



AUTUMN MID SEMESTER EXAMINATION-2015

Design & Analysis of algorithm [CS-3001]

Full Marks: 25

Time: 2 Hours

Answer any four questions including question No.1 which is compulsory.

The figures in the margin indicate full marks.

Candidates are required to give their answers in their own words as far as practicable and all parts of a question should be answered at one place only.

DAA MID-SEM SOLUTION & EVALUATION SCHEME

Q1 Answer the following questions:

(2 x 5)

- a) Consider the following C function.

```
int fun(int n)
{
    int i, j, p=0;
    for(i = 1; i < n; ++i)
        for(j = n; j > 1; j = j/2)
            ++p;
    return p;
}
```

What is most closely approximates value returned by the function fun?

Scheme:

Correct answer : 2 Mark

Wrong answer, but explanation approaches to answer : Step Marking

Answer:

$(n-1)\log_2 n \Rightarrow n \log_2 n$

The outer loop i body will execute (n-1) times. For each outer i iteration the inner j loop will execute $k = \log n$ times ($2^k \leq n$) means k times the value of p will be incremented by 1.

- b) Rank the following functions by order of growth in increasing sequence?
 $\log n, \log \sqrt{n}, \sqrt{n}, n \log \sqrt{n}, n!, 2^n$

Scheme:

Arrangement of given functions with correct sequence (only answer) : 2 Mark

Answer:

The functions by order of growth in increasing sequence
 $\log \sqrt{n}, \log n, \sqrt{n}, n \log \sqrt{n}, 2^n, n!$

- c) Can the master method be applied to solve the following recurrence?

$$T(n) = 4T(n/2) + n^2 \log n$$

Justify your answer.

Scheme:

Correct answer with justification: 2 Mark

Wrong answer, but explanation approaches to answer : Step Marking

Answer:

Master theorem can not be applied to solve the given recurrence.

Explanation

Solving by master theorem

Step-1: Guess for which case of master theorem

$$\text{Given } a=4, b=2, f(n)=n^2 \log n$$

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

Comparing $n^{\log_b a}$ with $f(n)$, we found $f(n)$ is asymptotically faster growing function. So we guess the solution may exist in case-3.

Step-2: Conformation test for case-3

If case-3, then $f(n) = \Omega(n^{\log_b a + \epsilon})$

$$\Rightarrow f(n) \geq cn^{\log_b a + \epsilon}$$

$$\Rightarrow n^2 \log n \geq cn^{2+\epsilon}$$

$$\Rightarrow n^2 \log n \geq cn^{2+\epsilon}$$

As no value of $\epsilon > 0$ exists that makes the above inequality valid, so master theorem can not be applied to solve the given recurrence.

- d) Write the merge sort procedure (only) which divides the array into two parts such that first part contains elements twice of second part. Also derive the time complexity of that merge sort.

Scheme:

Writing the algorithm with partitioning index q as mentioned below : 1 Mark

Time complexity derivation: 1 Mark

Answer:

Let p and r are the lower and upper index of the array. q is the partitioning point that divides the array into two parts such that first part contains elements twice of second part. First Part Second Part

$$\begin{array}{ccc} p & q & r \end{array}$$

$$\text{So, } q-p+1=2(r-q+1) \Rightarrow q=(2r+p-1)/3$$

MERGE-SORT(A, p, r)

{

$$q \leftarrow (2r+p-1)/3$$

MERGE-SORT(A, p, q)

MERGE-SORT($A, q+1, r$)

MERGE(A, p, q, r)

}

The Recurrence relation of the above algorithm is

$$T(n) = T(n/3) + T(2n/3) + O(n)$$

Solving the unbalanced partitioning we will get the solution as **$T(n) = n \log_{3/2} n$**

e) Match the following Algorithms and its recurrences

Bubble-sort $T(n)=T(n/2)+ \Theta(1)$

Quick-sort $T(n)=T(n-1)+ \Theta(n)$

Merge-sort $T(n)=T(k) + T(n-k)+\Theta(n)$

Binary-Search $T(n)=2T(n/2)+ \Theta(n)$

Scheme:

