EXPERIMENT-1
MERGESORT

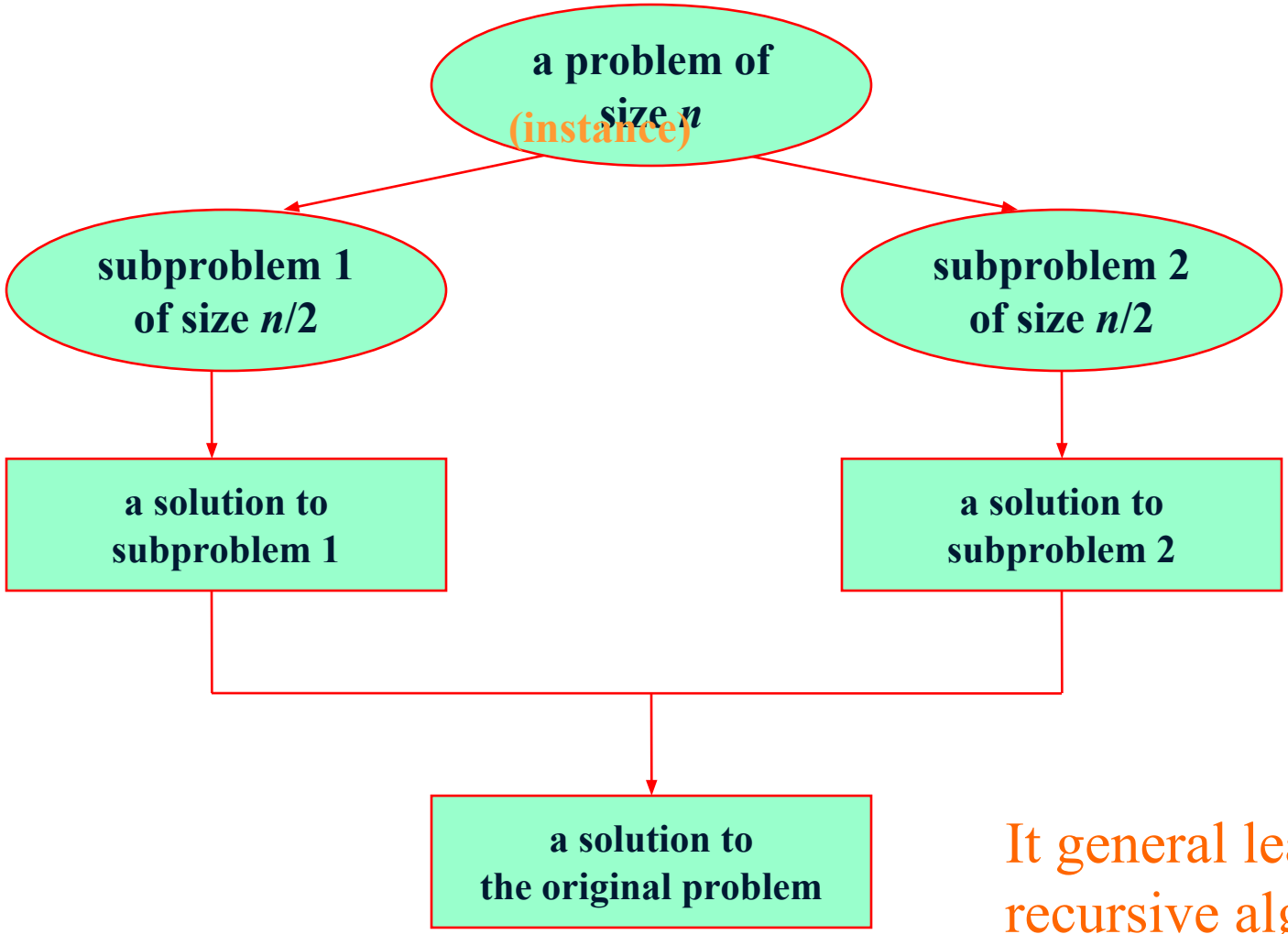
Implement Mergesort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

DIVIDE-AND-CONQUER

The most-well known algorithm design strategy:

- Divide instance of problem into two or more smaller instances
- Solve smaller instances recursively
- Obtain solution to original (larger) instance by combining these solutions

DIVIDE-AND-CONQUER TECHNIQUE (CONT.)



