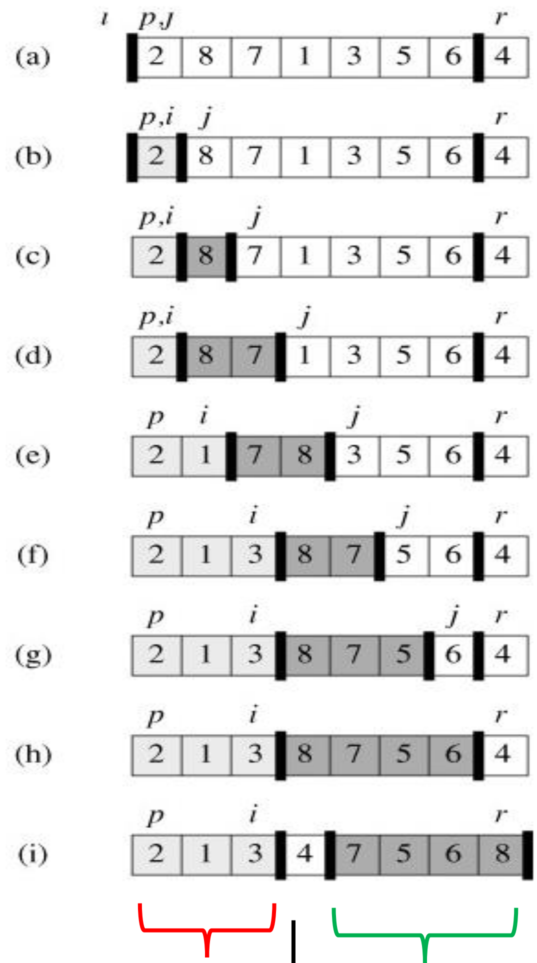
Operation of PARTITION on an 8-element array.



first partition with values <= pivot

second partition with values > pivot

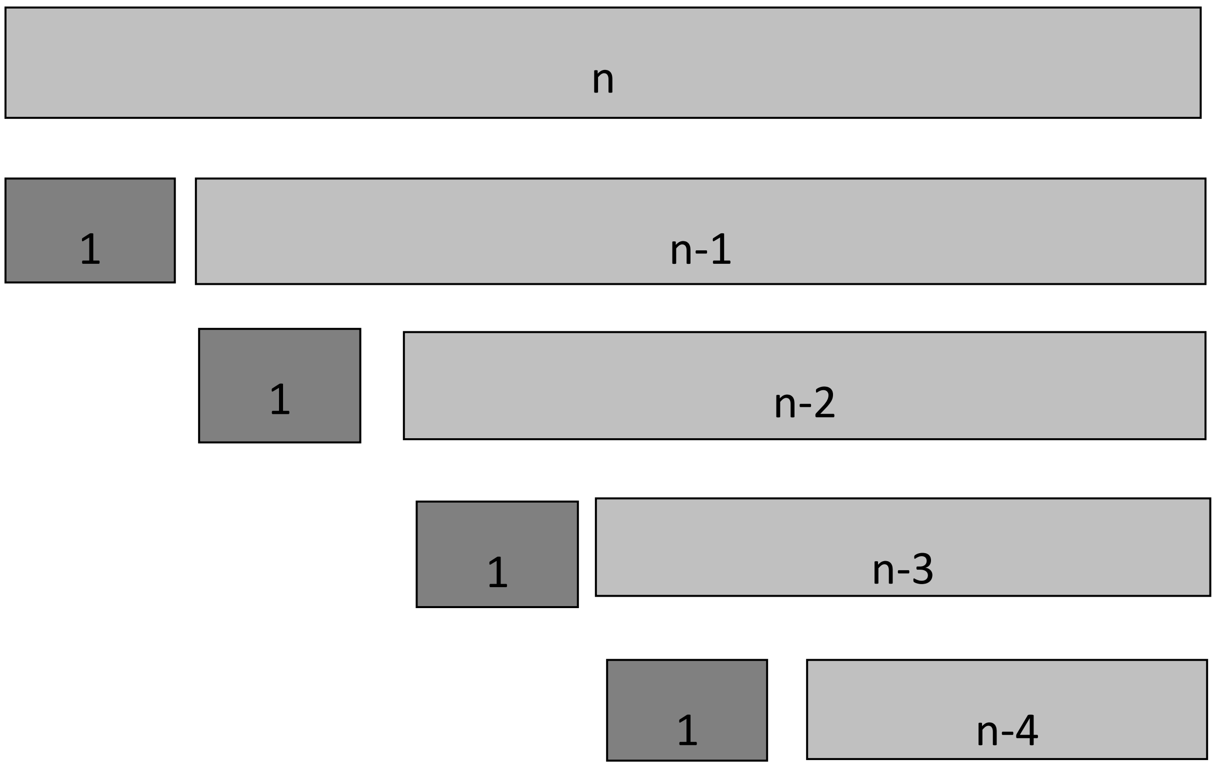
white element - pivot.

