Algorithms
Chapter 4: Divide and conquer

GATE CS PYQ
solved by Monalisa

- Section 5: Algorithms
- Searching, sorting, hashing. Asymptotic worst case time and space complexity. Algorithm design techniques : greedy, dynamic programming and divide-and-conquer . Graph traversals, minimum spanning trees, shortest paths
- Chapter 1:Algorithim Analysis:-Algorithm intro , Order of growth ,Asymptotic notation, Time complexity, space complexity, Analysis of Recursive & non recursive program, Master theorem]
- Chapter 2:Brute Force:-Sequential search, Selection Sort and Bubble Sort , Radix sort, Depth first Search and Breadth First Search.
- Chapter 3: Decrease and Conquer :- Insertion Sort, Topological sort,Binary Search .
- Chapter 4: Divide and conquer:-Min max problem , matrix multiplication ,Merge sort ,Quick Sort , Binary Tree Traversals and Related Properties .
- Chapter 5: Transform and conquer:- Heaps and Heap sort, Balanced Search Trees.
- Chapter 6: Greedy Method:-knapsack problem , Job Assignment problem, Optimal merge, Hoffman Coding, minimum spanning trees, Dijkstra's Algorithm.
- Chapter 7: Dynamic Programming:-The Bellman-Ford algorithm ,Warshall's and Floyd's Algorithm ,Rod cutting, Matrix-chain multiplication ,Longest common subsequence ,Optimal binary search trees
- Chapter 8: Hashing.
- Reference : Introduction to Algorithms by Thomas H. Cormen
- Introduction to the Design and Analysis of Algorithms, by Anany Levitin
- My Note

- GATE CS 2008 | Question: 43
- Consider the Quicksort algorithm. Suppose there is a procedure for finding a pivot element which splits the list into two sub-lists each of which contains at least one-fifth of the elements. Let $T(n)$ be the number of comparisons required to sort n elements. Then
- (A) $T(n) \leq 2T(n/5)+n$ (B) $T(n) \leq T(n/5)+T(4n/5)+n$
- (C) $T(n) \leq 2T(4n/5)+n$ (D) $T(n) \leq 2T(n/2)+n$
- One part contains n/5 elements and the other part contains 4n/5 elements
- Ans: (B) $T(n) \leq T(n/5)+T(4n/5)+n$

- GATE CS 2009 | Question: 39
- In quick-sort, for sorting *n* elements, the $(n/4)^{th}$ smallest element is selected as pivot using an O(n) time algorithm. What is the worst case time complexity of the quick sort?
- (A)Θ(n) (B)Θ(nlogn) (C)Θ(n^2) (D)Θ(n^2logn)
- After the $(n/4)^{th}$ smallest element is selected as pivot the list gets partitioned into one with n/4 and other with 3n/4 elements.
- So recurrence : T(n)=T(n/4)+T(3n/4)+ O(n) [pivot selection time O(n)]
- By using variation of master theorem
- T (n) = T (αn) + T((1- α)n) +cn [$0<\alpha<1, c>0$] then T(n)=**Θ(n log n)**
- Worst case time complexity of the quick sort Θ(n log n)
- Ans: (B)Θ(nlogn)

- GATE CS 2012 | Question: 39
- A list of *n* strings, each of length *n*, is sorted into lexicographic order using the merge-sort algorithm. The worst case running time of this computation is
- (A)O(nlogn) (B)O(n^2logn) (C)O(n^2+logn) (D)O(n^2)
- The worst case complexity of merge sort is O(n log n).
- Hence the worst case complexity for sorting first letter of n strings will be O(nlog n).
- In worst case, we have to sort all n letters in each string.
- Hence total complexity will be upto O(n * n log n) = O(n^2 log n)
- Ans : (B)O(n^2logn)

- GATE CS 2014 Set 1 | Question: 14
- Let P be quicksort program to sort numbers in ascending order using the first element as the pivot. Let t_1 and t_2 be the number of comparisons made by P for the inputs [1 2 3 4 5] and [4 1 5 3 2] respectively. Which one of the following holds?
- (A)$t_1=5$ (B)$t_1<t_2$ (C)$t_1>t_2$ (D)$t_1=t_2$
- [1 2 3 4 5] is the worst case of quicksort
- When first element or last element is chosen as pivot, Quick Sort behave worst case for the sorted array. Hence number of comparisons are high.
- [4 1 5 3 2] is best case as it is in random order. Hence number of comparisons are low.
- Ans: (C)$t_1>t_2$

