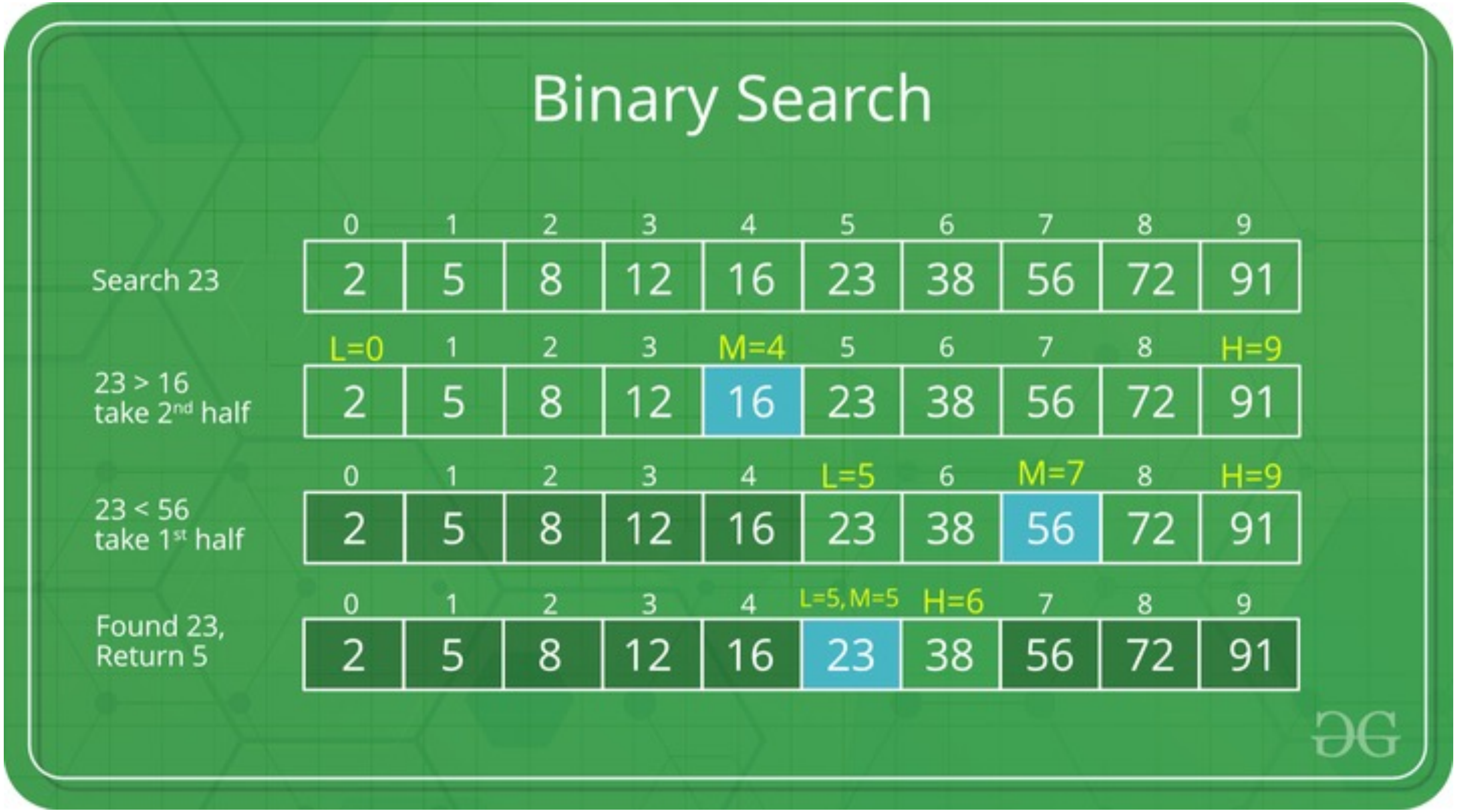
Time complexity



Standard Algorithm

1. Binary Search
2. Merge Sort
3. Quick Sort
4. Calculate Pow

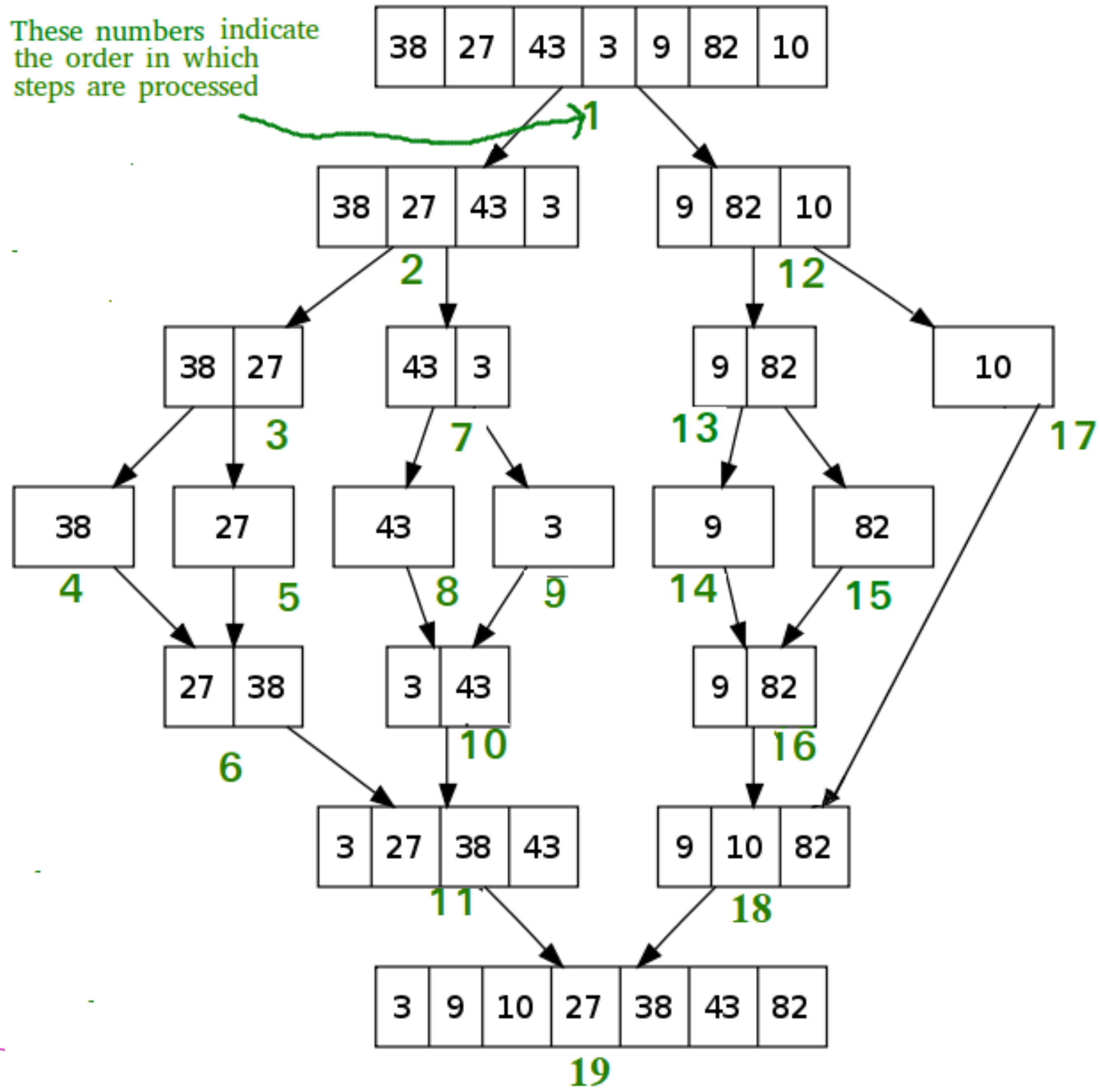
```python
# Returns index of x in arr if present, else -1
def binary_search(arr, low, high, x):

    # Check base case
    if high >= low:

        mid = (high + low) // 2

        # If element is present at the middle itself
        if arr[mid] == x:
            return mid

        # If element is smaller than mid, then it can only
        # be present in left subarray
        elif arr[mid] > x:
            return binary_search(arr, low, mid - 1, x)

        # Else the element can only be present in right subarray
        else:
            return binary_search(arr, mid + 1, high, x)

    else:
        # Element is not present in the array
        return -1
```


