AssignmentNo.	**5**
Title	Quick Sort Algorithm
PROBLEM STATEMENT/DEFINITION	Write a program for analysis of quick sort by using deterministic and randomized variant.
Objectives	To implement algorithms that follow divide and conquer design Strategy To analysis quick sort by using deterministic and randomized variant. To understand concept of recursion.
Software packages and hardware apparatus used	MySQL/Oracle PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
References	Quicksort - Wikipedia QuickSort - GeeksforGeeks http://en.wikipedia.org/wiki/Quicksort
STEPS	Refer to details
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

AssignmentNo.1

Title : Quick sort Algorithm

Objectives:

- To implement algorithms that follow divide and conquer design Strategy
- To analysis quick sort by using deterministic and randomized variant.
- To understand concept of recursion.

Theory: **Deterministic Quicksort**

Algorithm

```
Quicksort(A,p,r)

{

if p < r then

q := Partition(A,p,r); // rearrange A[p..r] in place

Quicksort(A, p,q-1);

Quicksort(A,p+1,r);

}
```

Divide-and-Conquer

The design of Quicksort is based on the divide-and-conquer paradigm.

Divide: Partition the array A[p..r] into two (possibly empty) subarrays A [p..q-1] and A[q+1,r] such that

A[x] <= A[q] for all x in [p..q-1] A[x] > A[q] for all x in [q+1,r]

b) Conquer: Recursively sort A[p..q-1] and A[q+1,r]

c) Combine: nothing to do here

Example:

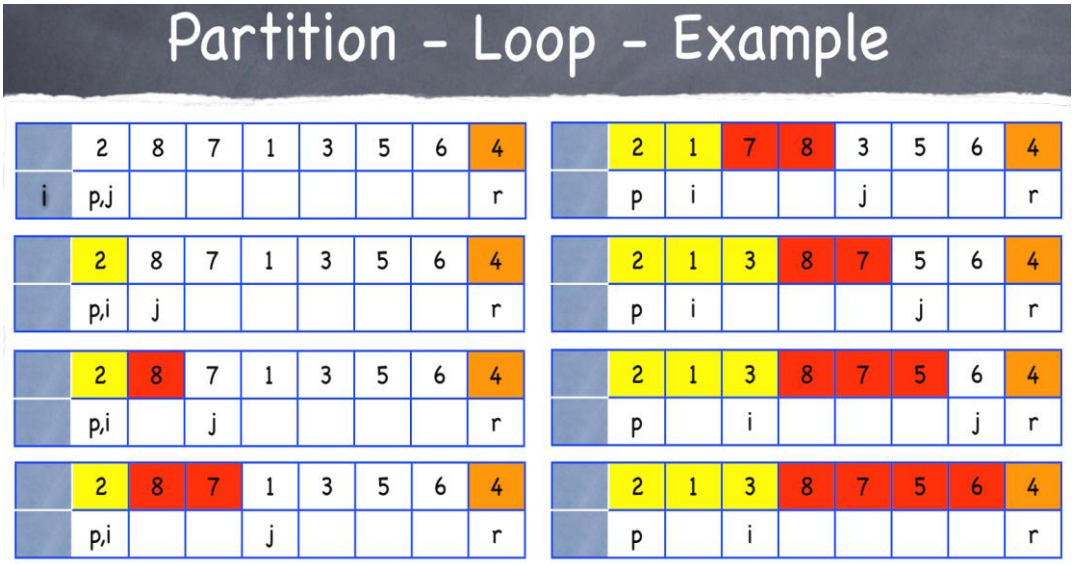
Partition Select pivot (orange element) and rearrange:

2	1	3	4	7	5	6	8	
p		i						r

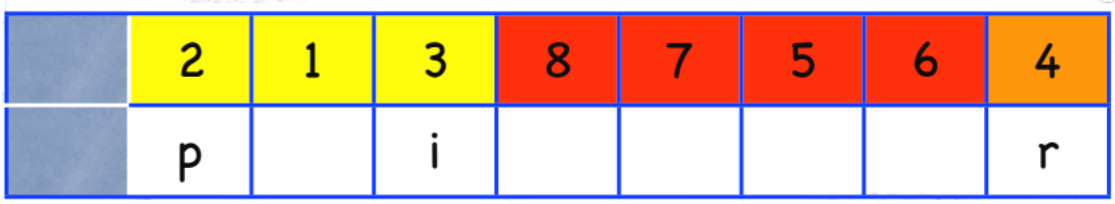
larger elements to the left of the pivot (red)

elements not exceeding the pivot to the right (yellow)

```
Partition(A,p,r)
x := A[r]; // select rightmost element as pivot
i := p-1;
for j = p to r-1
{ if A[j] <= x then
 i := i+1;
swap(A[i], A[j]);
 }
swap(A[i+1],A[r]);
return i+1
```



After the loop, the partition routine swaps the leftmost element of the right partition with the pivot element



swap(A[i+1],A[r])

now recursively sort yellow and red parts.

Analysis

Worst-Case Partitioning
The worst-case behavior for quicksort occurs on an input of length n when partitioning produces just one subproblem with n-1 elements and one subproblem with 0 elements. Therefore the recurrence for the running time

T(n) is: $T(n) = T(n-1) + T(0) + \theta(n) = T(n-1) + \theta(n) = \theta(n^2)$

Best-case partitioning:
If partition produces two subproblems that are roughly of the same size, then the recurrence of the running time is
$T(n) <= 2T(n/2) + \theta(n)$ so that
$T(n) = O(n \log n)$

Randomized Quicksort

Randomized Quicksort Algorithm

Randomized-Quicksort(A,p,r)

{

if p < r then

{

q := Randomized-Partition(A,p,r);

Randomized-Quicksort(A, p,q-1);

Randomized-Quicksort(A,p+1,r);

}

}

Randomized-Partition(A,p,r)

{

i := Random(p,r);

swap(A[i],A[r]);

Partition(A,p,r);

Almost the same as Partition, but now the pivot element is not the rightmost element, but rather an element from A[p..r] that is chosen uniformly at random.

}

Analysis

It follows that the expected running time of RandomizedQuicksort is O(n log n).

It is unlikely that this algorithm will choose a terribly unbalanced partition each time, so the performance is very good almost all the time.

Conclusion: We have successfully seen how deterministic and randomized quick sort is implemented

FAQ?

1. How quick sort work?
2. Which algorithm strategy quick sort uses?
3. How deterministic quicksort is different from randomize quick sort?
4. Explain with example deterministic quick sort?
5. What is the best case ,worst case ,average case complexity of Deterministic quick sort?
6. Comment on complexity of randomized quicksort.