GUJARAT TECHNOLOGICAL UNIVERSITY

BE-5 SEMESTER – OLD PAPER – S22 TO W25 – QUESTION BANK ANSWER

Subject Name & Code:

ANALYSIS AND DESIGN OF ALGORITHMS (3150703)

Unit 3: Divide and Conquer

Repeated Question 1 – Write and analyze binary search algorithm. Give its recurrence relation and solve it. (7 marks)

Appeared in: S25 (Q1c, 07), S23 (Q2c, 07), W24 (Q2c, 07), plus others lower marks – highest 7

Ans:

Binary Search Algorithm (iterative version):

```
Algorithm BinarySearch(arr, low, high, key):
  while low ≤ high do
    mid = floor((low + high) / 2)
    if arr[mid] == key then
      return mid
    else if arr[mid] < key then
      low = mid + 1
    else
      high = mid - 1
  return -1   // not found
```

Recurrence relation:

Each step compares with middle element and recurses on **one half**.

$$T(n) = T(n/2) + O(1)$$
$$T(1) = O(1)$$

Solving recurrence using Master Theorem:

General form: $T(n) = aT(n/b) + f(n)$

Here $a = 1, b = 2, f(n) = O(1) = O(n^0)$

Compute $\log_b a = \log_2 1 = 0 \rightarrow$ Compare $f(n)$ with $n^{\log_b a} = n^0 = 1$.

Since $f(n) = \Theta(1)$, case 2 applies (equal).

Solution: $T(n) = \Theta(n^{\log_b a} \log n) = \Theta(\log n)$

Alternate method (back substitution):

$$T(n) = T(n/2) + 1$$
$$= [T(n/4) + 1] + 1 = T(n/4) + 2$$

After k steps: $T(n) = T(n/2^k) + k$

When $n/2^k = 1 \Rightarrow k = \log_2 n$

$$T(n) = T(1) + \log n = O(\log n)$$

Real-world application: Searching in a sorted phonebook, database indexed column, or dictionary.

Repeated Question 2 – Explain Merge Sort algorithm with example, recurrence, and complexity. (7 marks)
Appeared in: S23 (Q2c, 07), W25 (Q2c, 07), W22 (Q2c, 07)
Ans:
Merge Sort Algorithm (Divide and Conquer):

1. **Divide:** Split array into two halves.
2. **Conquer:** Recursively sort each half.
3. **Combine:** Merge two sorted halves into one sorted array.

```
Algorithm MergeSort(arr, l, r):
  if l < r then
    m = (l + r) / 2
    MergeSort(arr, l, m)
    MergeSort(arr, m+1, r)
    Merge(arr, l, m, r)
```

Merge function – O(n) time using temporary arrays.
Example: Sort [38, 27, 43, 3, 9, 82, 10]
Split: [38,27,43,3] and [9,82,10] → further split until single elements → merge back:
[27,38] [3,43] → [3,27,38,43] ; [9,10,82] → final [3,9,10,27,38,43,82]
Recurrence relation:

$$T(n) = 2T(n/2) + O(n)$$

Where O(n) is time for merging.
Solving recurrence (Master Theorem):
$a = 2, b = 2, f(n) = O(n) \rightarrow \log_b a = \log_2 2 = 1 \rightarrow f(n) = \Theta(n^1) \rightarrow$ Case 2 applies.

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

Time Complexity:

- Best, worst, average: $\Theta(n \log n)$
- Space complexity: $O(n)$ auxiliary (temporary arrays)

Real-world application: External sorting (large files that don't fit in memory), stable sorting in programming libraries.

Repeated Question 3 – Explain Quick Sort algorithm, trace on given array, and comment on best/worst/average case. (7 marks)
Appeared in: S25 (Q2b, 07), S24 (Q2c, 07), W23 (Q2c, 07), S22 (Q1c, 07)
Ans:
Quick Sort Algorithm (Divide and Conquer):

1. Choose a **pivot** element.
2. **Partition** array so that elements < pivot come before, > pivot after.
3. Recursively sort left and right subarrays.

```
Algorithm QuickSort(arr, low, high):
  if low < high then
    pivotIndex = Partition(arr, low, high)
    QuickSort(arr, low, pivotIndex - 1)
    QuickSort(arr, pivotIndex + 1, high)
```

Partition function (Lomuto scheme): chooses last element as pivot, rearranges, returns pivot final index.
Trace on sample array: [10, 7, 8, 9, 1, 5] with pivot = last element (5)
After partition: [1, 5, 8, 9, 10, 7] (pivot at index 1)
Recursively sort left [1] (trivial) and right [8,9,10,7] with pivot 7 → … Final sorted:

[1,5,7,8,9,10]
Recurrence relation:

- **Best case** (pivot always divides into equal halves): $T(n) = 2T(n/2) + O(n) \rightarrow O(n\log\ n)$
- **Worst case** (pivot is smallest or largest, e.g., already sorted): $T(n) = T(n-1) + O(n) \rightarrow O(n^2)$
- **Average case** (random pivot): $O(n\log\ n)$

Complexity comparison:

Case	Time Complexity
Best	$O(n\log\ n)$
Worst	$O(n^2)$
Average	$O(n\log\ n)$

Space complexity: $O(\log\ n)$ recursion stack (best) to $O(n)$ worst.
Real-world application: Most practical sorting library (C++ std::sort, Java Arrays.sort for primitives) uses optimized Quick Sort or hybrid.