Each correct matching : 0.5 Mark

Answer:

Bubble-sort $T(n)=T(n-1)+ \Theta(n)$

Quick-sort $T(n)=T(k) + T(n-k)+\Theta(n)$

Merge-sort $T(n)=2T(n/2)+ \Theta(n)$

Binary-Search $T(n)=T(n/2)+ \Theta(1)$

Q2 a) What is the significance of asymptotic notations? Define different asymptotic notations used in algorithm analysis. (2.5)

Scheme:

Significance of asymptotic notation : 0.5 Mark

Correct Definition of asymptotic notations => 2 Marks

Answer:

- We often want to know a quantity approximately, instead of exactly, in order to compare one algorithm with another, we introduce some terminology that enables us to make meaningful (but inexactness) statements about the time and space complexity of an algorithm, they are known as **asymptotic notations**.
- **Significance of Asymptotic Notations**
 - Asymptotic Notations are used to describe the running time of an algorithm in a meaningful way, is defined in terms of functions whose domains are the set of natural numbers.
 - These notations refer to how the problem scales as the problem gets larger.
 - The asymptotic run time of an algorithm gives a simple and machine independent, characterization of its complexity.
 - The notations works well to compare algorithm efficiencies because we want to say that the growth of effort of a given algorithm approximates the shape of a standard function.

- **Definitions of different asymptotic notations used in algorithm analysis**

There are five different notations commonly used in algorithm analysis. They are

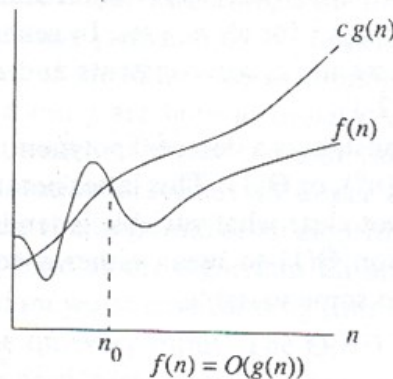
- i) O - Notation (Big-Oh Notation) : Asymptotic upper bound
- ii) Ω - Notation (Big-Omega Notation) : Asymptotic lower bound
- iii) Θ - Notation (Theta Notation) : Asymptotic tight bound
- iv) o - Notation (Little-oh Notation) : Upper bound that is not asymptotically tight
- v) ω - Notation (Little-omega Notation): Lower bound that is not asymptotically tight

i) O - Notation (Big-Oh Notation)

- It represents the upper bound of the resources required to solve a problem.(worst case running time)
- **Definition:** Formally it is defined as
For any two functions $f(n)$ and $g(n)$, which are non-negative for all $n \geq 0$, $f(n)$ is said to be $g(n)$, $f(n) = O(g(n))$, if there exists two positive constants c and n_0 such that

$$0 \leq f(n) \leq c g(n) \text{ for all } n \geq n_0$$

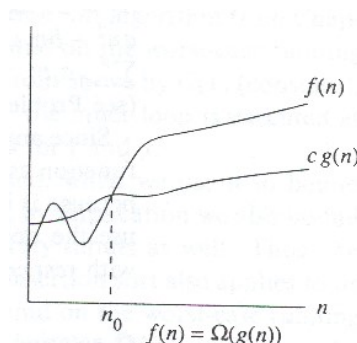
- Less formally, this means that for all sufficiently big n , the running time of the algorithm is less than $g(n)$ multiplied by some constant. For all values n to the right of n_0 , the value of the function $f(n)$ is on or below $g(n)$.



ii) Ω - Notation (Big-Omega Notation)

- **Definition:** For any two functions $f(n)$ and $g(n)$, which are non-negative for all $n \geq 0$, $f(n)$ is said to be $g(n)$, $f(n) = \Omega(g(n))$, if there exists two positive constants c and n_0 such that

