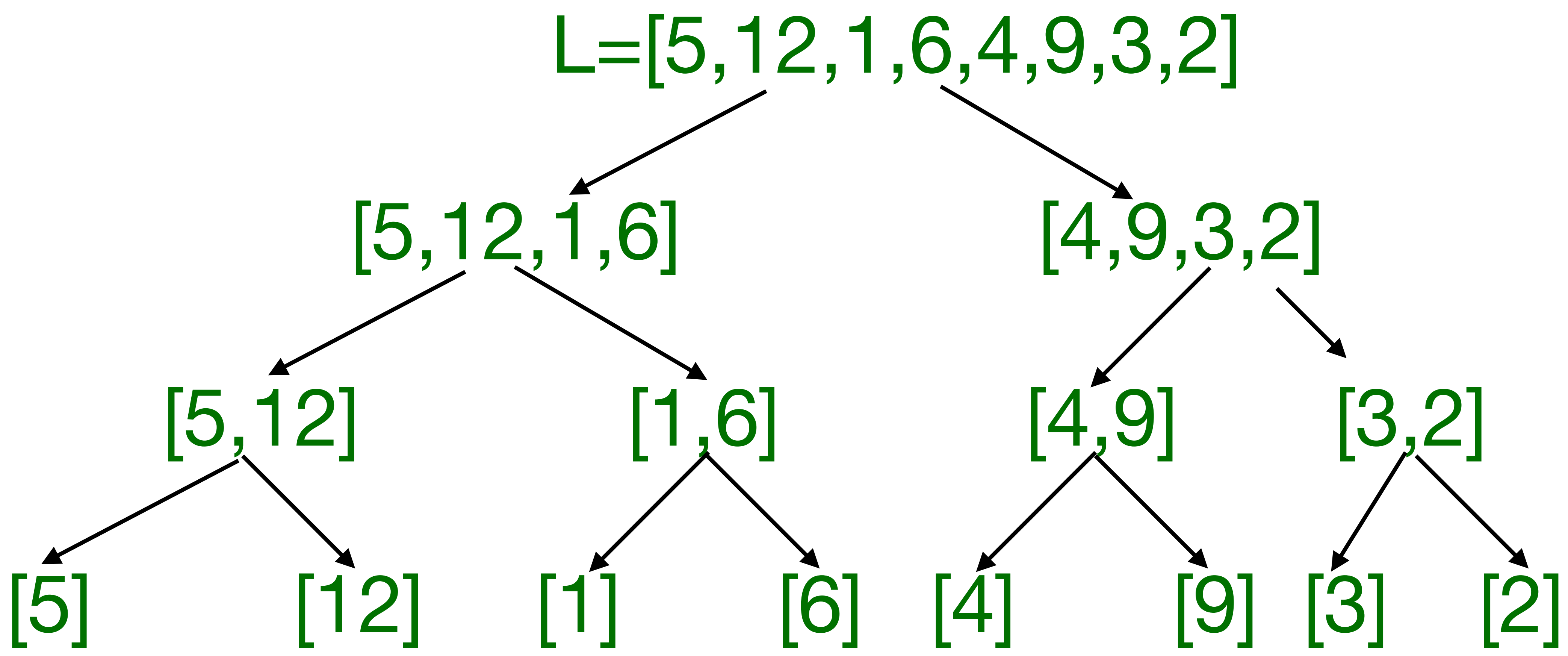
Iterative MergeSort & Algorithms on Sorted Lists

MergeSort

- Recall Basic Algorithm

```
Mergesort(L)
      if List is of length 1 return L otherwise
      — split L into two equal lists L1 & L2
      — recursively sort L1 & L2
      — Merge L1 & L2 and return merged list
```

Merge Sort



Merge Sort

[5, 12, 1, 6, 4, 9, 3, 2] -> n sorted listed of length 1

[5,12 1,6 4,9 2,3] -> n/2 sorted lists of length 2

[1,5,6,12] [2,3,4,9] -> n/4 sorted lists of length 4

[1,2,3,4,5,6,9,12] -> n/8 sorted lists of length 8

log n levels of merging

Time Complexity: O(n log n)

Space Complexity: Trivial analyse O(n log n) — can we reuse space?

Merge Sort

Space optimization:

When we merge the lists pairwise we need to output the result to a new array!

Can we do it with just two arrays A & B?

A [5, 12, 1, 6, 4, 9, 3, 2] -> n sorted listed of length 1

B [5,12 1,6 4,9 2,3] -> n/2 sorted lists of length 2

A [1,5,6,12] [2,3,4,9] -> n/4 sorted lists of length 4

B [1,2,3,4,5,6,9,12] -> n/8 sorted lists of length 8

```
# Merges two subarrays of arr[] write the output to b[l:r]
# First subarray is arr[l:m] # Second subarray is arr[m:r]
def mergeAB(arr,b, l, m, r):
   i = l   # Initial index of first subarray
   j = m   # Initial index of second subarray
   k = l   # Initial index of merged subarray
   while i < m and j < r :
      if arr[i] <= arr[j]:
         b[k] = arr[i]
         i += 1
      else:
         b[k] = arr[j]
         j += 1
      k += 1

   # Copy the remaining elements of arr[i:m], if there are any
   while i < m:
       b[k] = arr[i]
       i += 1
       k += 1
   # Copy the remaining elements of arr[j:r], if there are any
   while j < r:
       b[k] = arr[j]
       j += 1
       k += 1
```

```
def mergeIt(A,B,n,l):
#  A of size n consists of n/l sorted lists of size l each [last list may be shorter]
#  merge them in pairs writing the result to B [there may be one unpaired if not even]
    if n%l == 0:
        count=n//l
    else:
        count=n//l + 1
    for i  in range( count//2 ):
        left=i*l*2
        right=min(left+2*l,n)
        mergeAB(A,B,left,left+l,right)
    # Copy the last list if there is any (may happen if count is odd)
    for i in range(right,n):
        B[i]=A[i]
```

```
def mergeSort(A):
    n=len(A)
    l=1
    B=[0 for x in range(n)]
    dir=0
    while l < n:
        if dir == 0:
            mergeIt(A,B,n,l)
            dir=1
        else:
            mergeIt(B,A,n,l)
            dir=0
        l*=2
    #if result is in B copy result to A
    if dir==1:
        for i in range(n):
            A[i]=B[l]
```

Algorithms on Sorted List

- On a sorted list L, some things become trivial — eg
    - find min, find max or find kth largest
    - removing duplicates
- How about if we represent Sets by Sorted Lists
    - Set Union, Intersection can be done in Linear (O(n)) time.
    - How about set membership i.e is element *e* in S?
        - Can we do better than Sequential Search.

Binary Search

An example of Divide & Conquer

- Check if value k is in a sorted array A[left..right]
- Since A is sorted if you compare k and A[mid] (where mid=(left+right)/2)) we can discard half the search space

```
if k == A[mid]    return true
if k>A[mid]
        search in A[mid+1..right]
else
        search in A[left..mid-1]
```

```
def binary_search(arr, x):
    left = 0
    right = len(arr) - 1
    mid = 0
    while left <= right:
        mid = (left + right) // 2
       # Check if x is present at mid
        if x == arr[mid]:
           return mid
        if arr[mid] < x:
            left = mid + 1
        # If x is greater, ignore left half
        else:
            right = mid - 1
        # If x is smaller, ignore right half

    # If we reach here, then the element was not present
    return -1
```

Complexity

$T(n) = 1 + T(n/2)$

$= 1 + 1 + T(n/4)$

…..

$= 1 + 1 + .. + 1$ log n times $= O(\log n)$