Objectives

- Divide and Conquer Strategy
- Design Merge Sort Algorithm
- Analyse time and space complexity of Merge Sort
- Design Quick Sort Algorithm
- Analyse time and space complexity of Quick Sort
- Binary Search using Iterative and Recursion

Divide and Conquer Approach

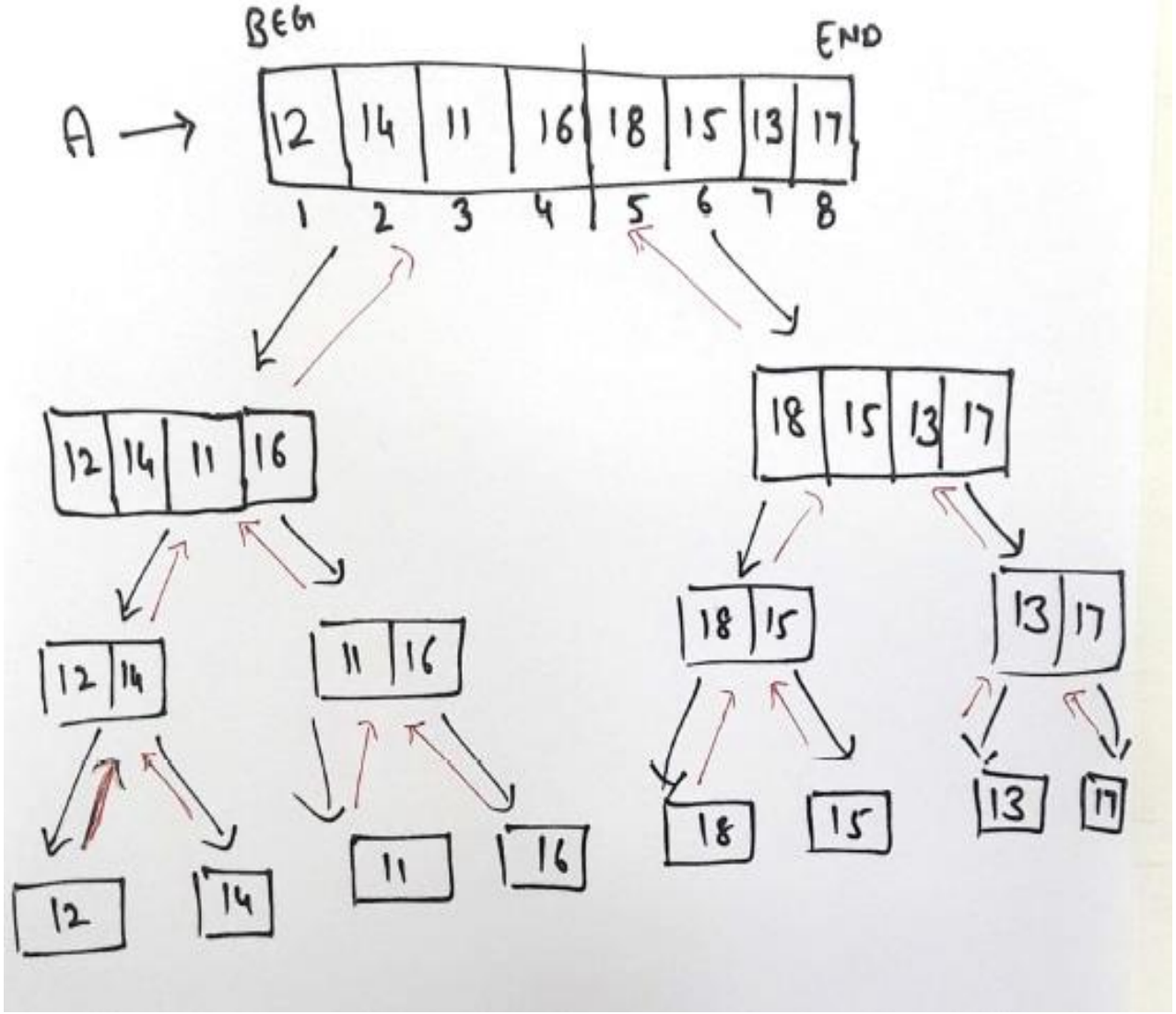
Divide: the main problem into a number of subproblems that are smaller instances of the same problem.

