SPRING MID SEMESTER EXAMINATION-2018

Design & Analysis of Algorithms

[CS-2008]

Full Marks: 25 **Time: 1.5 Hours**

Answer any five 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 SOLUTION & EVALUATION SCHEME

Q1 Answer the following questions: **(1 x 5)**

a) Define Big-Omega nonation.

Scheme:

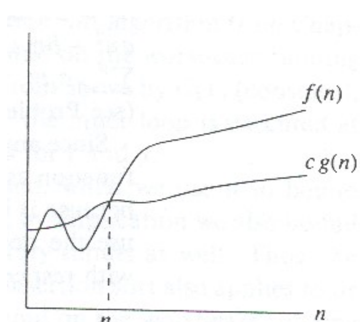
- Correct definition: 1 Mark
- Incorrect definition, but valid diagram : 0.5 Mark

Answer:

Big-Omega Notation (Ω - Notation)

Definition: For any two functions f(n) and g(n), which are non-negative for all n ≥ 0, f(n) is said to be g(n), f(n) = Ω (g(n)), if there exists two positive constants c and n0 such that

$$0 \le cg(n) \le f(n) \text{ for all } n \ge n0$$



$f(n) = \Omega(g(n))$

b) Represent the execution time of the following code in O-Notation (least upper bound).

```
int fun(int n)
{
    int i = 1, sum = 1;
    while (sum ≤ n)
    {
       i++;
       sum = sum + i;
    }
```

return sum;
}

Scheme:

- Correct answer : 1 Mark
- Other answer: 0 Mark

Answer:

O(√n)

c) **Suppose we are sorting an array of eight integers using quick sort and we have just finished the first partitioning with the array looking like this**

2, 5, 1, 7, 9, 12, 11, 10

Which of the following statement is correct?

i. The pivot could be either the 7 or the 9

ii. The pivot could be 7, but not 9

iii. The pivot is not the 7, but it could be the 9

iv. Neither the 7 nor the 9 is pivot

Scheme:

- Correct answer : 1 Mark
- Other answer: 0 Mark

Answer:

i.The pivot could be either the 7 or the 9

d) **In KBC, Abhishek Bachan writes down a number between 1 and 1000. Amitav Bachan must identify that number by asking “yes/no” question to Abhishek. If Amitav Bachan uses an optimal strategy then exactly how many “yes/no” questions should he ask to determine the answer at the end in worst case?**

Scheme:

- Correct answer : 1 Mark
- Incorrect answer, but strategy mentioned as binary search: 0.5 Mark
- Incorrect answer: 0 Mark

Answer:

10

e) **An array is intially sorted. When new elements are added, they are inserted at the end of the array and counted. Whenever the number of elements reached 20, the array is resorted and counter is cleared. Which sorting algorithm would be good choice to use for resorting the array?**

Scheme:

- Correct answer : 1 Mark
- Other answer: 0 Mark

Answer:

Insertion Sort

Q2 **Describe the step count and asymptotic efficiency of algorithms with INSERTION-SORT as an example.** **(5)**

<u>Scheme:</u>

- Correct algorithm: 2.5 Mark
- Explantion of time complexity
  by step count & asymptotic notation: 2.5 Marks
- Partial correct algorithm with some valid explanation: 0.5 to 2.5 Marks

<u>Answer:</u>

Line No.	Insertion Sort Algorithm	Cost	Times
1	INSERTION-SORT(A)	0	
2	{	0	
3	for j←2 to length[A]	c1	n
4	{	0	
5	key←A[j]	c2	n-1
6	//Insert A[j] into the sorted sequence A[1..j-1]	0	
7	i←j-1	c3	n-1
8	while(i>0 and A[i]>key)	c4	$\sum_{j=2}^{n} tj$
9	{	0	
10	A[i+1]←A[i]	c5	$\sum_{j=2}^{n} (tj-1)$
11	i←i-1	c6	$\sum_{j=2}^{n} (tj-1)$
12	}	0	
13	A[i+1]←key	c7	n-1
14	}	0	
15	}	0	

To compute T(n), the running time of INSERTION-SORT on an input of n values, we sum the products of the cost and times column, obtaining