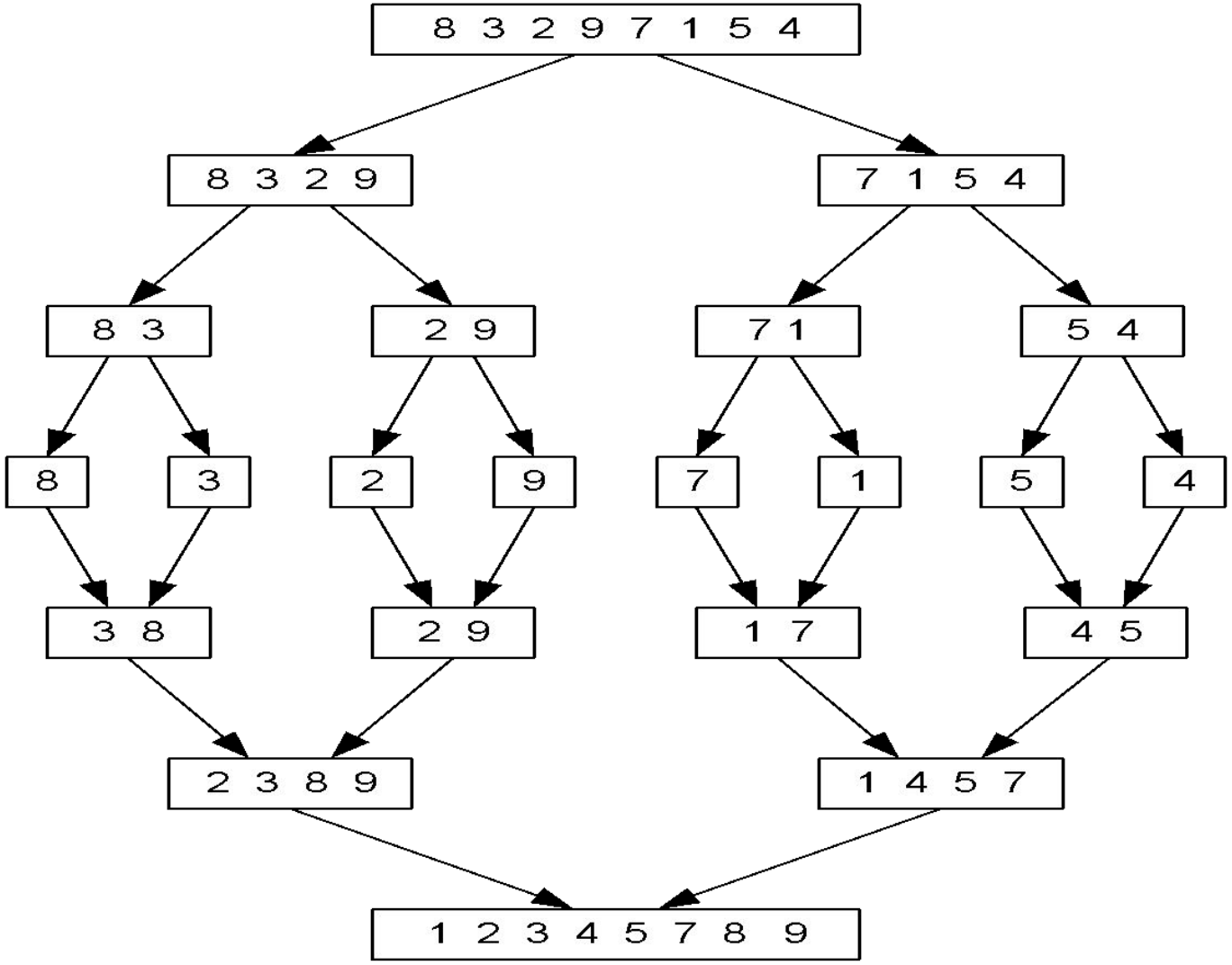
It general leads to a recursive algorithm!

PSEUDOCODE OF MERGESORT

```
Algorithm Mergesort(A,low,high)
//A[low:high] is a global array to be sorted
//small(P) is true if there is only one element
//to sort. In this case the list is already sorted.
{
    if (low<high) then
    {
        // Divide P into subproblems
        //find where to split the set
        mid:=[(low+high)/2];
        //solve the subproblems.
        Mergesort(A,low,mid);
        Mergesort(A,mid+1,high);
        //combine the solutions.
        Merge(A,low,mid,high);
     }
}
```

MERGESORT EXAMPLE

Mergesort does of merging single elements into sorted pairs.



MERGESORT

- Split array A[0..*n*-1] into about equal halves and make copies of each half in arrays B and C
- Sort arrays B and C recursively
- Merge sorted arrays B and C into array A as follows:
  - Repeat the following until no elements remain in one of the arrays:
    - compare the first elements in the remaining unprocessed portions of the arrays
    - copy the smaller of the two into A, while incrementing the index indicating the unprocessed portion of that array
  - Once all elements in one of the arrays are processed, copy the remaining unprocessed elements from the other array into A.

PSEUDOCODE OF MERGE

```
Algorithm Merge(A,low,mid,high)
//A[low:high] is a global array containing two sorted subsets in A[low:mid]   //and in
    A[mid+1:high].
//The goal is to merge these two sets into a single set
//residing in A[low:high].        B[ ] is an auxillary global array.
{           h:=low; i:=low; j:=mid+1;
            While((h<=mid) and (j<=high)) do
            {           if(A[h]<=A[j]) then
                  {           B[i]:=A[h];  h:=h+1;
                  }
                  else
                  {                 B[i]:=A[j];  j:=j+1;
                  }
                  i:=i+1;
            }
```

PSEUDOCODE OF MERGE

```
        if(h>mid) then
            for k:=j to high do
            {     B[i]:=A[k];   i:=i+1;
            }
        else
            for k:=h to mid do
            {  B[i]:=A[k];   i:=i+1;
            }
            for k:=low to high do A[k]:=B[k];
}
```