Binary Search
Search 23
0 1 2 3 4 5 6 7 8 9
2 5 8 12 16 23 38 56 72 91
23 > 16
take 2nd half
L=0 1 2 3 M=4 5 6 7 8 H=9
2 5 8 12 16 23 38 56 72 91
23 < 56
take 1st half
0 1 2 3 4 L=5 6 M=7 8 H=9
2 5 8 12 16 23 38 56 72 91
Found 23,
Return 5
0 1 2 3 4 L=5, M=5 H=6 7 8 9
2 5 8 12 16 23 38 56 72 91

Standard Algorithm

1. Binary Search
2. Merge Sort
3. Quick Sort
4. Calculate Pow

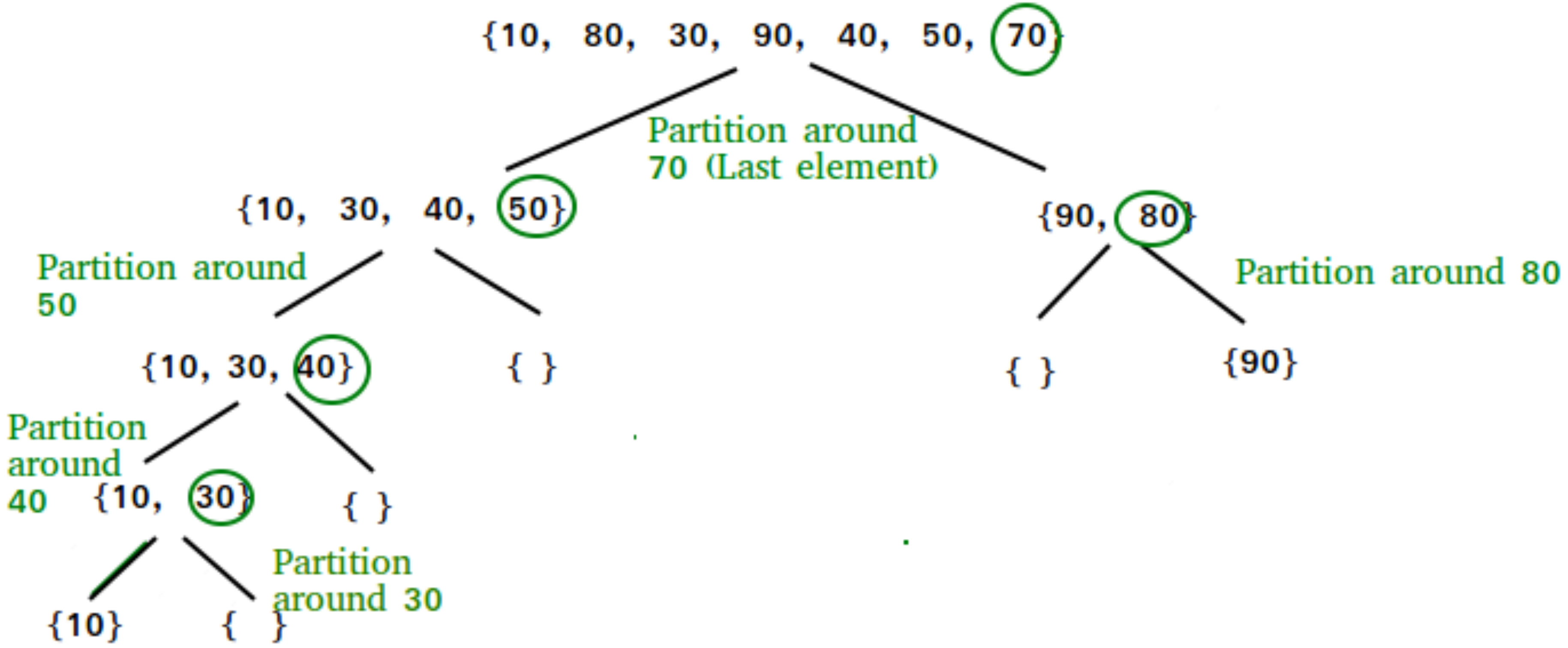
```python
# Python program for implementation of MergeSort
def mergeSort(arr):
    if len(arr) > 1:

         # Finding the mid of the array
        mid = len(arr)//2

        # Dividing the array elements
        L = arr[:mid]

        # into 2 halves
        R = arr[mid:]

        # Sorting the first half
        mergeSort(L)

        # Sorting the second half
        mergeSort(R)

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] <= R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        # Checking if any element was left
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
```


