Divide And Conquer Algorithm

Dr. Samuel Cho, PhD

NKU

1. Divide And Conquer: What, Why, and How
2. Simple Example of Divide and Conquer
3. Not So Simple Example of Divide and Conquer

Divide And Conquer: What, Why, and How

What is Divide And Conquer

- Divide and Conquer (DC) works by dividing the problem into sub-problems, conquering each sub-problem recursively, and combining these solutions.
- Dynamic Programming (DP) is a technique for solving problems with overlapping subproblems.
- DC and DP are similar in that they divide the problem into smaller sub-problems.
- However, whereas DC combines the sub-solutions DP uses the results of sub-solutions to get a result without the computation to get the sub-solutions.

How to Use Divide And Conquer

- Divide the problem into a number of subproblems that are smaller instances of the same problem.
- Conquer the subproblems by solving them recursively. If they are small enough, solve the subproblems as base cases.
- Combine the solutions to the subproblems into the solution for the original problem.

Why Divide and Conquer

- Divide and conquer algorithms work faster because they end up doing less work.
- Consider the classic divide-and-conquer algorithm of binary search: rather than looking at N items to find an answer, the binary search ends up checking only $Log_2 N$ of them.

Code

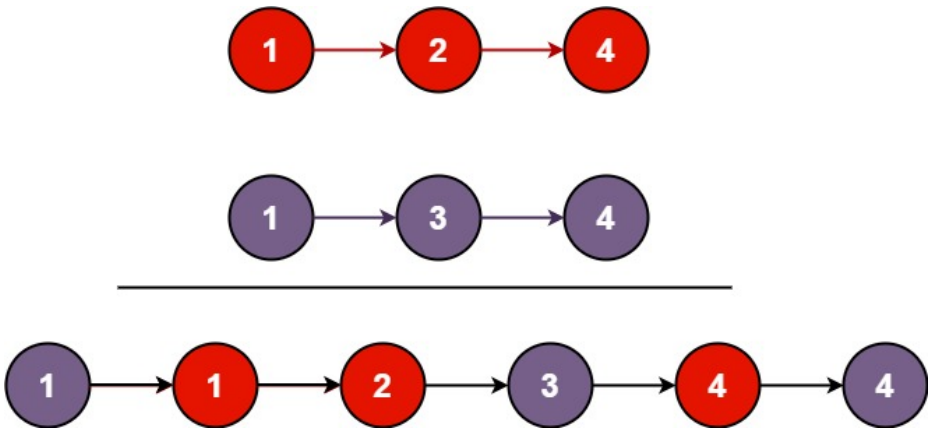
```
1  def binary_search(arr, low, high, x):
2      # Check base case
3      if high >= low:
4          mid = (high + low) // 2
5          if arr[mid] == x:
6              return mid
7          elif arr[mid] > x: # Divide
8              return binary_search(arr, low, mid - 1, x)
9          else: # Divide
10             return binary_search(arr, mid + 1, high, x)
11     else:
12         # Element is not present in the array
13         return -1
```

Simple Example of Divide and Conquer

Question

- You are given the heads of two sorted linked lists, list1 and list2.
- Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists.
- Return the head of the merged linked list.

- Input: list1 = [1,2,4], list2 = [1,3,4]
- Output: [1,1,2,3,4,4]



- We need recursion to merge two lists: X and Y.
- We start from an empty list A.
- We get the first elements and select the smaller one in A. Let's say X[0] is smaller than Y[0].
- Then we merge two lists with X[1:] and Y recursively.
- We also need to consider the terminal case when one of the lists is empty or Null.

Code

```
result = []
def mergeLists(l1, l2):
    if not l1 or len(l1) == 0:
        result.extend(l2)
        return result
    if not l2 or len(l2) == 0:
        result.extend(l1)
        return result

    if l1[0] < l2[0]:
        result.append(l1[0])
        mergeLists(l1[1:], l2)
    else:
        result.append(l2[0])
        mergeLists(l1, l2[1:])
    return result
```

Not So Simple Example of Divide and Conquer

Example

- You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.
- Merge all the linked-lists into one sorted linked-list and return it.

- Input: lists = [[1,4,5],[1,3,4],[2,6]]
- Output: [1,1,2,3,4,4,5,6]

Examples

```
The lists are
[
  1->4->5,
  1->3->4,
  2->6
]

merging them into one sorted list:
1->1->2->3->4->4->5->6
```

Idea And Implementation

- We already know how to merge two lists.
- We can use the solution of merging two lists to solve this problem.
- Divide and conquer algorithm iteratively can be applied to solve this problem.

Idea And Solution

- We merge two lists over and over again to make only one list.
- We use mergeLists() to merge two lists.
- Notice that when we can't make a pair, we just use one list.

Examples

```
[[A],[B],[C],[D],[E]] (len(lists) = 5) ->
[[A,B],[C,D],[E]] (len(lists) = 3)->
[[A,B,C,D],[E]] (len(lists) = 2) ->
[[A,B,C,D,E]] (return lists[0])=> [A,B,C,D,E]
```

- This is the code that implements the idea.

Code #

```
while len(lists) > 1:
    merged = []
    for i in range(0, len(lists), 2):
        l1 = lists[i]
        l2 = lists[i + 1] if (i + 1) < len(lists)
            ↪ else None
        result = []  # clear the result
        merged.append(mergeLists(l1, l2))
    lists = merged

return lists[0]
```