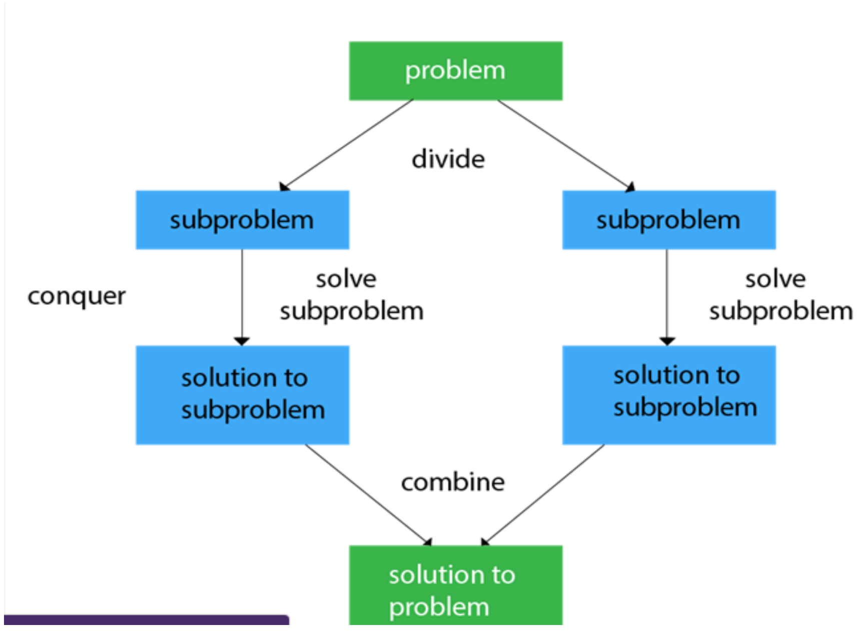
1)Explain basic concepts of divide and conquer.

Divide And Conquer
This technique can be divided into the following three parts:

1. **Divide:** This involves dividing the problem into smaller sub-problems.
2. **Conquer:** Solve sub-problems by calling recursively until solved.
3. **Combine:** Combine the sub-problems to get the final solution of the whole problem.

Divide And Conquer algorithm :

```
DAC(a, i, j)
{
    if(small(a, i, j))
      return(Solution(a, i, j))
    else
      m = divide(a, i, j)               // f1(n)
      b = DAC(a, i, mid)                 // T(n/2)
      c = DAC(a, mid+1, j)            // T(n/2)
      d = combine(b, c)                 // f2(n)
   return(d)
}
```