Repeated Question 4 – Solve recurrence using Master Theorem. (7 marks)
Appeared in: S24 (Q2b, 04), W24 (Q2c, 07), W23 (Q1b, 04), S22 (Q2c, 07) – highest 7 marks
Ans:
Master Theorem – For recurrence $T(n) = aT(n/b) + f(n)$ with $a \geq 1, b > 1$:
Let $p = \log_b a$. Compare $f(n)$ with n^p:

Case	Condition	Solution
1	$f(n) = O(n^{p-\epsilon})$ for some $\epsilon > 0$	$T(n) = \Theta(n^p)$
2	$f(n) = \Theta(n^p \log^k n)$ with $k \geq 0$	$T(n) = \Theta(n^p \log^{k+1} n)$
3	$f(n) = \Omega(n^{p+\epsilon})$ and $af(n/b) \leq cf(n)$ for some $c < 1$	$T(n) = \Theta(f(n))$

Examples solved:

1. $T(n) = 9T(n/3) + n$
   $a = 9, b = 3, p = \log_3 9 = 2, f(n) = n = O(n^{2-1}) \rightarrow$ Case 1 $\rightarrow \Theta(n^2)$
2. $T(n) = T(2n/3) + 1$
   $a = 1, b = 3/2, p = \log_{3/2} 1 = 0, f(n) = 1 = \Theta(n^0 \log^0 n) \rightarrow$ Case 2 $\rightarrow \Theta(\log\ n)$
3. $T(n) = 2T(n/2) + n\log\ n$
   $a = 2, b = 2, p = 1, f(n) = n\log\ n = \Theta(n^1 \log^1 n) \rightarrow$ Case 2 with $k = 1 \rightarrow \Theta(n\log^2 n)$

Note: For the 7-mark question, you must state the theorem, three cases, and solve at least two recurrences from the paper or as above.

Other Important Question 1 – Write recurrence for Quick Sort best/worst case and solve. (7 marks)
Appeared in: W25 (Q2c, 07)
Ans:
Quick Sort recurrence – depends on pivot selection.

Worst case (pivot always min or max, e.g., sorted array with last element as pivot):
Partition produces one subarray of size $n-1$ and one of size 0.
$T(n) = T(n-1) + T(0) + O(n) \rightarrow T(n) = T(n-1) + O(n)$
Solve: $T(n) = T(n-1) + cn$
Expand: $T(n) = T(n-2) + c(n-1) + cn = \cdots = T(1) + c\sum_{k=2}^{n} k = O(n^2)$
Best case (pivot divides array into two equal halves):

$$T(n) = 2T(n/2) + O(n)$$

By Master Theorem (a=2, b=2, f(n)=n, p=1, case 2): $T(n) = \Theta(n\log n)$
Average case (random pivot, equally likely positions):
$T(n) = \frac{1}{n}\sum_{k=1}^{n}[T(k-1) + T(n-k)] + O(n) \rightarrow$ solves to $O(n\log n)$

Other Important Question 2 – Solve recurrence using suitable method: $T(n) = T(n/6) + T(5n/6) + \Theta(n)$ (7 marks)
Appeared in: S25 (Q2c, 07)
Ans:
Use **Recursion Tree Method**.

- At root: work = cn (for some constant c)
- Two children: sizes $n/6$ and $5n/6$, each contributes $c(n/6)$ and $c(5n/6)$, total = cn
- Next level: four children: sizes $n/36, 5n/36, 5n/36, 25n/36$. Sum = $c(n/36 + 5n/36 + 5n/36 + 25n/36) = c(36n/36) = cn$
- Every level has total work = cn.
- Tree depth: longest path shrinks by factor $5/6$. Size becomes 1 when $(5/6)^k n = 1 \rightarrow k = \log_{6/5} n$.
- Number of levels = $O(\log n)$.
- Total work = $cn \times$ levels = $\Theta(n\log n)$.