Analysis of Quick sort

The running time of quick sort depends on the partitioning of the sub arrays:

(a) Worst case partitioning (Unbalanced partitioning)

- Worst case occurs when the sub arrays are completely unbalanced. i.e. 0 elements in one sub array and $n - 1$ elements in the other sub array



Analysis of Quick sort

(a) Worst case partitioning (Repeated Substituted method)

- Partitioning → $\Theta(n)$
- Recursive call on an array of size 0 → $T(0) = \Theta(1)$
- Recursive call on an array of size (n-1) → $T(n-1)$
- Therefore **Recurrence** Equation is
- $T(n) = T(n-1) + T(0) + \Theta(n)$
- $= T(n-1) + \Theta(n)$
-
- $= T(n-2) + \Theta(n-1) + \Theta(n)$
- ………………….
- $= T(0) + \Theta(1) + \Theta(2) + \ldots. + \Theta(n-1) + \Theta(n)$
- $= \sum_{k=1}^{n} (\Theta(k)) = \Theta \sum_{k=1}^{n} k = \Theta(n^2)$
- Worst case Running Time is $\Theta(n^2)$

Analysis of Quick sort

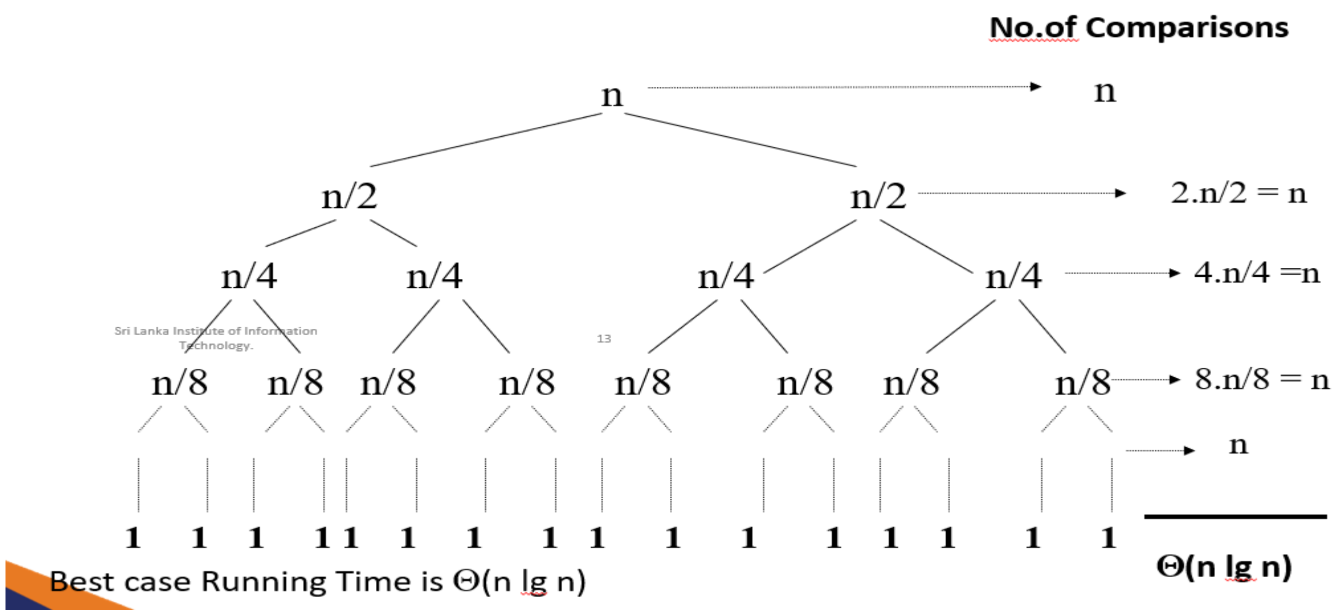
(b)) Best case partitioning

Best case occurs when PARTITION produces two sub arrays , one is of size (n-1)/2 and the other is of size (n-1)/2. In this case, quicksort runs much faster.

Recurrence equation is

$$T(n) = 2T(n/2) + \Theta(n)$$

Best Case: **Analysis of Quick Sort (with recursion tree)**



Merge sort

Merge Sort is a sorting algorithm based on divide and conquer.

Its worst-case running time has a lower order of growth than insertion sort.

