Advanced Algorithms & Data Structures

Department of CSE

ADVANCED ALGORITHMS AND DATA STRUCTURES
23CS03HF

Topic:

Quick Sort

AIM OF THE SESSION

To familiarize students with the concept of Quick Sort

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1.Demonstrate :- Quick sort algorithm.

Describe :- Quick sort time complexity and recurrence relation

LEARNING OUTCOMES

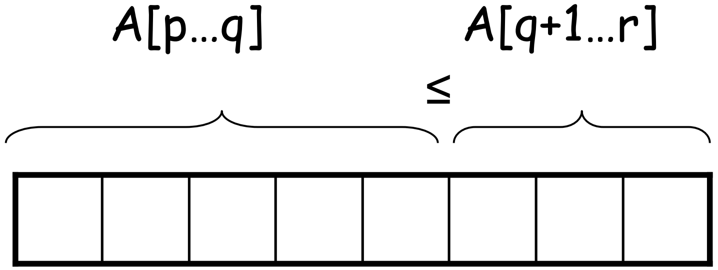
At the end of this session, you should be able to:

1. Define :- Quick sort .
2. Describe :- Quick sort time complexity and recurrence relation
3. Summarize:- Description about the quick sort and time complexity of quick sort

QUICK SORT

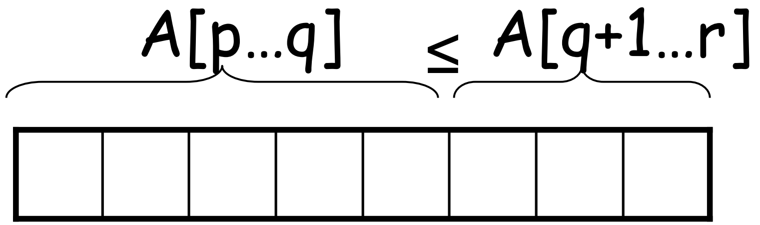
- QuickSort is a Divide and Conquer algorithm.
- It picks an element as pivot and partitions the given array around the picked pivot.
- There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.



- Sort an array A[p…r]

- **Divide**
  - Partition the array A into 2 subarrays A[p..q] and A[q+1..r], such that each element of A[p..q] is smaller than or equal to each element in A[q+1..r]
  - Need to find index q to partition the array



- **Conquer**
  - Recursively sort A[p..q] and A[q+1..r] using Quicksort
- **Combine**
  - Trivial: the arrays are sorted in place
  - No additional work is required to combine them
  - The entire array is now sorted