$T(n)=c1n + c2(n-1) + c3(n-1) + c4\sum_{j=2}^{n} tj + c5\sum_{j=2}^{n}(tj-1) + c6\sum_{j=2}^{n}(tj-1) + c7(n-1)$ ………… (1)

<u>Best case Analysis:</u>

- Best case occurs if the array is already sorted.
- For each j=2 to n, we find that A[i]≤key in line number 8 when i has its initial value of j-1. Thus tj=1 for j=2 to n and the best case running time is
  $T(n)=c1n + c2(n-1) + c3(n-1) + c4(n-1) + c7(n-1) =O(n)$

<u>Worst case Analysis:</u>

- Worst case occurs if the array is already sorted in reverse order
- In this case, we compare each element A[j] with each element in the entire sorted subarray A[1..j-1], so tj=j for j=2 to n.
- The worst case running time is
  $T(n)=c1n + c2(n-1) + c3(n-1) + c4(n(n+1)/2-1) + c5(n(n-1)/2) + c6(n(n-1)/2) +c7(n-1)=O(n^2)$

Q3 **State and explain Master's Theorem and solve the recurrence.** **(5)**

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

<u>Scheme:</u>

- Explanation of master theorem : 2.5 Marks
- Finding solution of recurrence with proper steps: 2.5 marks

<u>Answer:</u>

Type:-1 (Master Theorem as per CLRS)	**Solution of recurrence** **$T(n) = 2T(n/4) + n^2$**
The Master Theorem applies to recurrences of the following form: $T(n) = aT(n/b) + f(n)$ where $a \geq 1$ and $b > 1$ are constants and f(n) is an asymptotically positive function. T(n) is defined on the non-negative integers by the recurrence. T(n) can be bounded asymptotically as follows: There are 3 cases: a) **Case-1:** If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then **$T(n) = \Theta(n^{\log_b a})$** b) **Case- 2:** If $f(n) = \Theta(n^{\log_b a})$, then **$T(n) = \Theta(n^{\log_b a} \log n)$** c) **Case-3:** If $f(n) = \Omega(n^{\log_b a+\epsilon})$ with $\epsilon > 0$, and : $af(n/b) \leq cf(n)$, then $T(n) = \Theta(f(n))$, for some constant $c < 1$ and all sufficiently large n, then **$T(n) = \Theta(f(n))$**	Here, a=2, b=4, $f(n)=n^2$ $n^{\log_b a} = n^{\log_4 2} = n^{0.5}$ Step-1:Comparing $n^{\log_b a}$ with f(n), we found f(n) is asymptotically larger larger than n^2. So we guess case-3 of master theorem. Step-2: As per case-3, $f(n)=\Omega(n^{\log_b a+\epsilon})$ must be satisfied first. Let it be true. $f(n) = \Omega(n^{\log_b a+\epsilon})$ => $f(n) \geq c*n^{\log_b a+\epsilon}$ => $n^2 \geq c*n^{0.5+\epsilon}$ ---------------- (1) Taking c=1 and ϵ=0.5, the above inequality is valid for n_0=1 Now, testing $af(n/b) \leq cf(n)$ for some constant $c < 1$ true or not? $af(n/b) \leq cf(n)$ let it be true => $2f(n/4) \leq cf(n)$ =>$2*n^2/4 \leq c*n^2$ =>$0.5 \leq c$ ------------------------ (2) This enequality is true As both eq (1) & (2) are found true, so as per case-3 of master theorem, **$T(n)=\Theta(f(n))=\Theta(n^2)$**

Type:-2 (Master Theorem)	Solution of recurrence $T(n) = 2T(n/4) + n^2$
If the recurrence is of the form $T(n) = aT(n/b) + n^k \log^p n$, where $a \geq 1$, $b > 1$, $k \geq 0$ and p is a real number, then comapre a with b^k and conclude the solution as per the following cases. **Case-1:** If $a > b^k$, then $T(n) = \Theta(n^{\log_b a})$ **Case-2:** If $a = b^k$, then a) If $p > -1$, then $T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$ b) If $p = -1$, then $T(n) = \Theta(n^{\log_b a} \log\log n)$ c) If $p < -1$, then $T(n) = \Theta(n^{\log_b a})$ **Case-3:** If $a < b^k$, then a) If $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$ b) If $p < 0$, then $T(n) = \Theta(n^k)$	Here, a=2, b=4, k=2, p=0 $b^k = 4^2 = 16$ Comparing a with b^k, we found a is smaller than b^k., so this will fit to case-3. Now p=0, so case-3.a solution is the recurrence solution. $\mathbf{T(n)= \Theta(n^2 \log^0 n)=\Theta(n^2)}$