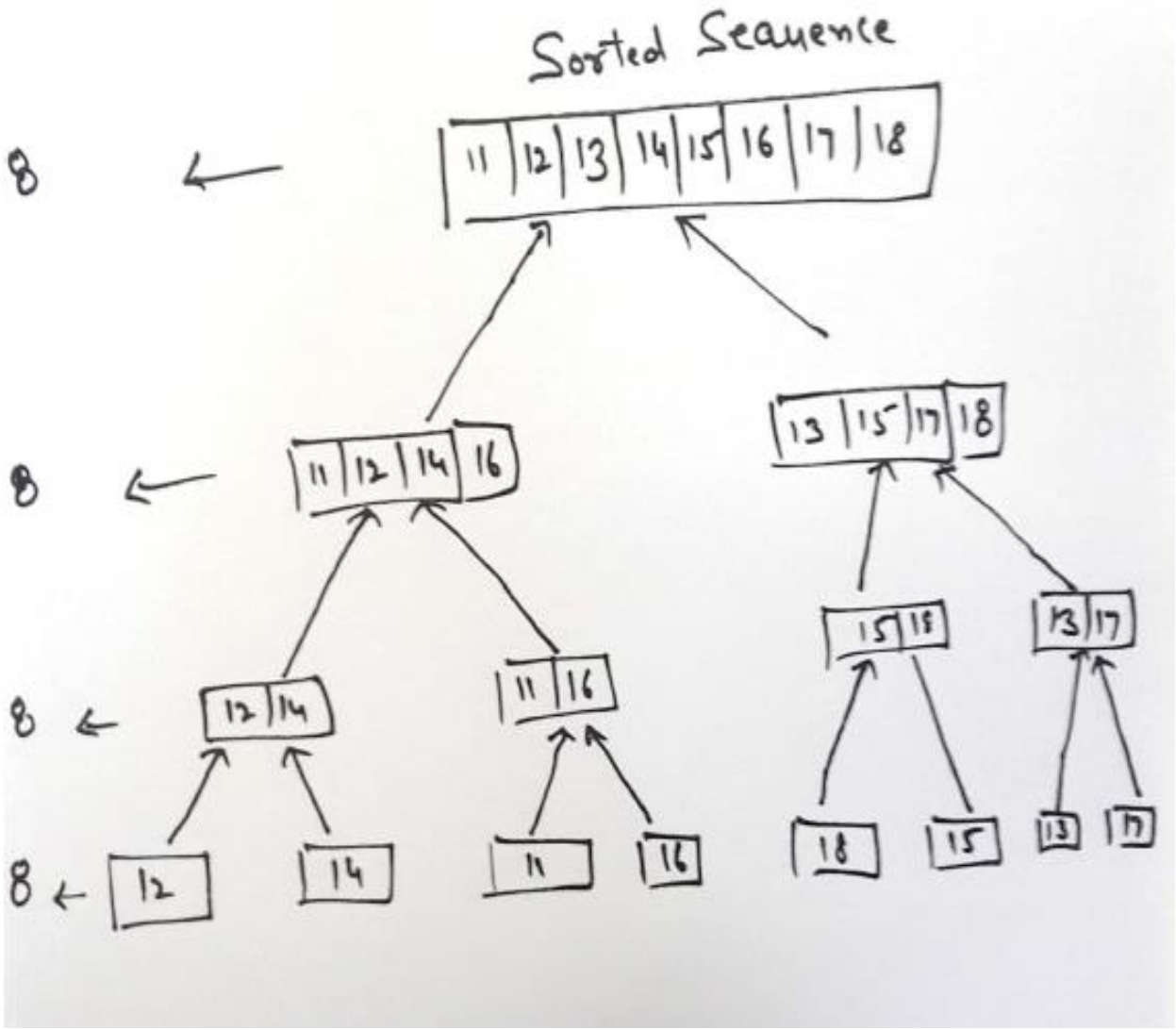
Conquer: solve the subproblems recursively. If the size of subproblems is small enough, solve them directly.

Combine: the solutions to the subproblems into the solution for the original problem.

Merge Sort follows the divide and conquer paradigm.

1. Divide the n elements sequence to be sorted into two subsequences of $\frac{n}{2}$ elements each.
2. Sort the two subsequences recursively using merge sort.
3. Merge the two sorted subsequences to produce the sorted array.