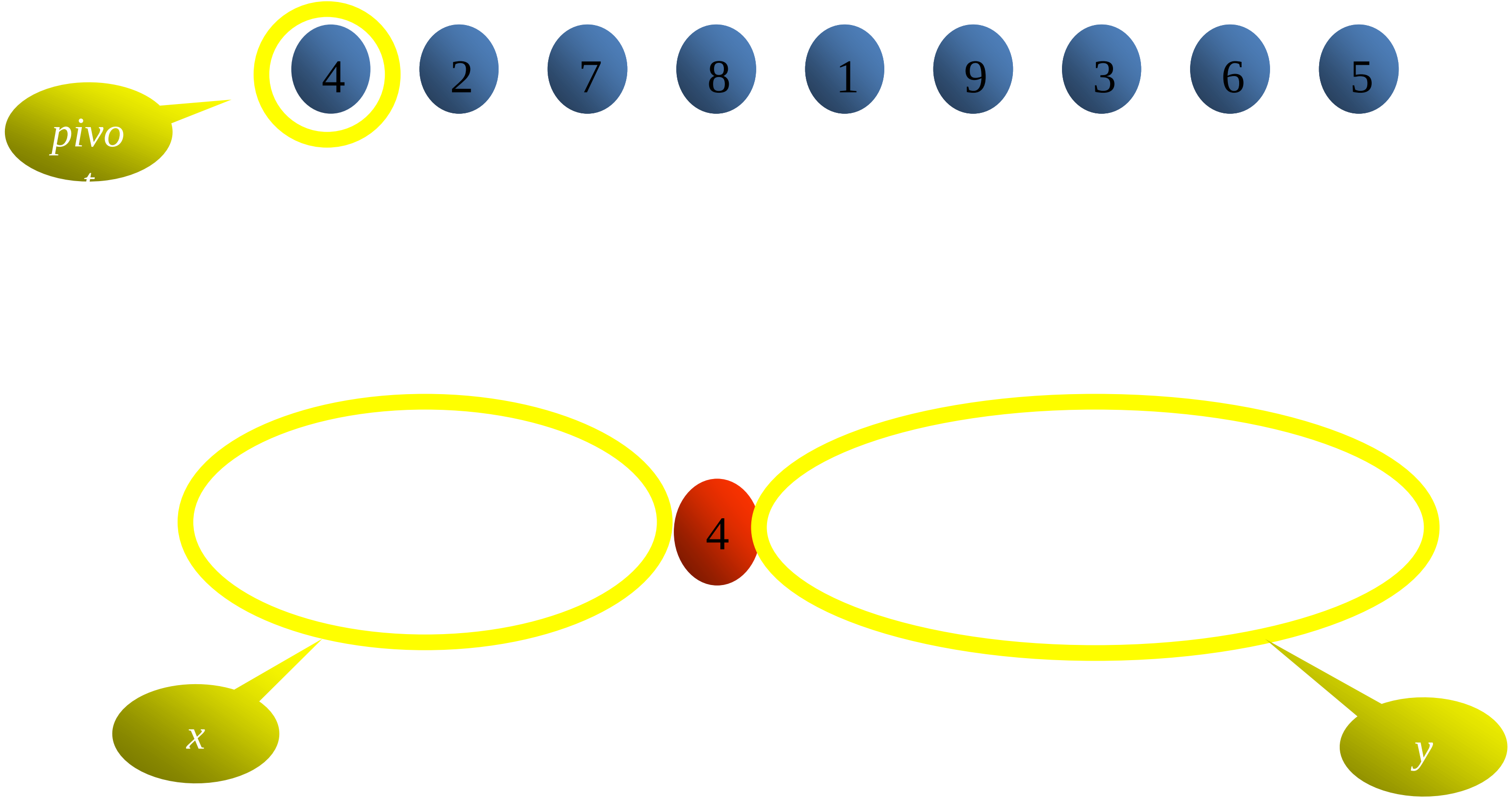
- **Divide**:
  - Pick any element as the **pivot**, e.g, the first element
  - Partition the remaining elements into

    FirstPart, which contains all elements < pivot

    SecondPart, which contains all elements > pivot

- **Recursively sort** FirstPart and SecondPart.
- **Combine**: no work is necessary since sorting is done in place.

pivot divides a into two sublists x and y.



Example

Keep going from left side as long as a[i]<pivot and from the right side as long as a[j]>pivot

pivot → 85   24   63   95   17   31   45
98

i   j

85   24   63   95   17   31   45
98

i   j

85   24   63   95   17   31   45
98

i   j

85   24   63   95   17   31   45
98

i   j

If i<j interchange i^{th} and j^{th} elements and then Continue the process.

85	24	63	45	17	31	95	98
			i			j	

85	24	63	45	17	31	95	98
				i		j	

85	24	63	45	17	31	95	98
					i		

85 24 63 45 17 31 95 98

j

i

85 24 63 45 17 31 95 98

j i

If $i \geq j$ interchange j^{th} and pivot elements and then divide the list into two sublists.