Q4 **Explain Divide-and-Conquer approach and apply this approach to design an Algorithm to calculate x^n in $O(\log_2 n)$ time, where x is a real number and n is a non-negative integer.** **5)**

Scheme:

- Explanation of Divide-and-Conquer approach: 2 Marks
- Correct algorithm to calculate x^n in $O(\log_2 n)$ time : 3 Marks
- Correct algorithm to calculate x^n in $O(n)$ time : 1.5 Marks

Answer:

Divide and Conquer (D-n-C) Algorithm

- In this approach, the problem is broken into several sub problem that are similar to the original problem but smaller in size, the sub problems are solved recursively, and then these solutions are combined to create a solutions to the original problem.
- The divide and conquer paradigm involves 3 steps at each level of the recursion.

  Divide the problem into a number of sub problems.

  Conquer the sub problems by solving them recursively. If the sub problem sizes are small enough, however, just solve the sub problems in a straight forward manner.

  Combine the solutions to the problems into the solution for the original problem.

C-Function/Algorithm to calculate x^n

```
int POWER( int x, int n)
{
     if ( n==0)
         return 1;
    else if(n==1)
         return x;
    else if (n%2==0)
         return POWER( x * x, n/2);
    else
         return POWER( x * x, n/2 ) * x ;
}
```

Q5 a) **Write the PARTITION() algorithm of Quick-Sort. Describe in a step by step process to get the pass1 result by applying PARTITION() algorithm by taking 6th element (underlined) as pivot on the following array elements. <5, 4, 3, 6, 2, 5, 9, 8, 7, 3, 6, 1>** **(5)**

Scheme:

- Correct PARTITION() algorithm : 2.5 Marks
- Step by step explanation to get the pass1 result : 2.5 marks

Answer:

RANDOMIZED-PARTITION(A, p, r) { /*Generate a random number within a range p to r*/ i ← RANDOM(p, r); /*Swap ith indexed element with last elemnt*/ A[i] ↔ A[r]; return PARTITION(A, p, r); }	PARTITION(A, p, r) { x ← A[r]; //last elemnt is pivot i ← p-1; for j←p ro r-1 { if(A[j] ≤ x) { i ← i +1; A[i] ↔ A[j];//Intermediate swap } } i ← i +1; A[i] ↔ A[r]; //Final swap return i; }

Step by step explanation to get the pass1 result by applying PARTITION() algorithm by taking 6th element (underlined) as pivot on the following array elements. <5, 4, 3, 6, 2, 5, 9, 8, 7, 3, 6, 1>

Underlined means swapping of data

	p											r
0)	5	4	3	6	2	**5**	9	8	7	3	6	**1**
i	j=p											r
	5	4	3	6	2	1	9	8	7	3	6	5
	i	j										
a)	**5**	**4**	3	6	2	1	9	8	7	3	6	5
	i		j									
	4	5	3	6	2	1	9	8	7	3	6	5
		i	j									
b)	4	**5**	**3**	6	2	1	9	8	7	3	6	5
		i		j								
	4	3	5	6	2	1	9	8	7	3	6	5
			i		j							
c)	4	3	**5**	6	**2**	1	9	8	7	3	6	5
			i			j						
	4	3	2	6	5	1	9	8	7	3	6	5
				i		j						
d)	4	3	2	**6**	5	**1**	9	8	7	3	6	5
				i			j					
	4	3	2	1	5	6	9	8	7	3	6	5
					i					j		
e)	4	3	2	1	**5**	6	9	8	7	**3**	6	5
					i						j	
	4	3	2	1	3	6	9	8	7	5	6	5
						i						j=r
f)	4	3	2	1	3	**6**	9	8	7	5	6	**5**
Pass-1	4	3	2	1	3	**5**	9	8	7	5	6	6
			First Half						Second Half			
						Pivot Element						