The merge sort algorithm closely follows the divide-and-conquer paradigm. Intuitively, it operates as follows.

- Divide: Divide the *n*-element sequence to be sorted into two subsequences of *n*/2 elements each.
- Conquer: Sort the two subsequences recursively using merge sort.
- Combine: Merge the two sorted subsequences to produce the sorted answer.

- **Divide** by splitting into two subarrays $A[p \, . \, . \, q]$ and $A[q + 1 \, . \, . \, r]$, where q is the halfway point of $A[p \, . \, . \, r]$.
- **Conquer** by recursively sorting the two subarrays $A[p \, . \, . \, q]$ and $A[q + 1 \, . \, . \, r]$.
- **Combine by** merging the two sorted subarrays $A[p \, . \, . \, q]$ and $A[q + 1 \, . \, . \, r]$ to produce a single sorted subarray $A[p \, . \, . \, r]$.

  To accomplish this step, we'll define a procedure MERGE*(A, p, q, r)*.

  Merge sort procedure

  Input : A an array in the range 1 to n**.**

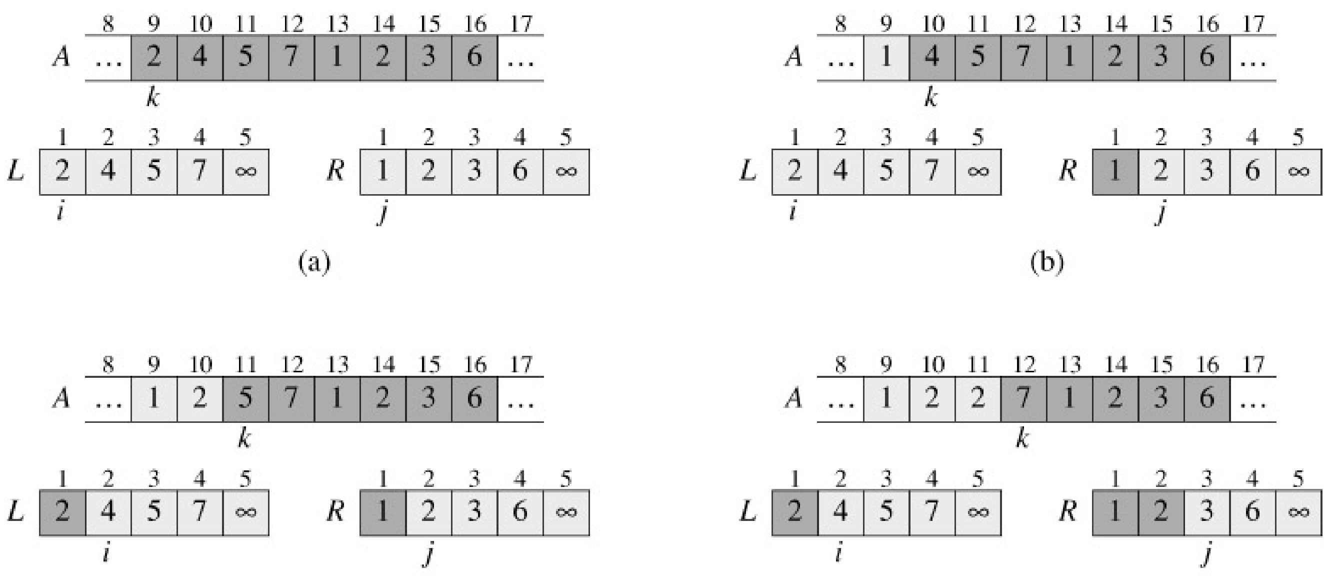
  Output : Sorted array A.

```
MERGESORT (A, p, r)
if p < r
 q = ⌊(p+r )/2⌋
      MERGESORT (A, p, q)
      MERGESORT (A, q+1, r)
   MERGE (A, p, q, r)
```

