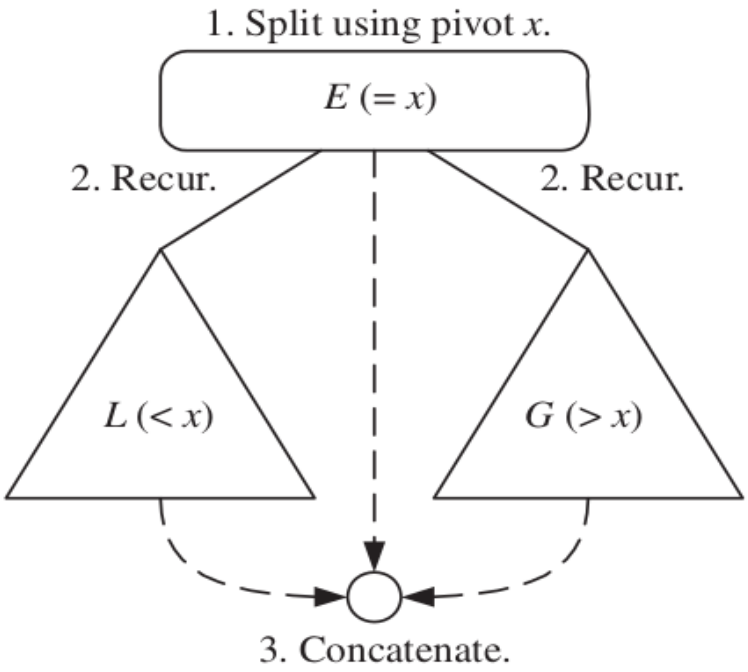
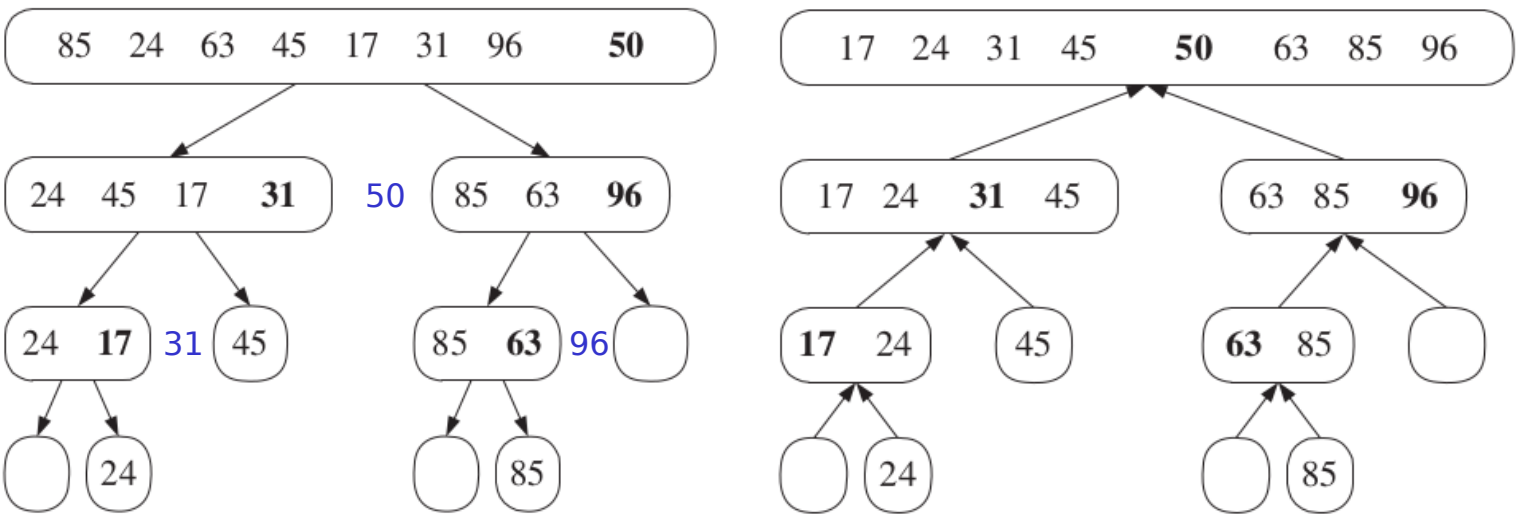
QUICK SORT - Steps:

1. ***Divide:*** If $S$ has at least two elements (nothing needs to be done if $S$ has zero or one element), select a specific element $x$ from $S$, which is called the ***pivot***. As is common practice, choose the pivot $x$ to be the last element in $S$. Remove all the elements from $S$ and put them into three sequences:

   - $L$, storing the elements in $S$ less than $x$
   - $E$, storing the elements in $S$ equal to $x$
   - $G$, storing the elements in $S$ greater than $x$.

   (If the elements of $S$ are all distinct, $E$ holds just one element—the pivot.)
2. ***Recur:*** Recursively sort sequences $L$ and $G$.
3. ***Conquer:*** Put the elements back into $S$ in order by first inserting the elements of $L$, then those of $E$, and finally those of $G$.



{85 24 63 45 50 31 96 17}

L={}  E={17}  G={85 24 63 45 50 31 96}

{85 24 63 45 17 31 50 96}

L={85 24 63 45 31 50}  E={96}  G={}

---

```
partition (arr[], low, high)
{
   pivot = arr[high];
   i = (low - 1)
   for (j = low; j < high; j++)
   {
      // If current element is smaller than the pivot
      if (arr[j] < pivot)
      {
         i++;    // increment index of smaller element
         swap arr[i] and arr[j]
      }
   }
   swap arr[i+1] and arr[high])
   return (i + 1)
}
```

Initial:  85 24 63 45 17 31 96 50
low = 0,  high = 7

Pivot: arr[7] = 50

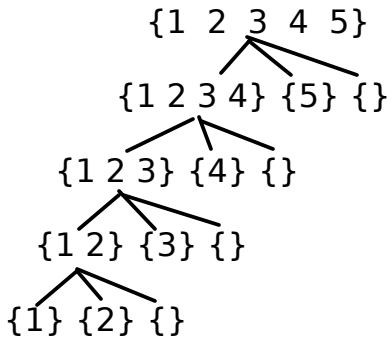
i = 3
j = 6

85 24 63 45 17 31 96 50
24 85 63 45 17 31 96 50
24 45 63 85 17 31 96 50
24 45 17 85 63 31 96 50
24 45 17 31 63 85 96 50
24 45 17 31 50 85 96 63

24 45 17 31 | 50 | 85 96 63

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
   if (low < high)    // i.e. if arr[] is of length >= 2
   {
      pi = partition(arr, low, high);
      /* pi is partitioning index, i.e. index of pivot */

      quickSort(arr, low, pi - 1);    // Before pi, i.e. L
      quickSort(arr, pi + 1, high); // After pi, i.e. G
   }
}
```




```
partition (arr[], low, high)
{
   pivot = arr[high];
   i = low
   for (j = low; j < high; j++)
   {
      // If current element is smaller than the pivot
      if (arr[j] < pivot)
      {
         swap arr[i] and arr[j]
         i++;    // increment index of smaller element
      }
   }
   swap arr[i] and arr[high])
   return (i)
}
```

Worst case time compexity:

(n-1) levels, and
O(n) operations at each level
(counted over all partition functions)
= O(n^2)

Best case time complexity:

O(log(n)) levels, and
O(n) operations at each level
(counted over all partition functions)
= O(n log(n))

Average case time complexity:
O(n log(n))