BEG
END
A
12 14 11 16 18 15 13 17
1 2 3 4 5 6 7 8
12 14 11 16
18 15 13 17
12 14
11 16
18 15
13 17
12
14
11
16
18
15
13
17


Sorted Seamence
11 12 13 14 15 16 17 18
11 12 14 16
13 15 17 18
12 14
11 16
15 18
13 17
12
14
11
16
18
15
13
17

$$Algorithm\ MergeSort(A, BEG, END) \text{-----} T(n)$$

$$\{$$

$$if(BEG < END)$$

$$\{$$

$$MID = \lfloor (BEG + END)/2 \rfloor; \text{----------} (1)$$

$$MergeSort(A, BEG, MID); \text{---------} T(\frac{n}{2})$$

$$MergeSort(A, MID + 1, END); \text{---------} T(\frac{n}{2})$$

$$Merge(A, BEG, MID, END); \text{----------} (n)$$

$$\}$$

$$\}$$

Merging process

```
Algo Merge(A, BEG, MID, END)
{
n1 = MID − BEG + 1;
n2 = END − MID;
for(i = 1; i <= n1; i++)
{L[i] = A[BEG + i − 1];}
for(j = 1; i <= n2; j++)
{R[j] = A[MID + j];}
L[n1 + 1] = ∞;
R[n2 + 1] = ∞;
 i = 1; j = 1;
 for(k = BEG, k <= END; k++)
 {
 if  L[i] <= R[j]
 {   A[k] = L[i];  i = i + 1;
 }
 else
 { A[k] = R[j]; j = j + 1;
 }
 }
```