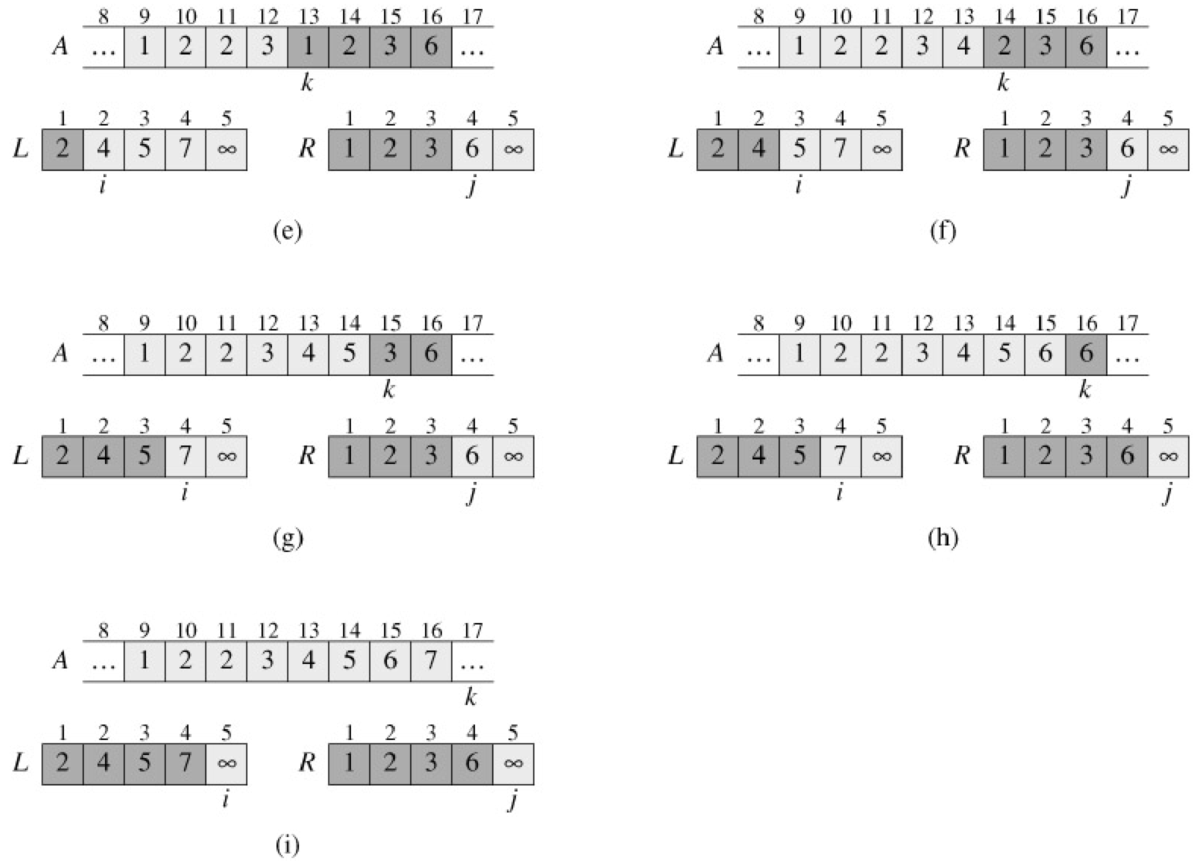
Merge procedure

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
create arrays L[1.. n1 + 1] and R[1.. n2 + 1]
for i = 1 to n1
        L[i] = A[p + i - 1]
for j = 1 to n2
        R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
        if L[i] ≤ R[j]
              A[k] = L[i]
              i = i + 1
        else  A[k] = R[j]
              j = j + 1
```


Illustration when the subarray $A[9..16]$ contains the sequence 〈2, 4, 5, 7, 1, 2, 3, 6〉



(a) (b) (c) (d)



Analysis of Merge Sort

- To find the middle of the sub array will take $\Theta(1)$.
- To recursively solve each sub problem will take $2T(n/2)$.
- To combine sub arrays will take $\Theta(n)$.

Therefore $\mathbf{T(n)=2T(n/2)+\Theta(n)+\Theta(1)}$

We can ignore $\Theta(1)$ term.

$$\mathbf{T(n)=2T(n/2)+\Theta(n)}$$

Analysis of Merge Sort

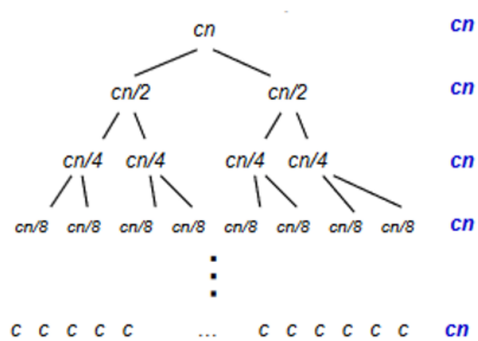
$$T(n) = 2T(n/2) + cn$$
$$2T(n/2) = 4T(n/4) + 2cn/2$$
$$4T(n/4) = 8T(n/8) + 4cn/4$$

-

-

-

$$T(2) = nT(1) + (n/2)\, c*2$$



add and cancel:

$$T(n) = nT(1) + cn + cn + \ldots + cn$$
$$= nT(1) + cn * \log_2 n = \Theta(n\log n)$$

Summary.

- Divide and conquer method.
- Quicksort algorithm.
- Quicksort analysis.
- Mergesort algorithm.
- Mergesort analysis.