31 24 63 45 17 85 95 98

j

Two sublists:

31 24 63 45 17 | 85 | 95 98

Recursively sort

FirstPart and SecondPart

QickSort(low, j-1) QickSort(j+1,high)

Quick Sort Algorithm

```
Algorithm QuickSort(low,high)
//Sorts the elements a[low],…..,a[high] which resides in the global array a[1:n] into //ascending
    order;
// a[n+1] is considered to be defined and must ≥ all the elements  in a[1:n].
{
        if( low< high )  // if there are more than one element
        {           // divide p into two subproblems.
              j :=Partition(low,high);
                 // j is the position of the partitioning element.
              QuickSort(low,j-1);
              QuickSort(j+1,high);
               // There is no need for combining solutions.
        }
}
```

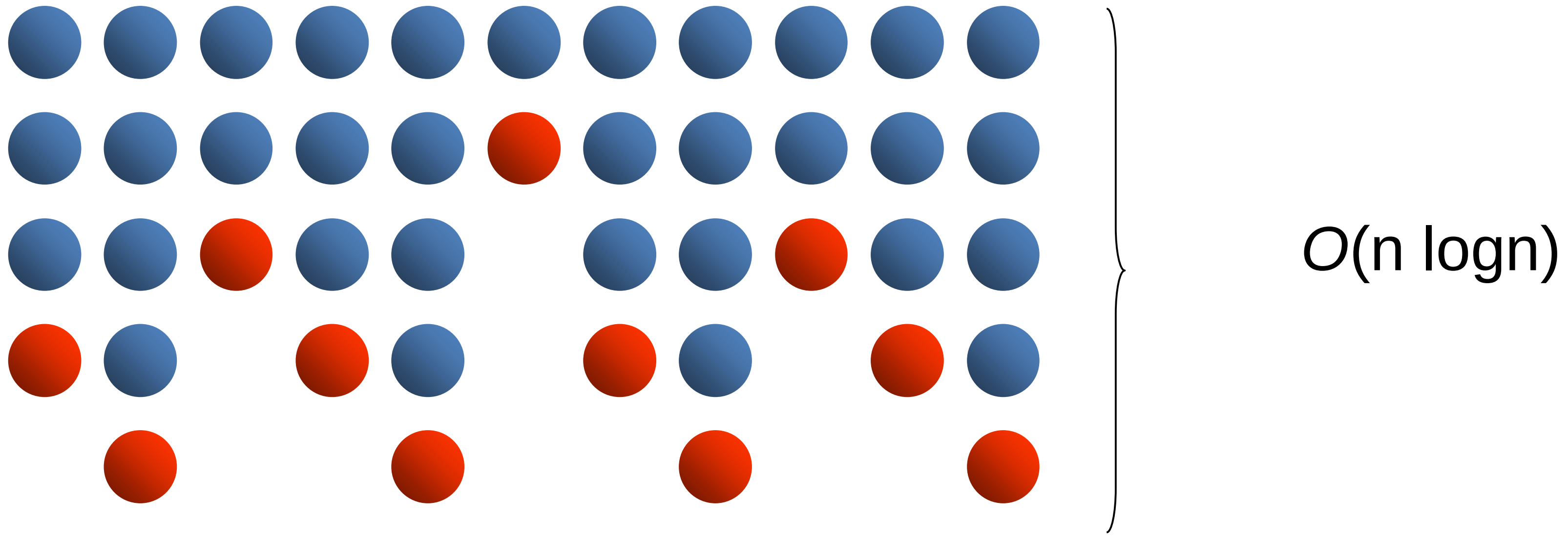
```
Algorithm Partition(l,h)
{
    pivot:= a[l] ; i:=l;  j:= h+1;
    while( i < j ) do
    {
            i++;

            while( a[ i ] < pivot ) do
                i++;
            j--;
            while( a[ j ]  > pivot ) do
                j--;

            if ( i < j ) then Interchange(i,j ); // interchange ith and
    }                                   //  jth elements.

    Interchange(l, j ); return j;  // interchange pivot and jth  element.
}
```

Algorithm interchange (x,y)