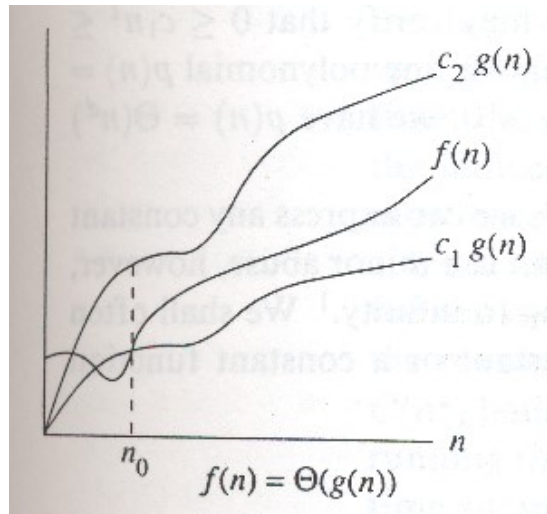
$$0 \leq c g(n) \leq f(n) \text{ for all } n \geq n_0$$



iii) Θ - Notation (Theta Notation)

- Definition:** For any two functions $f(n)$ and $g(n)$, which are non-negative for all $n \geq 0$, $f(n)$ is said to be $\Theta(g(n))$, if there exists positive constants c_1 , c_2 and n_0 such that

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0$$



iv) o - Notation (Little-oh Notation)

For any two functions $f(n)$ and $g(n)$, which are non-negative for all $n \geq 0$, $f(n)$ is said to be little oh of $g(n) \Rightarrow f(n) = o(g(n))$, for any positive constant $c > 0$, if there exists a positive constant $c > 0$ such that $0 \leq f(n) < c g(n)$ for all $n \geq n_0$.

v) ω - Notation

For any two functions $f(n)$ and $g(n)$, which are non-negative for all $n \geq 0$, $f(n)$ is said to be little omega of $g(n) \Rightarrow f(n) = \omega(g(n))$, for any positive constant $c > 0$, if there exists a positive constant $c > 0$ such that $0 \leq c g(n) < f(n)$ for all $n \geq n_0$.

b) Solve the following recurrence.

(2.5)

$$T(n) = 4T(n/2) + n^2 \text{ where } n > 1 \text{ and } T(1) = 1$$

Scheme:

Correct Answer: 0.5 Mark

Correct Answer with explanation :2 Mark

Answer:

Solving the recurrence $T(n) = 4T(n/2) + n^2$ where $n > 1$ and $T(1) = 1$, by master theorem.

Step-1: Given $a=4$, $b=2$, $f(n) = n^2$

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

Comparing both $n^{\log_b a}$ and $f(n)$, we found both are same. So as per Master theorem we guess the solution may exist in case-2

Step-2: Confirmation test for case-2

If case-2, then $f(n) = \theta(n^{\log_b a})$

$$\Rightarrow c_1 n^{\log_b a} \leq f(n) \leq c_2 n^{\log_b a}$$

$$\Rightarrow c_1 n^2 \leq n^2 \leq c_2 n^2$$

The above inequality is valid for some constant $c_1=1, c_2=2, n_0=1$, so case-2 is confirmed. As per case-2 the solution is

$$T(n) = \theta(n^{\log_b a} \log_2 n) = \theta(n^2 \log_2 n)$$

- Q3 a) Write the PARTITION() algorithm of Quick Sort and describe step by step how you would get the pass1 result by taking last element as pivot on the following data. (3)

8, 2, 1, 5, 6, 1, 3, 7, 4, 9, 5

Derive the average case time complexity of quick sort.