3. Analysis of Merge Sort:

$$Algorithm\ MergeSort(A, BEG, END) \text{-----} T(n)$$

$$\{$$

$$if(BEG < END)$$

$$\{$$

$$MID = \lfloor (BEG + END)/2 \rfloor; \text{----------} (1)$$

$$MergeSort(A, BEG, MID); \text{--------} T(\frac{n}{2})$$

$$MergeSort(A, MID+1, END); \text{--------} T(\frac{n}{2})$$

$$Merge(A, BEG, MID, END); \text{----------} (n)$$

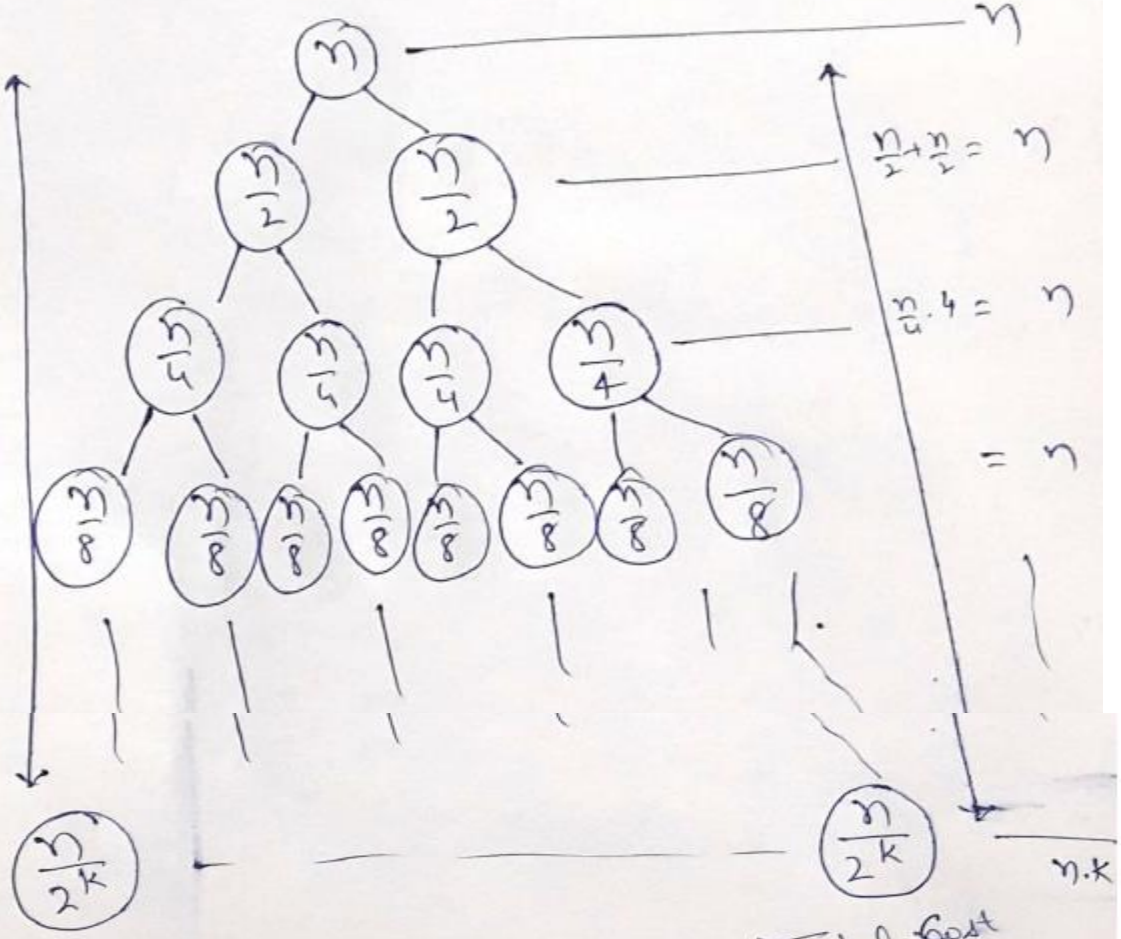
$$\}$$

$$\}$$

The recurrence relation is:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(\frac{n}{2}) + n & n > 1 \end{cases}$$

$T(n) = 2T\left(\frac{n}{2}\right) + n$



Total Cost

$\frac{n}{2^k} = 1$

$k = \log_2 n$

So $T(n) = \Theta(n \log n)$

Pros and cons of Merge Sort

1. Large size data: handles millions of no. or records
2. Linked list: merge sort is suitable for linked list. We can merge two linked list without creating third linked list.
3. External sorting: perform sorting in chunks or pieces of data.
4. Stable: after sorting the order of duplicate records are maintained. Eg. Arrangement of records.

Cons:

1. Extra space for sorting: also called not in place sorting. Results are placed in extra array. We need some extra array.
2. No small problem: when size of input data is small merge sort is relatively slow and we use insertion sort. Since merge sort uses recursion. And recursion used stack. Maximum size of stack depends on height of the tree i.e. $\log n$ stack space it requires along with extra space of size $n + \log n \approx \theta(n)$.

QUICK SORT:

1. Like Merge sort, Quick sort also follows Divide and Conquer paradigm.
2. In place sorting algorithm

Divide: partition the array $A[BEG,\ldots,END]$ into two subarrays $A[BEG,\ldots,idx-1]$ and $A[idx+1,\ldots,END]$ such that each element of $A[BEG,\ldots,END]$ is less than or equal to $A[idx]$ and every element of $A[idx+1,\ldots,END]$ is greater than $A[idx]$. The index idx is obtained from partitioning procedure.

Conquer: sort the two subarrays $A[BEG,\ldots,idx-1]$ and $A[idx+1,\ldots,END]$ by recursive calls to quick sort.

Combine: since the subarrays are already sorted, so entire array $A[BEG,\ldots,END]$ is now sorted.

QUICKSORT(A, p, r)