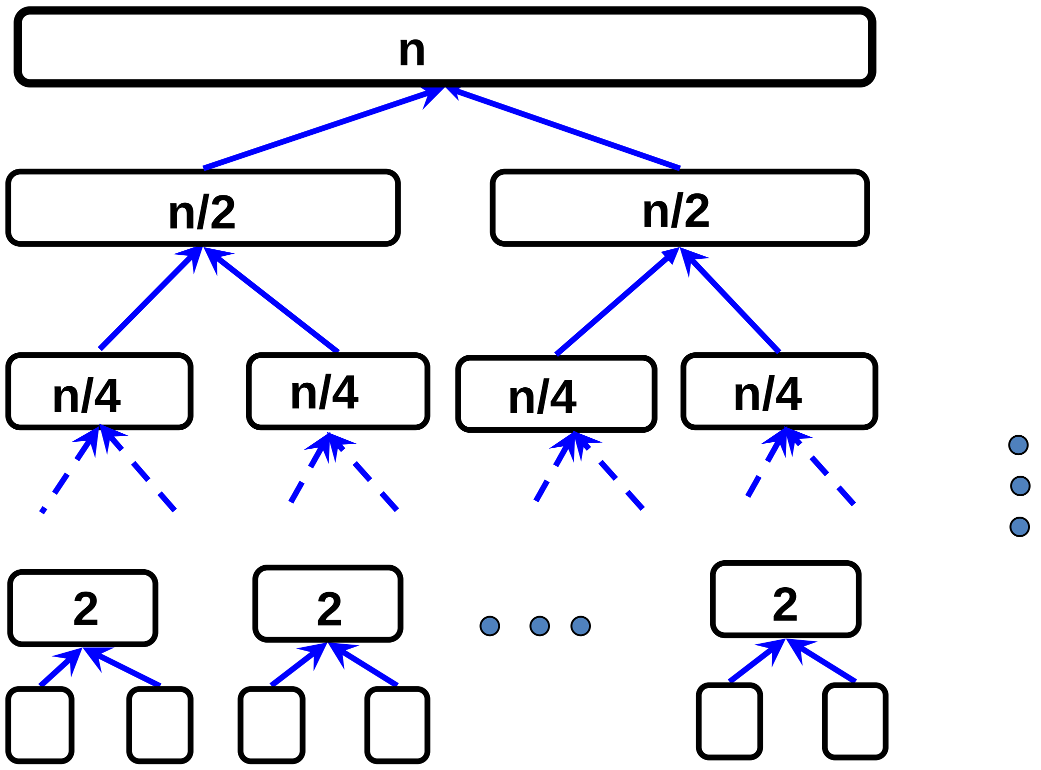
```
{
    temp=a[x];
    a[x]=a[y];
    a[y]=temp;
}
```

Time complexities of Quick sort
A best/good case

- It occurs only if each partition divides the list into two equal size sublists.
- T(n)=2T(n/2)+cn =O(nlog n)



Best/good Case



- Total time: $O(n\log n)$

Best Case Time Complexity

When n is a power of 2, $n = 2^k$, we can solve this equation by successive substitutions:

$$\begin{aligned} T(n) &= 2(2T(n/4) + cn/2) + cn \\ &= 4T(n/4) + 2cn \\ &= 4(2T(n/8) + cn/4) + 2cn \\ &\vdots \\ &= 2^k T(1) + kcn \\ &= an + cn \log n \end{aligned}$$

It is easy to see that if $2^k < n \leq 2^{k+1}$, then $T(n) \leq T(2^{k+1})$. Therefore

