- Case → Best: O(1)
  → Average: O(log n)
  → Worst: O(log n)

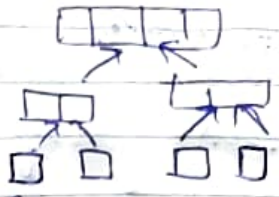
Binary Search

```
1. Def. binary Search (A,x):
2.   n = len(A)
3.   beg = 0
4.   end = n-1
5.   result = -1
6.   While (big <= end):
7.     mid = (big+end)/2
8.     If (A[mid] <= x):
9.       beg = mid+1
10.        result = mid
11.    Else:
12.      end = mid-1
13.    Return result
```

Shivam Singh
2K21/CO/446

- Case → Best = $O(n \log n)$
  → Average = $O(n \log n)$
  → Worst = $O(n \log n)$

Merge Sort



1. Start
2. Declare array & left, right, mid variable
3. Perform merge function
   if left > right
   return
   mid = (left + right)/2
   mergesort (array, left, mid)
   merge sort (array, mid+1, right)
   merge (array, left, mid, right)
4. Stop

   Merge Sort (arr[], l, r)
5. If r > l
   → find middle point to divide array into two halves:
   • middle $m = l + (r-l)/2$
   → Call merge sort for first half:
   • Call merge sort (arr, l, m)
   → Call merge sort for second half:
   • Call merge sort (arr, m+1, r)
   → Merge the two halves sorted in step 2 & 3.
   • Call merge (arr, l, m, r)

Shivam Singh
2K21/CO/446

- Case
  - Best : $O(n \log n)$
  - Average : $O(n \log n)$
  - Worst : $O(n^2)$

Quick Sort

```
QUICKSORT (array A, start, end)
1.  {
2.  if (start < end)
3.  {
4.  p = partition (A, start, end)
5.  QUICKSORT (A, start, p-1)
6.  QUICKSORT (A, p+1, end)
7.  }
8.  }
```

Partition Algorithm :

```
Partition (~~int~~ array A, start, end)
   {
1.  pivot ? A[end]
2.    i ? start - 1
3.  for j ? start to end - 1 {
4.  do if (A[j] < pivot) {
5.  then i ? i+1
6.  swap A[i] with A[j]
7.  }}
8.  swap A[i+1] with A[end]
9.  return i+1
10. }
```

Shivam Singh
2K21/CO/446