- GATE CS 2014 Set 1 | Question: 39
- The minimum number of comparisons required to find the minimum and the maximum of 100 numbers is _____
- Number of comparison for divide and conquer minmax problem 1.5n-2
- 1.5*100-2=150-2=148
- Ans: 148

- **GATE CS 2014 Set 3 | Question: 14**
- You have an array of n elements. Suppose you implement quicksort by always choosing the central element of the array as the pivot. Then the tightest upper bound for the worst case performance is
- (A)$O(n^2)$ (B)$O(n\log n)$ (C)$\Theta(n\log n)$ (D)$O(n^3)$
- When we choose the first element as the pivot, the worst case of quick sort comes if the input is already sorted order.
- Now, when we choose the middle element as pivot, sorted input no longer gives worst case behavior . For example,
- 1,2,3,4,5,6,7 worst case behavior for quick sort when the first element is pivot but not for middle element as pivot.
- One input behave worst case if minimum or maximum element is pivot.
- If in input always central element is the minimum or Maximum then its worst case.
- Each split will be 1 element on one side and $n-1$ elements on other side.
- $T(n) = 2T(n/2) + \Theta(n)$ Best case
- $T(n) = T(n-1)+\Theta(n)$ Worst case
- Ans: (A)$O(n^2)$

- GATE CS 2015 Set 1 | Question: 2
- Which one of the following is the recurrence equation for the worst case time complexity of the quick sort algorithm for sorting n(≥2) numbers? In the recurrence equations given in the options below, c is a constant.
- (A)$T(n)=2T(n/2)+cn$ (B)$T(n)=T(n-1)+T(1)+cn$
- (C)$T(n)=2T(n-1)+cn$ (D)$T(n)=T(n/2)+cn$
- Worst case for quick sort happens when each split will be 1 element on one side and n-1 elements on other side.
- If one array is already sorted increasing or decreasing order are worst case.
- Recurrence relation :$T(n)=T(n-1)+T(1)+cn$
- Ans: (B)$T(n)=T(n-1)+T(1)+cn$

GATE CS 2015 Set 2 | Question: 45

Suppose you are provided with the following function declaration in the C programming language.

```
int partition(int a[], int n);
```

The function treats the first element of a[] as a pivot and rearranges the array so that all elements less than or equal to the pivot is in the left part of the array, and all elements greater than the pivot is in the right part. In addition, it moves the pivot so that the pivot is the last element of the left part. The return value is the number of elements in the left part.

The following partially given function in the C programming language is used to find the k^{th} smallest element in an array a[] of size n using the partition function. We assume k<n.

```
int kth_smallest (int a[], int n, int k) {
    int left_end = partition (a, n);
    if (left_end+1==k) {
        return a[left_end]; }
    if (left_end+1 > k) {
        return kth_smallest (_____); }
    else {
        return kth_smallest (_____); } }
```

- if (left_end+1 > k) means k^{th} smallest element on the left array.
- The "+1" is for array index in C language starts from 0.
- We can do a recursive call as kth_smallest(a, left_end, k);
- If the above condition is false, left_end +1<k, means k^{th} smallest element is on the right part of the array and it will be (k-left_end-1)th element there as left_end+1 elements are gone in the left part.
- So, the recursive call will be kth_smallest(a + left_end + 1, n - left_end – 1, k - left_end - 1);

The missing arguments lists are respectively

Ans: (A)

(A)(a, left_end,k) and (a+left_end+1,n−left_end−1,k−left_end−1)

(B)(a, left_end,k) and (a,n−left_end−1,k−left_end−1)

(C)(a+ left_end+1,n−left_end−1,k−left_end−1) and (a, left_end,k)

(D)(a,n−left_end−1,k−left_end−1) and (a,left_end,k)