Final answer: $T(n) = \Theta(n\log n)$

Other Important Question 3 – Solve following recurrences: (a) $T(n) = 2T(n/2) + n\log n$ (b) $T(n) = 4T(n/4) + n$ (7 marks)
Appeared in: S25 (Q2c, 07)
Ans:
(a) $T(n) = 2T(n/2) + n\log n$
Master Theorem: $a = 2, b = 2, p = \log_2 2 = 1$, $f(n) = n\log n = \Theta(n^1 \log^1 n) \rightarrow$ Case 2 with $k = 1$.

$$T(n) = \Theta(n\log^2 n)$$

(b) $T(n) = 4T(n/4) + n$
Master Theorem: $a = 4, b = 4, p = \log_4 4 = 1$, $f(n) = n = \Theta(n^1) \rightarrow$ Case 2 with $k = 0$.

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

Other Important Question 4 – Explain recursion-tree method with suitable example. (7 marks)
Appeared in: S24 (Q2c, 07)
Ans:
Recursion-tree method – visualizes recurrence as a tree where each node represents the cost of a subproblem. Sum costs across levels.
Steps:

1. Draw root with cost $f(n)$ and a children each of size n/b.
2. Expand until base case (size 1).
3. Compute cost per level and sum over levels.

Example: $T(n) = 2T(n/2) + n^2$

- Level 0: cost = n^2
- Level 1: two nodes, each cost $(n/2)^2 \rightarrow$ total = $2 \cdot n^2/4 = n^2/2$
- Level 2: four nodes, each cost $(n/4)^2 \rightarrow$ total = $4 \cdot n^2/16 = n^2/4$
- Level k (k from 0 to $\log_2 n$): total cost = $n^2/2^k$
- Sum = $n^2 \sum_{k=0}^{\log n}(1/2^k) = n^2 \cdot \frac{1-(1/2)^{\log n+1}}{1-1/2} \approx 2n^2$
- Therefore $T(n) = \Theta(n^2)$

[DG PROMPT]
Title: Recursion Tree for T(n)=2T(n/2)+n²
Description: Draw a binary tree. Root labeled "n²". Two children labeled "(n/2)²". Each child further splits into two grandchildren labeled "(n/4)²". Continue until leaves of size 1. Show level sums: level0 = n², level1 = n²/2, level2 = n²/4, etc.

Other Important Question 5 – Solve recurrence using iterative method: $T(n) = T(n-1) + 1, T(0) = 0$ (7 marks)
Appeared in: S22 (Q2c, 07)
Ans:
Iterative method (back substitution):

$$T(n) = T(n-1) + 1$$
$$= [T(n-2) + 1] + 1 = T(n-2) + 2$$

After k steps: $T(n) = T(n-k) + k$
Base: $T(0) = 0$. Set $n - k = 0 \rightarrow k = n$

$$T(n) = T(0) + n = 0 + n = n$$

Therefore $T(n) = \Theta(n)$

Other Important Question 6 – What is recurrence? Solve $T(n) = T(n-1) + 1$ using substitution method. (4 marks)
Appeared in: S23 (Q2b, 04)
Ans:
Recurrence relation – An equation that defines a function in terms of its value at smaller inputs. Used to analyze recursive algorithms.
Solve $T(n) = T(n-1) + 1$ with $T(0) = 0$ by substitution:
Guess: $T(n) = n$
Base case: $n = 0, T(0) = 0$ matches.
Inductive step: Assume $T(k) = k$ for all $k < n$.
Then $T(n) = T(n-1) + 1 = (n-1) + 1 = n$.
Hence proved. Complexity $\Theta(n)$.

Other Important Question 7 – Perform analysis of recurrence $T(n) = 2T(n/2) + \Theta(n^2)$ using recurrence tree. (7 marks)
Appeared in: W22 (Q2c, 07)
Ans:
Recursion tree for $T(n) = 2T(n/2) + cn^2$ (c constant):

- Level 0: cost = cn^2
- Level 1: two nodes, each cost $c(n/2)^2 = cn^2/4 \rightarrow$ total = $2 \cdot cn^2/4 = cn^2/2$
- Level 2: four nodes, each cost $c(n/4)^2 = cn^2/16 \rightarrow$ total = $4 \cdot cn^2/16 = cn^2/4$
- Level i: 2^i nodes, each cost $c(n/2^i)^2 = cn^2/4^i \rightarrow$ total = $2^i \cdot cn^2/4^i = cn^2/2^i$
- Number of levels: $\log_2 n$ (until size 1)

- Total cost = $cn^2 \sum_{i=0}^{\log} (1/2^i) = cn^2 \cdot \frac{1-(1/2)^{\log n+1}}{1-1/2} = cn^2(2 - 2/n) = \Theta(n^2)$

Thus $T(n) = \Theta(n^2)$. (The n^2 term dominates over the recursion.)

Other Important Question 8 – Write algorithm for Max-Min problem using divide and conquer. Calculate complexity. (4 marks)

Appeared in: W24 (Q3b, 04)

Ans:

Max-Min problem – Find maximum and minimum elements in an array.