$$T(n) = O(n \log n)$$

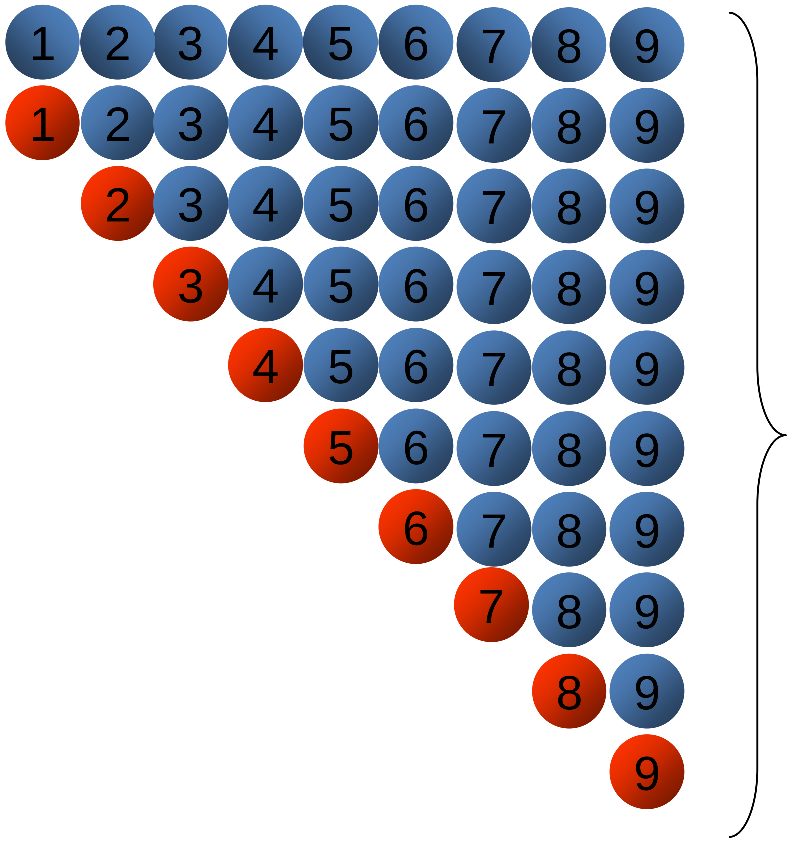
Average case complexity

- To do average case analysis, we need to consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy.
- We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set.
- Following is recurrence for this case.