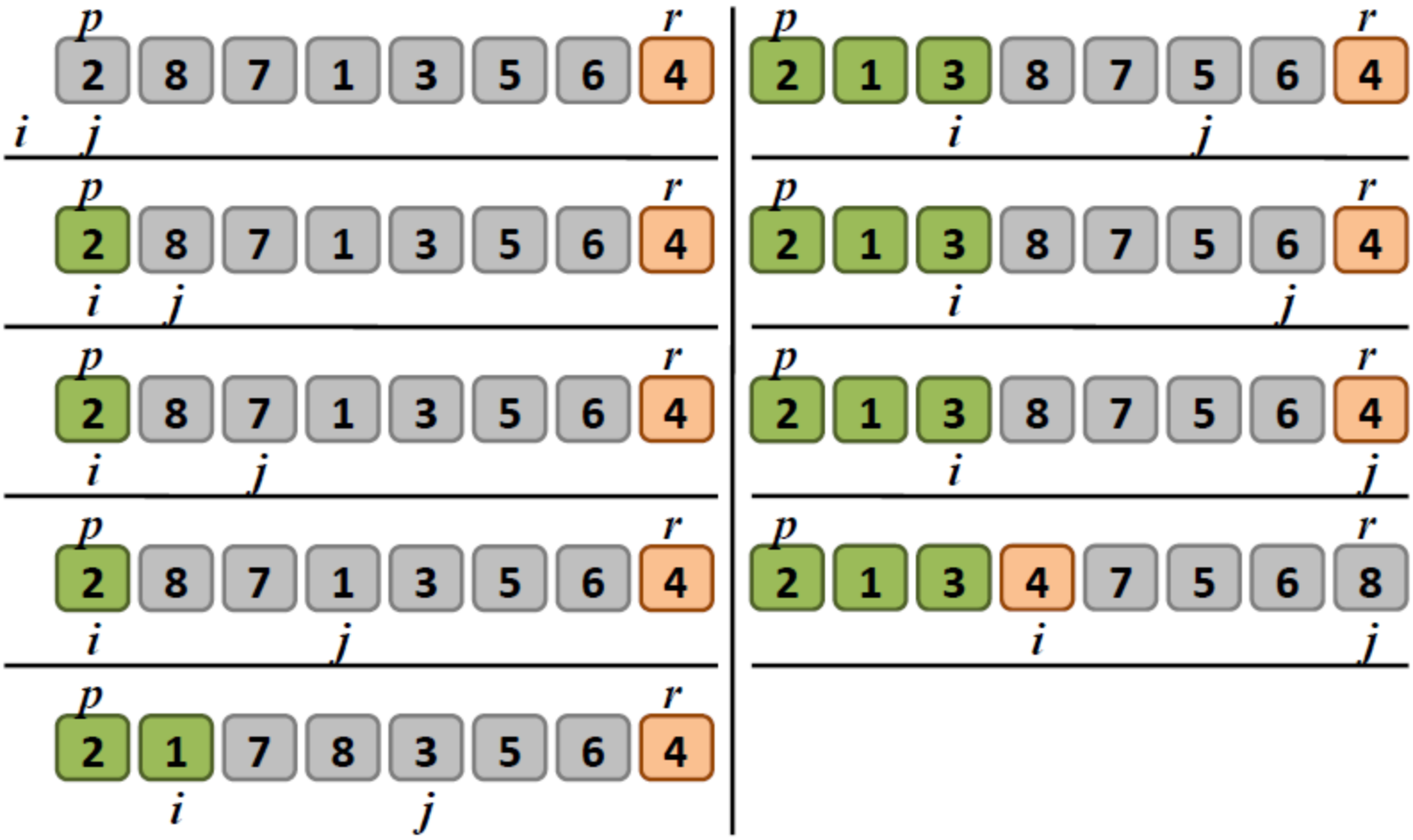
- QUICKSORT(A, p, r)

1. if $p < r$
2. q = PARTITION(A, p, r)
3. QUICKSORT($A, p, q - 1$)
4. QUICKSORT($A, q + 1, r$)

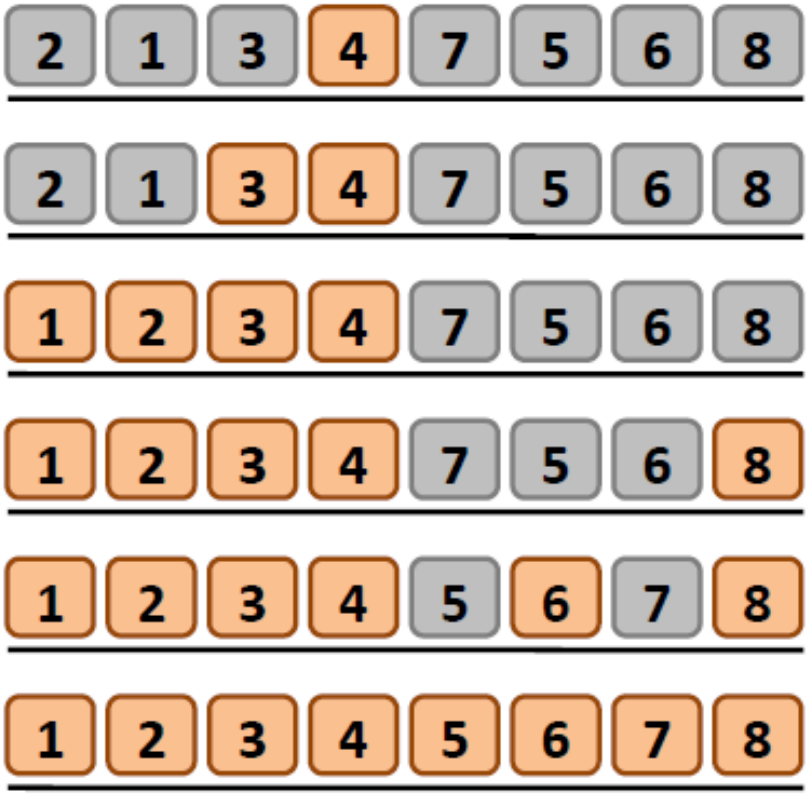
- To sort an array A with n elements, the first call to QUICKSORT is made with $p = 0$ and $r = n - 1$.