Scheme:

PARTITION Algorithm: 1 Mark

Representation of intermediate steps of pass-1 : 1 Mark

Average case time complexity derivation : 1 Mark

Answer:**PARTITION Algorithm:**

PARTITION(A,p,r)

```
{
  x ← A[r]
  i ← p-1
  for j ← p to r-1
  {
    if A[j] ≤ x
    {
      i ← i+1
      A[i] ↔ A[j]
    }
  }
  i ← i+1
  A[i] ↔ A[r]
  return i
}
```

Representation of intermediate steps of pass-1

Given Data: 8, 2, 1, 5, 6, 1, 3, 7, 4, 9, 5

i	j=p									r	
	8	2	1	5	6	1	3	7	4	9	5

i	p	j								r
8	2	1	5	6	1	3	7	4	9	5

i	p			j								r
2	8	1	5	6	1	3	7	4	9	5		

	p	i	j									r
2	8	1	5	6	1	3	7	4	9	5		

	p		i		j							r
2	1	8	5	6	1	3	7	4	9	5		

	p			i	j							r
2	1	8	5	6	1	3	7	4	9	5		

	p				i		j					r
2	1	5	8	6	1	3	7	4	9	5		

	p					i		j				r
2	1	5	8	6	1	3	7	4	9	5		

	p						i		j			r
2	1	5	1	6	8	3	7	4	9	5		

	p							i		j		r
2	1	5	1	6	8	3	7	4	9	5		

	p								i		j		r
2	1	5	1	3	8	6	7	4	9	5			

	p									i		j		r
2	1	5	1	3	8	6	7	4	9	5				

	p										i		j	r
2	1	5	1	3	4	6	7	8	9	5				

	p											i		rj
2	1	5	1	3	4	6	7	8	9	5				

	p												i	rj
2	1	5	1	3	4	5	7	8	9	6				

Average case time complexity derivation :

Average case time complexity of quick sort is $O(n \log_2 n)$

- b) Given 10 activities, $A = \langle a_1, a_2, \dots, a_n \rangle$ along with their start time (s_i) and finish time (f_i) as $S_i = \langle 1, 2, 4, 3, 7, 7, 8, 9, 11, 12 \rangle$ and $f_i = \langle 3, 5, 4, 7, 5, 9, 14, 18, 10, 12 \rangle$ and that requires the exclusive use of a common stage for scheduling these activities. Use an efficient method that computes a schedule with largest number of activities on that stage. (2)

Scheme:

Some values are incorrectly written. Identifying these values and correct the same by taking some assumptions (reverse the timings etc) : 1 Mark

Sorting the activities in their increasing order of their finishing time and finding out the solution : 1 Mark

Answer:

The activities in their increasing order of their finishing time (Assuming the starting time is less than finishing time in case any incorrect value is written, reverse the timings)

Activities (a_i)	Starting Time (s_i)	Finishing Time (f_i)	Accept/ Reject
a1	1	3	√
a3	4	5	√
a2	2	5	X
a4	3	7	X
a5	5	7	√
a6	7	9	√
a9	10	11	√
a10	12	12	√
a7	8	14	X
a8	9	19	X

The solution is $\langle a_1, a_3, a_5, a_6, a_9, a_{10} \rangle$

- Q4 a) Find an optimal solution to the knapsack instance $n=7$, $W=15$. ($v_1, v_2, v_3, v_4, v_5, v_6, v_7$) = (10, 5, 15, 7, 6, 18, 3) and ($w_1, w_2, w_3, w_4, w_5, w_6, w_7$) = (2, 3, 5, 7, 1, 4, 1), where n is the number of items, W is the knapsack capacity that thief can carry, v_i stands for value or profit w_i stands for weight of the i^{th} element. (2)

Scheme:

Finding value per weight of each items and arrange them in decreasing order:
1Mark

Finding the solution vector or the optimal solution : 1 Mark

Answer:

Step-1 (Finding value per weight of each items)

tem/ Object (i)	Value/Profit (v_i)	Weight (w_i)	Value per weight (v_i/w_i)
1	10	2	5
2	5	3	1.66
3	15	5	3
4	7	7	1
5	6	1	6
6	18	4	4.5
7	3	1	3

Step-2 (sorting the items in decreasing order of their value per weight)