These numbers indicate the order in which steps are processed
38 27 43 3 9 82 10
1
38 27 43 3
9 82 10
2
12
38 27
43 3
9 82
10
3
7
13
17
38
27
43
3
9
82
4
5
8
9
14
15
27 38
3 43
9 82
6
10
16
3 27 38 43
9 10 82
11
18
3 9 10 27 38 43 82
19

Standard Algorithm

1. Binary Search
2. Merge Sort
3. Quick Sort
4. Calculate Pow

```python
# Function to find the partition position
def partition(array, low, high):

    # choose the rightmost element as pivot
    pivot = array[high]

    # pointer for greater element
    i = low - 1

    # traverse through all elements
    # compare each element with pivot
    for j in range(low, high):
        if array[j] <= pivot:

            # If element smaller than pivot is found
            # swap it with the greater element pointed by i
            i = i + 1

            # Swapping element at i with element at j
            (array[i], array[j]) = (array[j], array[i])

    # Swap the pivot element with the greater element specified by i
    (array[i + 1], array[high]) = (array[high], array[i + 1])

    # Return the position from where partition is done
    return i + 1

def quickSort(array, low, high):
    if low < high:

        # Find pivot element such that
        # element smaller than pivot are on the left
        # element greater than pivot are on the right
        pi = partition(array, low, high)

        # Recursive call on the left of pivot
        quickSort(array, low, pi - 1)

        # Recursive call on the right of pivot
        quickSort(array, pi + 1, high)
```


{10, 80, 30, 90, 40, 50, 70}
Partition around 70 (Last element)
{10, 30, 40, 50}
{90, 80}
Partition around 50
Partition around 80
{10, 30, 40}
{ }
{ }
{90}
Partition around 40
{10, 30}
{ }
Partition around 30
{10}
{ }

Standard Algorithm

1. Binary Search
2. Merge Sort
3. Quick Sort
4. Calculate Pow

```
def power(x, y):
    if (y == 0):
        return 1
    elif (int(y % 2) == 0):
        return (power(x, int(y / 2)) * power(x, int(y / 2)))
    else:
        return (x * power(x, int(y / 2)) * power(x, int(y / 2)))
```

Standard Algorithm

1. Binary Search
2. Merge Sort
3. Quick Sort
4. Calculate Pow

```
dp = [0] * MAX
def power(x, y):
    if (dp[y] != 0):
        return dp[y]
    if (y == 0):
        return 1
    elif (int(y % 2) == 0):
        dp[y] = (power(x, int(y / 2)) * power(x, int(y / 2)))
    else:
        dp[y] = (x * power(x, int(y / 2)) * power(x, int(y / 2)))

    return dp[y]
```