- The return value of partition is the number of elements in the left part.
- Lets take a array of n=10, k=5
- *int kth_smallest (int a[], int n, int k) { //(a,10,5)*
- a={5,4,3,2,1,7,6,10,9,8} After *partition* a={1,4,3,2,5,7,6,10,9,8}
- *int left_end = partition (a, n); //left_end = 4*
- *if (left_end+1==k) //4+1=5*
- *{ return a[left_end]; } //a[4]=5*
- Let k=4
- *if (left_end+1 > k) //4+1>4*
- *{ return kth_smallest (a, left_end,k); } //(a,4,4)*
- a={1,4,3,2} After *partition* a={1,4,3,2}
- *int left_end = partition (a, n); //left_end = 0*
- *else //0+1<4*
- *{ return kth_smallest(a+left_end+1,n−left_end−1, k−left_end−1);} //(a+1,3,3)*
- a={4,3,2} After *partition* a={2,3,4}
- *int left_end = partition (a, n); //left_end = 2*
- *if (left_end+1==k) //2+1=3*
- *{ return a[left_end]; } //a[2]=4*

- *int kth_smallest (int a[], int n, int k) {*
- *int left_end = partition (a, n);*
- *if (left_end+1==k) {*
- *return a[left_end]; }*
- *if (left_end+1 > k) {*
- *return kth_smallest (a, left_end,k); }*
- *else { return kth_smallest (a+left_end+1,n−left_end−1,k−left_end−1); } }*
- Let k=7
- *else { return kth_smallest (a+left_end+1,n−left_end−1,k−left_end−1); } } //(a+5,5,2)*
- a={7,6,10,9,8} After *partition* a={6,7,10,9,8}
- *int left_end = partition (a, n); //left_end = 1*
- *if (left_end+1==k) //1+1=2*
- *{ return a[left_end]; } //a[1]=7*

- GATE CS 2015 Set 3 | Question: 27
- Assume that a mergesort algorithm in the worst case takes 30 seconds for an input of size 64. Which of the following most closely approximates the maximum input size of a problem that can be solved in 6 minutes?
- (A)256 (B)512 (C)1024 (D)2018
- The worst case time complexity of Mergesort is $\Theta(n\log n) = c \times n\log n$.
- For an input of size 64, the algorithm takes 30s.
- $c \times 64 \log_2 64 = 30s$
- $c \times 64 \times 6 = 30s$
- $c = \frac{30}{64 \times 6} = \frac{5}{64}$ s
- Let the input size of the problem that can be solved in 6 minutes be x.
- $c \times x\log_2 x = 6m = 6 \times 60s = 360s$
- $x\log_2 x = \frac{360}{c} = \frac{360 \times 64}{5} = 72 \times 64 = 3^2 \times 2^3 \times 2^6 = 9 \times 2^9 = 512 \times 9$
- $x\log_2 x = 512 \times \log_2 512$
- $x = 512$
- Ans: (B)512

- GATE CS 2016 Set 1 | Question: 13
- The worst case running times of *Insertion sort* , *Merge sort* and *Quick sort*, respectively are:
- (A)$\Theta(n\log n)$, $\Theta(n\log n)$ and $\Theta(n^2)$
- (B)$\Theta(n^2)$, $\Theta(n^2)$ and $\Theta(n\log n)$
- (C)$\Theta(n^2)$, $\Theta(n\log n)$ and $\Theta(n\log n)$
- (D)$\Theta(n^2)$, $\Theta(n\log n)$ and $\Theta(n^2)$
- Running Times Best Case Worst case
- *Insertion sort* = $\Theta(n)$ $\Theta(n^2)$
- *Merge sort*= $\Theta(n\log n)$ $\Theta(n\log n)$
- *Quick sort*= $\Theta(n\log n)$ $\Theta(n^2)$
- Ans: (D)$\Theta(n^2)$, $\Theta(n\log n)$ and $\Theta(n^2)$

- GATE CS 2016 Set 2 | Question: 13
- Assume that the algorithms considered here sort the input sequences in ascending order. If the input is already in the ascending order, which of the following are TRUE?
- I. Quicksort runs in $\Theta(n^2)$ time II. Bubblesort runs in $\Theta(n^2)$ time
- III. Mergesort runs in $\Theta(n)$ time IV.Insertion sort runs in $\Theta(n)$ time
- (A)I and II only (B)I and III only
- (C)II and IV only (D)I and IV only
- I. Quicksort runs in $\Theta(n^2)$ time for already sorted input. This is true
- II. If no swap happens then bubble sort can stop in single loop.$\Theta(n)$ is best case.This is false.
- III. Mergesort always runs in $\Theta(nlogn)$ time. This is false
- IV. Insertion sort runs in $\Theta(n)$ time in case of sorted input(Best case). This is true.
- Ans: (D)I and IV only