U=W=15

Item/ Object (i)	Value/Profit (v_i)	Weight (w_i)	Value per weight (v_i/w_i)	Weight taken	x_i
5	6	1	6	$1 \leq 15$ $U=15-1=14$	1
1	10	2	5	$2 \leq 14$ $U=14-2=12$	1
6	18	4	4.5	$4 \leq 12$ $U=12-4=8$	1
3	15	5	3	$5 \leq 8$ $U=8-5=3$	1
7	3	1	3	$1 \leq 3$ $U=3-1=2$	1
2	5	3	1.66	$3 > 2$ Fraction=2/3	2/3
4	7	7	1	-	0

The **solution vector or the optimal solution** is

$(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 2/3, 1, 0, 1, 1, 1)$

$$\begin{aligned}
 \text{Profit earned} &= \sum_{i=1}^7 v_i x_i \\
 &= 1 \times 10 + 2/3 \times 5 + 1 \times 15 + 0 \times 7 + 1 \times 6 + 1 \times 18 + 1 \times 3 \\
 &= 10 + 3.33 + 15 + 0 + 6 + 18 + 3 = 55.33
 \end{aligned}$$

- b) Sort the following data in descending order by using heap sort technique. (2)
50, 12, 28, 100, 18, 35, 75, 86, 15, 80

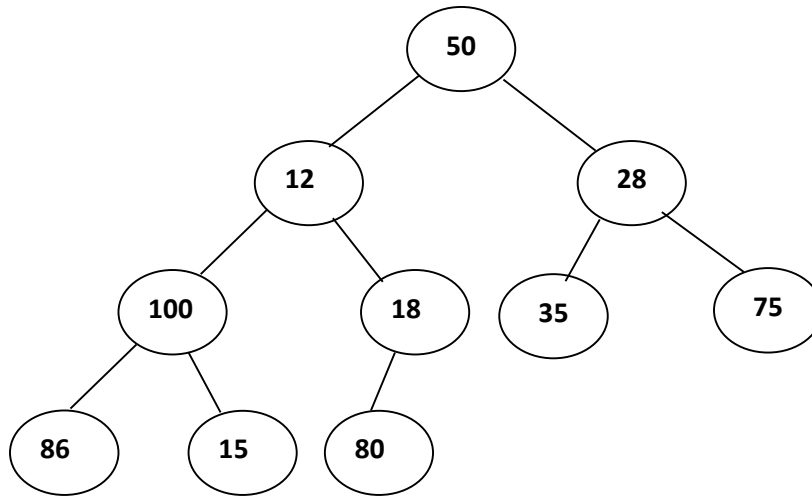
Scheme:

Showing the intermediate steps to build a Min Heap : 1 Mark

Showing some initial steps and drawing the final figure to sort by heap sort : 1 Mark

Answer:

Step-0: Constructing the heap structure with the data given as follows
50, 12, 28, 100, 18, 35, 75, 86, 15, 80



Step-1: Building a min-heap by applying MIN-HEAPIFY from index of element 18 down to the index of the root element.

- c) Consider a complete binary tree where the left and the right subtrees of the root are max-heaps. What is upper bound to convert the tree onto a heap? (1)

Scheme:

Only answer : 1 Mark

Answer:

$O(\log_2 n)$

Explanation

In the worst case applying MAX-HEAPIFY(1) will terminate at leaf that visits the heights of the tree.

- Q5 a) Write an algorithm to find out two elements from an array having non-negative numbers such that the difference should be maximum. (3)

Scheme:

Correct algorithm : 3 Mark

Answer:

MAX-TWO-ARRAY(A, p, r)

```
{
    max=min=a[p]
    for i ← p+1 to r
    {
        if a[i]> max
            max ← a[i]
        if a[i] < min
            min ← a[i]
    }
    return(max, min)
}
```

- c) An unordered list contains n distinct elements. What is the minimum number of comparisons required to find an element in this list that is neither maximum nor minimum? (2)

A. $O(1)$ B. $O(\log n)$ C. $O(n \log n)$ D. $O(n)$

Scheme:

Correct option : 2 Marks

Answer:

Option-A

Explantion

Comparing first three elements to get max and min. The max and min may be the final max or min value. But the middle element that is in between max and min is the element that is neither max nor min.

So to get this we require two comparisons that is $O(1)$ constant time.

=====