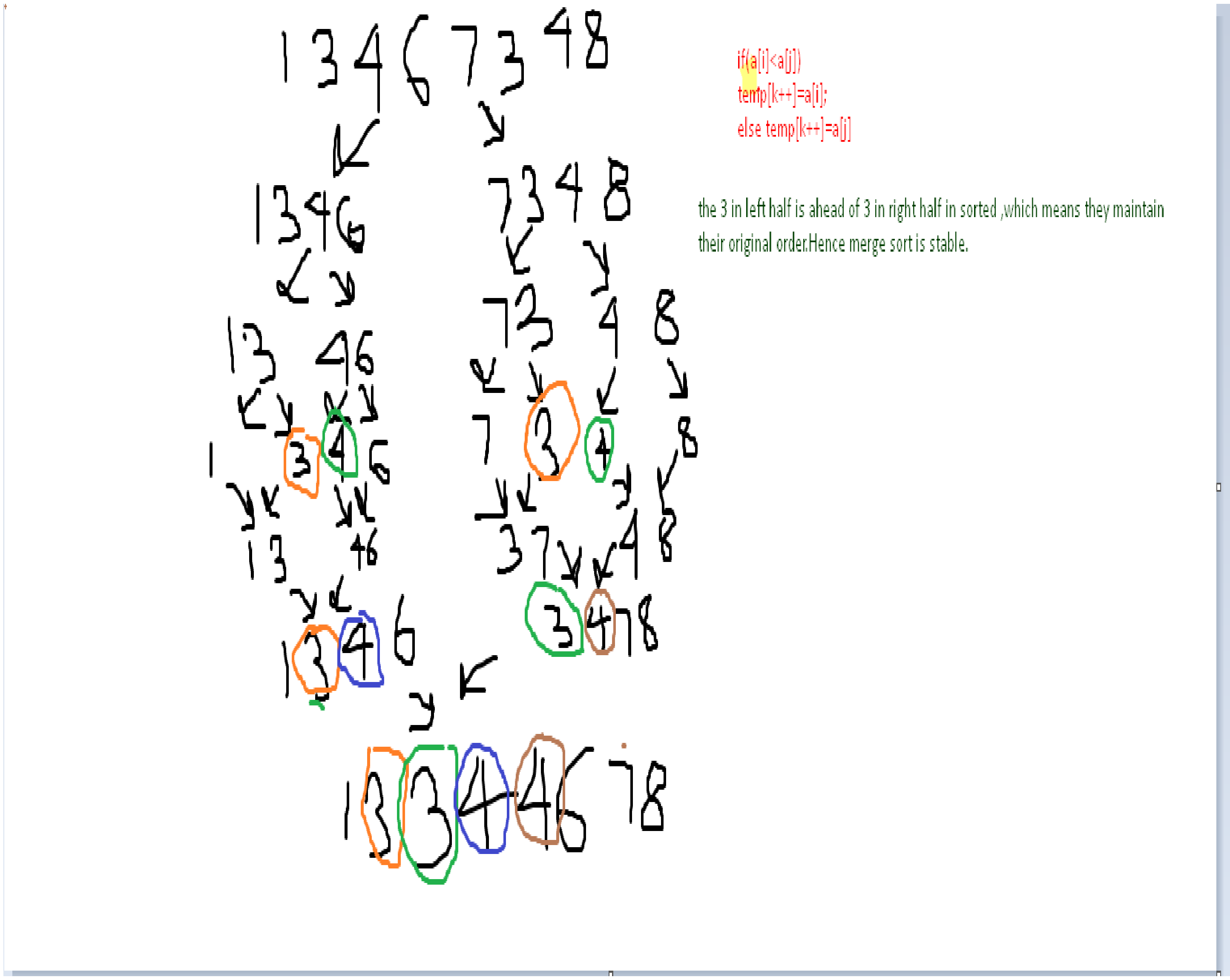


Applications of Divide and Conquer Approach:

Following algorithms are based on the concept of the Divide and Conquer Technique:

1. **Binary Search:** The binary search algorithm is a searching algorithm, which is also called a half-interval search or logarithmic search. It works by comparing the target value with the middle element existing in a sorted array. After making the comparison, if the value differs, then the half that cannot contain the target will eventually eliminate, followed by continuing the search on the other half. We will again consider the middle element and compare it with the target value. The process keeps on repeating until the target value is met. If we found the other half to be empty after ending the search, then it can be concluded that the target is not present in the array.
2. **Quicksort:** It is the most efficient sorting algorithm, which is also known as partition-exchange sort. It starts by selecting a pivot value from an array followed by dividing the rest of the array elements into two sub-arrays. The partition is made by comparing each of the elements with the pivot value. It compares whether the element holds a greater value or lesser value than the pivot and then sort the arrays recursively.

2)Is merge sort stable?
Explain with 1,3,4,6,7,3,4,8
Yes.
Merge Sort is a **stable sort** which means that the same element in an array maintain their original positions with respect to each other.



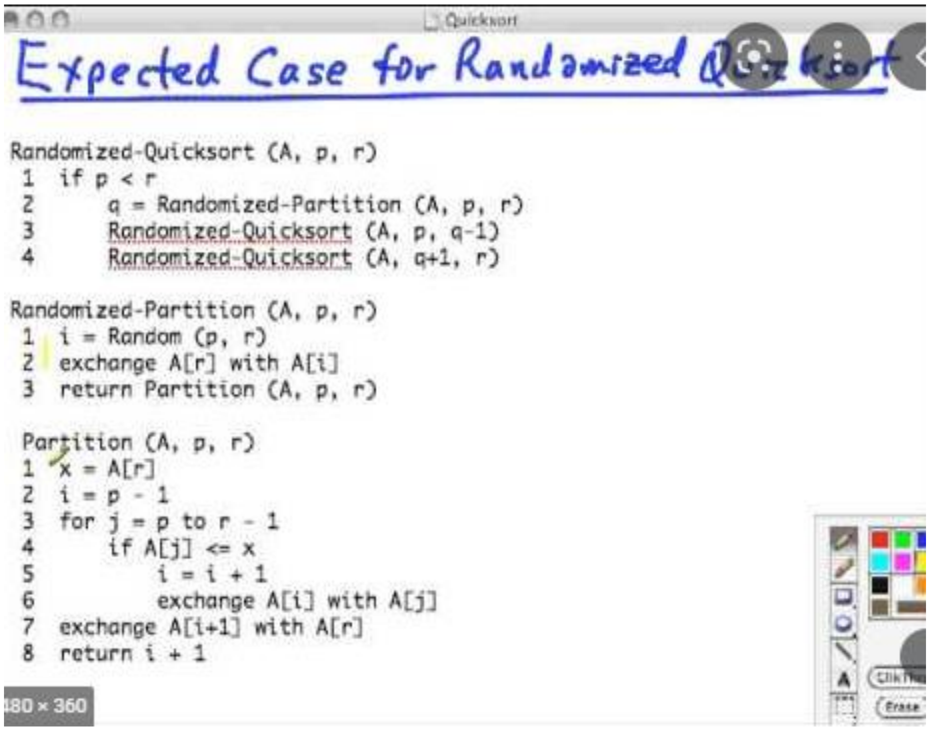
```
if(a[i]<a[j])
temp[k++]=a[i];
else temp[k++]=a[j]
```