1. PARTITION(A, p, r)
2. $x = A[r]$
3. $i = p - 1$
4. for $j = p$ to $r - 1$
5. if $A[j] \leq x$
6. $i = i + 1$
7. Exchange $A[i]$ with $A[j]$
8. Exchange $A[i + 1]$ with $A[r]$
9. return $i + 1$

Example: 2, 8, 7, 1, 3, 5, 6, 4



Contd...



Analysis of Quick sort:

Best-case: Suppose that the partitioning is done at the middle that is the two subproblems of size not more than $\frac{n}{2}$. one is of size $\left\lfloor \frac{n}{2} \right\rfloor$ and other is of size $\left\lceil \frac{n}{2} \right\rceil - 1$. So the recurrence relation becomes $T(n) = 2T(\frac{n}{2}) + \theta(n)$, so best case time complexity:

$$T(n) = \theta(n \log n).$$

Worst case: if we have list in the sorted order, then partitioning produces one subproblems with $n-1$ elements and one with zero element. And assume that this partitioning arises in each recursive calls, so partitioning cost $\theta(n)$ time. The recurrence relation for running time is: $T(n)=T(n-1)+\theta(n)$ which implies that worst case complexity is: $T(n)=\theta(n^2)$.

Space Complexity: Quick sort is an inplace algorithm. That is it does not use extra space. It is a recursive algorithm, it uses stack. So the maximum size of stack in worst case could be n and minimum size of stack be equal to height of tree $\log n$. So space complexity varies from $\log n$ to n.

Binary Search:

- Divide and Conquer technique that is if problem is large then divide into sub problems and once each sub problem is solved, combine them to obtain the solution of main problem.

Consider a set of elements:

$$A \rightarrow \begin{bmatrix} \overset{1}{13} & \overset{2}{16} & \overset{3}{18} & \overset{4}{22} & \overset{5}{24} & \overset{6}{27} & \overset{7}{35} & \overset{8}{39} & \overset{9}{41} & \overset{10}{46} & \overset{11}{52} & \overset{12}{57} & \overset{13}{63} & \overset{14}{65} & \overset{15}{72} \end{bmatrix}$$

$$key = 52$$

Consider a set of elements:

$$A \rightarrow \begin{bmatrix} \overset{1}{13} & \overset{2}{16} & \overset{3}{18} & \overset{4}{22} & \overset{5}{24} & \overset{6}{27} & \overset{7}{35} & \overset{8}{39} & \overset{9}{41} & \overset{10}{46} & \overset{11}{52} & \overset{12}{57} & \overset{13}{63} & \overset{14}{65} & \overset{15}{72} \end{bmatrix}$$

$key = 52$

BEG	END	$MID = \left\lfloor \frac{BEG + END}{2} \right\rfloor$	
1	15	8	$key > A[MID], BEG = MID + 1$
9	15	12	
9	11	10	$key < A[MID], END = MID - 1$
11	11	11	

After 4 comparisons, key is found.

Algorithm:

$\text{int} \quad Binary\_Search(A, n, key)$

$\{ \quad BEG = 1, END = n;$

$while(BEG < END)$

$\{$

$MID = \left\lfloor \frac{BEG + END}{2} \right\rfloor$

$if(key == A[MID])$

$return \quad MID;$

$if(key < A[MID])$

$\{$

$END = MID - 1;\}$

$else\{$

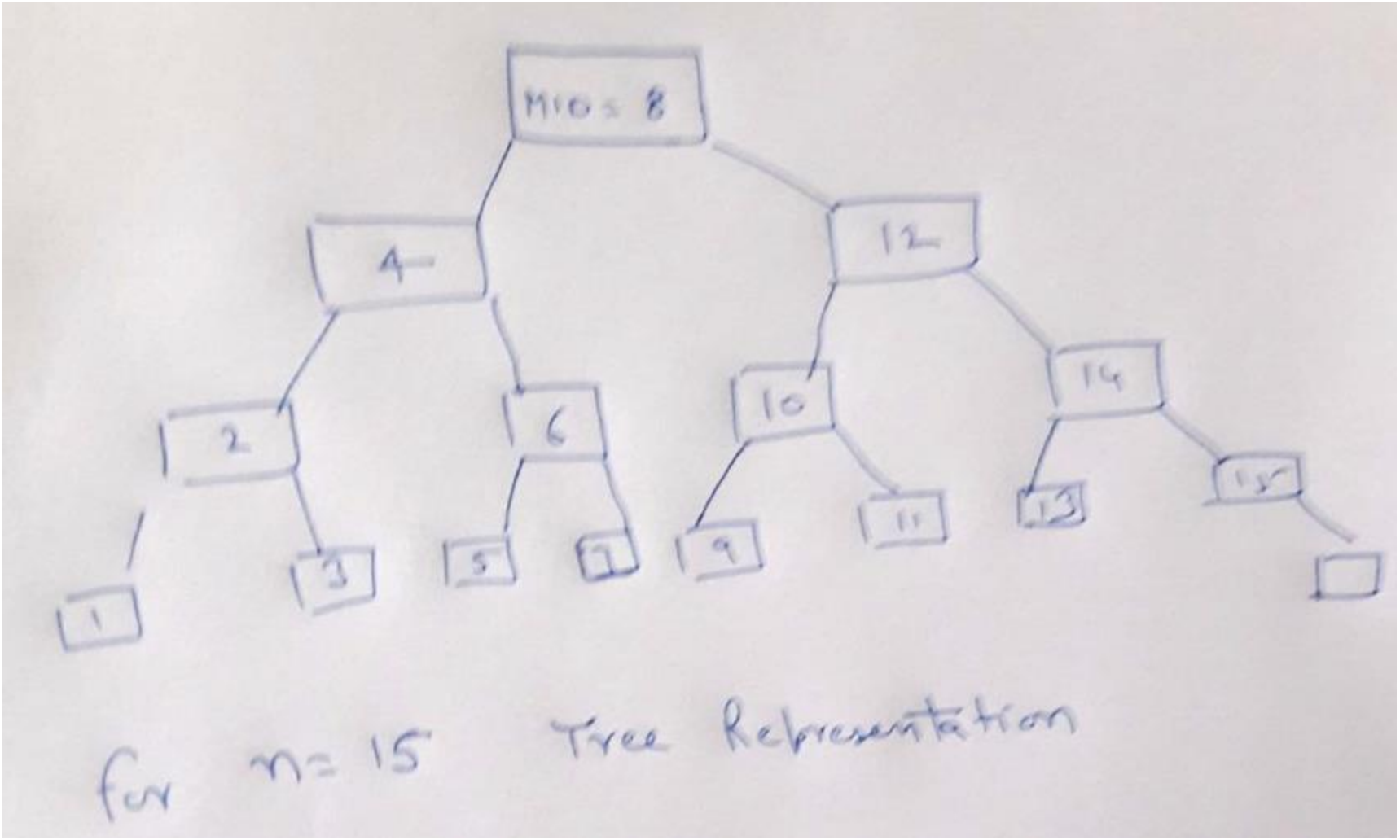
$BEG = MID + 1;$

$\}$

$\}$

$return \ 0;$

$\}$



for n = 15 Tree Representation

Time taken:

$time\ complexity : \log n$

$Min.\ time : O(1)$

$MaxTime : O(\log n)$

$for\ \ unsuccessful\ \ search : O(\log n)$

Which is the analysis of binary search?

$Algo\ BS\_recursive(A, BEG, END, Key)$

{

$if\ (BEG == END)$

{

$if\ (A[BEG] == key)$

$\{ return\ BEG; \}$

$else\ \{ return\ 0; \}$

$else$

{

$mid = \left\lfloor \frac{BEG + END}{2} \right\rfloor$

$if\ (key == A[MID]$

$return\ MID;$

$if\ (key < A[MID])$

$return\ BS\_recursive(BEG, MID - 1, Key);$

$else$

$return\ BS\_recursive(MID + 1, END, Key);$

}

}

Divide and Conquer follow recursive approach and its time complexity is evaluated using recurrence relation. The recurrence relation of binary search using recursion is:

$$T(n) = \begin{cases} 1 & if\ n = 1 \\ T\left(\frac{n}{2}\right) + 1 & n > 1 \end{cases}$$

So by Master theorem, time complexity: $O(\log n)$

THANKS !