- $T(n) = T(n/9) + T(9n/10) + (n)$
- $T(n)=O(nlogn)$

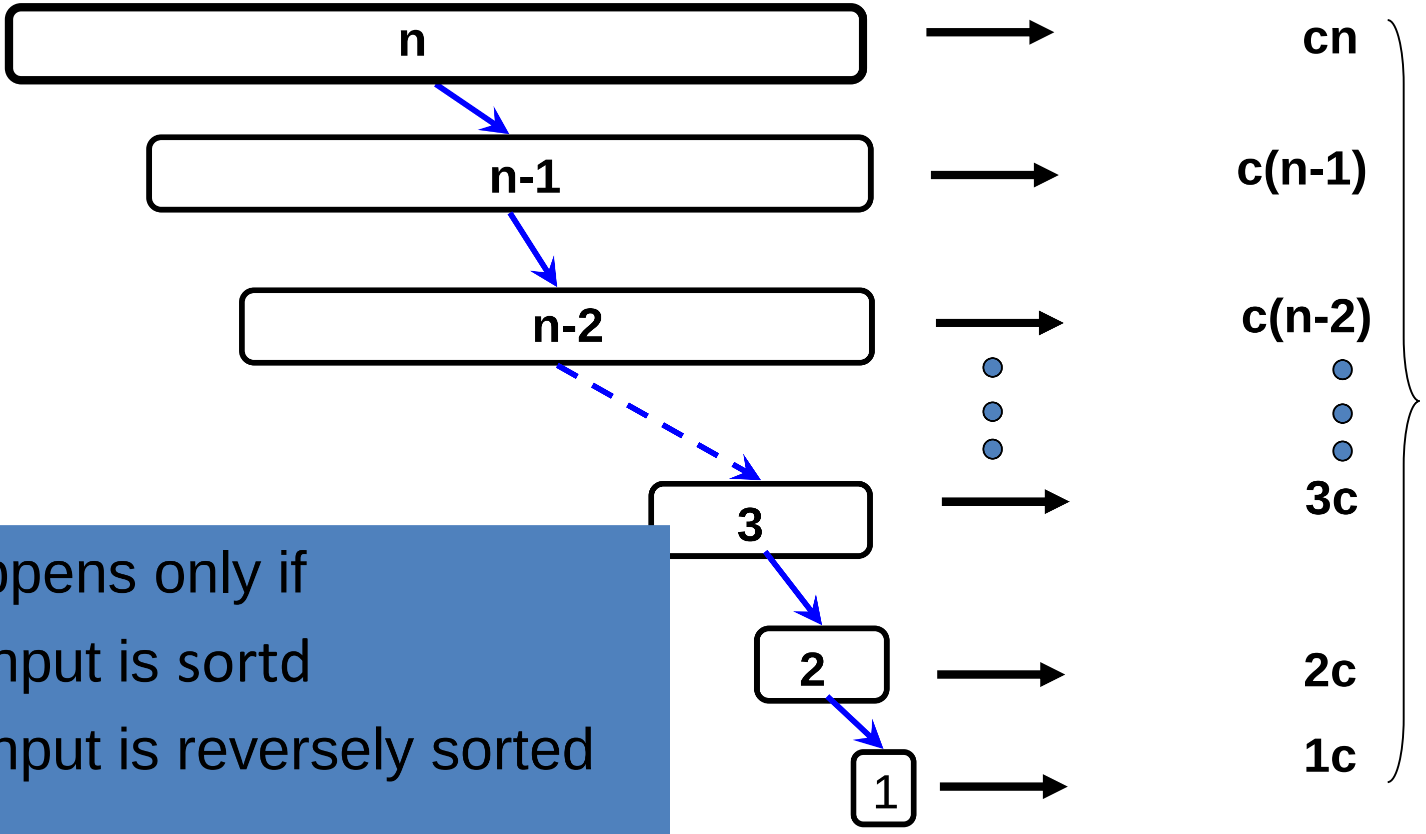
Time complexity analysis

A worst/bad case $T(n)=T(n-1)+cn=O(n^2)$



$O(n^2)$

Worst/bad Case



Happens only if

- input is sortd
- input is reversely sorted

Total time:

$O(n^2)$

Worst-Case Analysis:

The pivot is the smallest element, all the time. Then $i = 0$, and if we ignore $T(0) = 1$, which is insignificant, the recurrence is $T(N) = T(N - 1) + cN$, $N > 1$

We telescope, using above equation repeatedly. Thus,

$$T(N - 1) = T(N - 2) + c(N - 1)$$

$$T(N - 2) = T(N - 3) + c(N - 2)$$

$$\cdots$$

$$T(2) = T(1) + c(2)$$

Adding up all these equations yields

$T(N) = T(1) + c((N+1)(N/2)-1) = O(N^2)$ as claimed earlier.

SUMMARY

- **Pivot Selection**: Quick sort selects a pivot element from the array.

- **Partitioning**: It partitions the array into two sub-arrays, with elements less than the pivot on one side and elements greater than the pivot on the other.

- **Recursive Sorting**: The sub-arrays are recursively sorted using the same process.

- **Efficiency**: Quick sort has an average-case time complexity of O(n log n) and a worst-case time complexity of O(n^2), making it efficient for large datasets.

SELF-ASSESSMENT QUESTIONS

In the quick sort algorithm, what is the purpose of the pivot element?

(a) To act as a temporary storage
(b) To partition the array into two sub-arrays
(c) To find the maximum element
(d) To merge two sorted arrays

What is the average-case time complexity of quick sort?

(a) O(n)
(b) O(n log n)
(c) O(n^2)
(d) O(log n)

TERMINAL QUESTIONS

1. Analyze quick sort algorithm find out recurrence relation
2. Compare and contrast quick sort with merge sort in terms of their time complexity, space complexity, and use cases.

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books :

1Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein., 3rd, 2009, The MIT Press.

2 Algorithm Design Manual, Steven S. Skiena., 2nd, 2008, Springer.

3 Data Structures and Algorithms in Python, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser., 2nd, 2013, Wiley.

4 The Art of Computer Programming, Donald E. Knuth, 3rd, 1997, Addison-Wesley Professiona.

MOOCS :

1. https://www.coursera.org/specializations/algorithms?=

2.https://www.coursera.org/learn/dynamic-programming-greedy-algorithms#modules

THANK YOU