the 3 in left half is ahead of 3 in right half in sorted ,which means they maintain their original order.Hence merge sort is stable.

3)Explain QuickSort using Random Pivoting(Randomized Quick Sort)

Using random pivoting we improve the expected or average time complexity to O (N log N). The Worst-Case complexity is still O (N^2).

We run the Quicksort algorithm. Each time picking a random element in the array as the pivot.

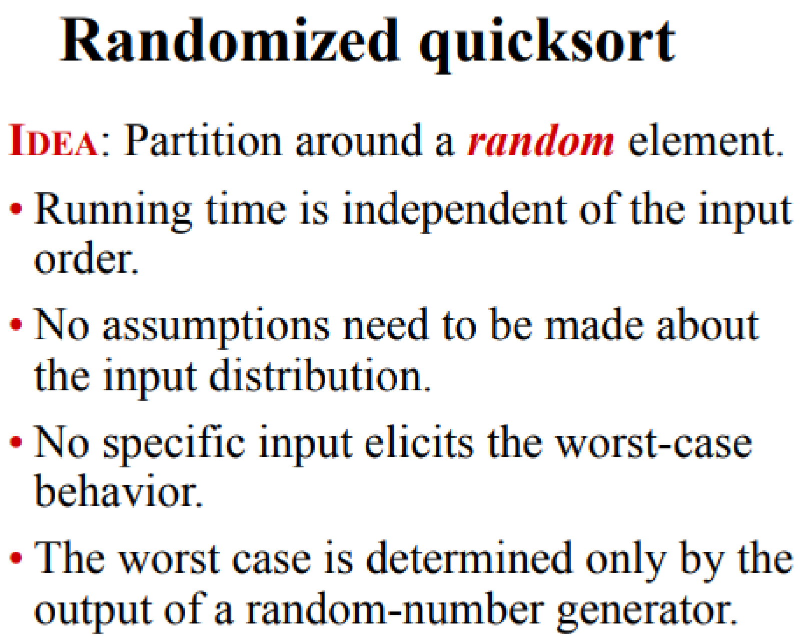


Expected Case for Randomized Quicksort

```
Randomized-Quicksort (A, p, r)
  if p < r
      q = Randomized-Partition (A, p, r)
      Randomized-Quicksort (A, p, q-1)
      Randomized-Quicksort (A, q+1, r)

Randomized-Partition (A, p, r)
  i = Random (p, r)
  exchange A[r] with A[i]
  return Partition (A, p, r)

Partition (A, p, r)
  x = A[r]
  i = p - 1
  for j = p to r - 1
      if A[j] <= x
          i = i + 1
          exchange A[i] with A[j]
  exchange A[i+1] with A[r]
  return i + 1
```



Randomized quicksort

IDEA: Partition around a *random* element.

- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.

4)What will be the position of the element in case of successful binary search in best case?

In the best case, where the target value is the middle element of the array, **its position is returned after one iteration**.

for eg . my sorted array is 3 , 5 , 6, 7, 9 , 11, 25 and the element to be searched is let's say 7 so the mid index is 3 , also arr[3]==element so binary search algorithm will converge here itself(in a single go) therefore the The Time Complexity will be O(1).