- **GATE CS 2019 | Question: 20**
- An array of 25 distinct elements is to be sorted using quicksort. Assume that the pivot element is chosen uniformly at random. The probability that the pivot element gets placed in the worst possible location in the first round of partitioning (rounded off to 2 decimal places) is ___.
- If pivot element is Minimum or Maximum then it is Worst case of quicksort.
- Total elements =25
- For worst case number of candidates =2
- The probability=$\frac{2}{25}$ =0.08
- Ans: 0.08
-

- **GATE CS 2019 | Question: 25**
- Consider a sequence of 14 elements:
- A=[−5,−10,6,3,−1,−2,13,4,−9,−1,4,12,−3,0] . The sequence sum S(i,j)=$\sum_{k=i}^{j} A[k]$. Determine the maximum of S(i,j), where $0 \leq i \leq j < 14$. (Divide and conquer approach may be used.)
- Answer: _____
- **Subsequence :** A subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.
- **Subarray :** A subarray of n-element array is an array composed from a contiguous block of the original array's elements.
- maximum of S(i,j)=S(2,11)=6 + 3 – 1 – 2 + 13 + 4 – 9 – 1 + 4 + 12=29
- Ans: 29

GATE CS 2021 Set 1 | Question: 2

Let *P* be an array containing *n* integers. Let *t* be the lowest upper bound on the number of comparisons of the array elements, required to find the minimum and maximum values in an arbitrary array of *n* elements. Which one of the following choices is correct?

A.$t>2n-2$ B.$t>3\lceil n/2\rceil$ and $t\leq 2n-2$ C.$t > n$ and $t \leq 3\lceil n/2\rceil$ D.$t>\lceil \log_2(n)\rceil$ and $t\leq n$

- Best case comparison =n-1 [Ascending order]
- Worst case comparison =2(n-1) [Descending order]
- Average case comparison=$\frac{3n}{2}$-2 [Random order]
- Recurrence Relation $T(n)=2T(\frac{n}{2})+2$
- After solving $T(n)= \frac{3n}{2}$-2
- Best case ≤ Average case ≤ Worst case
- $n-1 \leq \frac{3n}{2}-2 \leq 2n-2$
- *t* be the lowest upper bound.
- So *t* will be in between Best case and Avg case. $n-1 < t \leq \frac{3n}{2}-2$
- Ans : C.$t > n$ and $t \leq 3\lceil n/2$

GATE DA 2024| Question: 18

Consider the following tree traversals on a full binary tree:

(i) Preorder (ii) Inorder (iii) Postorder

Which of the following traversal options **is/are** sufficient to uniquely reconstruct the full binary tree?

(A) (i) and (ii) (B) (ii) and (iii) (C) (i) and (iii) (D) (ii) only

Ans : (A),(B),(C)

- **GATE DA 2024| Question: 20**
- Consider sorting the following array of integers in ascending order using an in-place Quicksort algorithm that uses the last element as the pivot.

60	**70**	**80**	**90**	**100**

- The minimum number of swaps performed during this Quicksort is _____.
- As it is already sorted so 0 swap required to perform Quicksort
- Ans : 0

- **GATE DA 2024| Question: 42**
- Let H, I, L, and N represent height, number of internal nodes, number of leaf nodes, and the total number of nodes respectively in a rooted binary tree.Which of the following statements is/are always **TRUE**?
- (A)$L \leq I + 1$ (B)$H + 1 \leq N \leq 2^{H+1} - 1$ (C)$H \leq I \leq 2^{H} - 1$ (D)$H \leq L \leq 2^{H\text{-}1}$
- (A)$L \leq I + 1$ number of leaf nodes ≤ number of internal nodes+1 True
- (B)$H + 1 \leq N \leq 2^{H+1} - 1$ height+1 ≤ total number of nodes ≤ $2^{\text{height}+1} - 1$ True
- (C)$H \leq I \leq 2^{H} - 1$ height ≤ number of internal nodes ≤ $2^{\text{height}} - 1$ True
- (D)$H \leq L \leq 2^{H\text{-}1}$ height ≤ number of leaf nodes ≤ $2^{\text{height-1}}$ False
- Ans : (A),(B),(C)