Divide and Conquer algorithm:

```
Algorithm MaxMin(arr, low, high):
   if low == high then
      return (arr[low], arr[low])
   else if high == low + 1 then
      if arr[low] < arr[high] then
         return (arr[low], arr[high])
      else
         return (arr[high], arr[low])
   else
      mid = (low + high) / 2
      (min1, max1) = MaxMin(arr, low, mid)
      (min2, max2) = MaxMin(arr, mid+1, high)
      return (min(min1, min2), max(max1, max2))
```

Recurrence: $T(n) = 2T(n/2) + O(1)$ (only two comparisons at merge)

Solve: $a = 2, b = 2, f(n) = 1 = O(n^0)$, $\log_2 2 = 1 \rightarrow$ Case 1 $\rightarrow T(n) = \Theta(n)$.

Better than brute-force (2n-2 comparisons) – total comparisons $\approx 3n/2 - 2$.

Other Important Question 9 – Write algorithm for matrix multiplication using divide and conquer. Calculate complexity. (4 marks)

Appeared in: W24 (Q3b, 04)

Ans:

Standard Divide and Conquer Matrix Multiplication (not Strassen's):

Given matrices A and B of size n×n (n power of 2).

Divide each into four submatrices of size n/2 × n/2:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Then $C = A \times B$ where:

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}, C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}, C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

Algorithm:

```
MatrixMultiply(A, B, n):
   if n == 1 then return A[1][1] * B[1][1]
   Partition A and B into submatrices (n/2)
   C11 = Add(MatrixMultiply(A11,B11), MatrixMultiply(A12,B21))
   C12 = Add(MatrixMultiply(A11,B12), MatrixMultiply(A12,B22))
   C21 = Add(MatrixMultiply(A21,B11), MatrixMultiply(A22,B21))
```

C22 = Add(MatrixMultiply(A21,B12), MatrixMultiply(A22,B22))
return C

Recurrence: 8 multiplications of size n/2 + 4 additions of size n²/4 → $T(n) = 8T(n/2) + O(n^2)$
Solve: $a = 8, b = 2, p = \log_2 8 = 3$, $f(n) = O(n^2)$ → Case 1 → $T(n) = \Theta(n^3)$.
Same as naive algorithm. Strassen's reduces to 7 multiplications → $O(n^{2.81})$.

Other Important Question 10 – Sort the list "G,U,J,A,R,A,T,S,A,R,K,A,R" using Merge Sort. (3 marks)
Appeared in: W23 (Q3a, 03)
Ans:
List: G, U, J, A, R, A, T, S, A, R, K, A, R (13 elements)
Step-by-step Merge Sort:

1. **Split recursively until single elements.**
   Left half: [G,U,J,A,R,A,T] → further split → [G,U,J] [A,R,A,T] → ...
   Right half: [S,A,R,K,A,R]
2. **Merge back in sorted order** (shown below final merge only for brevity):

After full splitting and merging:
Final sorted list:
A, A, A, A, G, J, K, R, R, R, S, T, U
Verification: Count As: positions 1,2,3,4? Actually four As? List has A at indices 4,6,9,12?
Original: G,U,J,A,R,A,T,S,A,R,K,A,R → A appears 4 times. Sorted:
A,A,A,A,G,J,K,R,R,R,S,T,U.

Other Important Question 11 – Explain properties of Heap Tree. Sort given data using Heap Sort. (7 marks)
Appeared in: W23 (Q3c, 07)
Ans:
Note: Heap Sort is typically under "Sorting" but included here.
Properties of Heap Tree (Binary Heap):

- **Complete binary tree** – all levels filled except possibly last, filled left to right.
- **Heap property:**
  - **Max-heap:** Parent ≥ children (root is largest).
  - **Min-heap:** Parent ≤ children (root is smallest).
- Height = $\lfloor \log_2 n \rfloor$.
- Array representation: parent at i, left child at 2i+1, right at 2i+2 (0-based).
- Useful for priority queues, Heap Sort.

Heap Sort Algorithm:

1. Build max-heap from array (O(n) time).
2. Repeatedly extract root (largest) and swap with last element, then heapify reduced heap.

Example – Sort given data: *No specific data given in question. Provide generic example.*
Take array: [4, 10, 3, 5, 1]

- Build max-heap: [10, 5, 3, 4, 1]
- Swap root 10 with last → [1,5,3,4,10]; heapify [1,5,3,4] → [5,4,3,1]
- Swap 5 with last → [1,4,3,5,10]; heapify [1,4,3] → [4,1,3]
- Swap 4 with last → [3,1,4,5,10]; heapify [3,1] → [3,1]
- Swap 3 with last → [1,3,4,5,10] → sorted.

Complexity: $O(n \log n)$ time, $O(1)$ extra space.
Real-world application: Operating system process scheduling (priority queue), Dijkstra's

algorithm.

****