Q6 **We are given an array of n elements, where first 1/3rd of the array elements are in ascending order and rest of the array elements are in descending order. Write an Algorithm to sort the array in O(n) time.**

<u>Scheme:</u>

- Correct algorithm to sort array in O(n) time: 5 marks
- Algorithm to sort arrayother than O(n) time: 2.5 marks

<u>Answer:</u>

/*Array A contains 1/3rd of array elements from index p to q in ascending order, rest from q+1 to r in descending order.*/

```
MERGE(A, p, q, r)
{
   //Create a temporary array TA having (r-p+1) elements
   //Array A divided into two parts (i.e. Left array A is sorted in ascending from index p to q and right array A is sorted from index r to q+1
    i←p, j←r
    k←0
    //Compare one element of left part of array A with one element of right part of array A one by one and do the following till each part has at least one element
    while (i≤q and j≥q+1)
    {
        if (A[i]<A[j])
        {
           TA[k]=A[i]
           i←i+1
           k←k+1
        }
        else
        {
          TA[k]=A[j]
           j←j-1
           k←k+1
          }
     }
   //Copy rest elements if any from left array to temporary array TA
    while(i≤q)
   {
          TA[k]=A[i]
           i←i+1
           k←k+1
     }
```

```
//Copy rest elements if any from right array to temporary array TA
while(j≥q+1)
 {
     TA[k]=A[j]
      j←j-1
      k←k+1
 }
 //Copy all elements from temporary array TA back to array A
 k←0
 for i← p to r
 {
    A[i]←TA[k]
     k←k+1
 }
}
```

Q7 **Write an algorithm to find the majority element in an array in O(n) time. The majority is an element that occurs for more than half of the size of the array. For example, the number 2 in the array {1, 2, 3, 2, 2, 2, 5, 4, 2} is the majority element because it appears five times and the size of the array is 9.** **(5)**

Scheme:

- Correct algorithm/C-function solved with O(n) time: 5 Marks
- Correct algorithm/C-function solved with O($n\log_2 n$) time: 3.5 Marks
- Correct algorithm/C-function solved with O(n^2) time: 2.5 Marks
- Incorrect algorithm, but some valid correct steps: 0.5 to 2.5 Marks
- Incorrect algorithm: 0 Mark

Sample Answer:

Solution-1

<table>
<tr><td colspan="2">/*Finding maximum frequency in an array of n numbers*/</td></tr>
<tr><td>MAX-FREQUENCY(A, n)
{
  /*Find the maximum element of array*/
   max←FIND-MAX(A, n)
   /*create a count array named as COUNT with max+1 elements and initilize to zero*/
  for i ← 0 to max
  {
    COUNT[i] ← 0</td>
<td> /*Checking the frequency of me more than half of array elements or not*/
  if (maxelefrq > n/2)
     return maxelemt;
  else
      return -1;   //Indication for no majority of elements.
  }
}</td></tr>
</table>

} for i ← 0 to n-1 { COUNT[A[i]]←COUNT[A[i]]+1 ; } **/*Finding max. element in COUNT array*/** maxelefrq ← COUNT[0] maxelmt← 0 for i ← 1 to max { if(COUNT[i] >maxelefrq) { maxelefrq ← COUNT[i] maxelmt ← i } }	

Solution-2

Concept: Basic idea of the algorithm is if we cancel out each occurrence of an element e with all the other elements that are different from e then e will exist till end if it is a majority element.

```
/*C-Function to find majority of elements*/
int findMajorityElement (int a[], int n)
{
   int count, me,i;
   me=a[0]; //assume first element qualifies for majority element
   count=1;
   for(i=1; i<n; i++)
   {
      if(me==a[i])
      {
         count ++;
      }
      else
      {
         if(count==0)
         {
            me = a[i];
```

```
            count = 1;
        }
        else
            count--;
      }
   }
  /*Counting frequency of element me*/
  count = 0;
  for (i = 0; i <n; i++)
  {
     if (a[i] == me)
        count++;
  }
  /*Checking the frequency of me more than half of array elements or not*/
  if (count > n/2)
       return me;
  else
        return -1;    //Indication for no majority of elements.
}
```

============XXXXXXXX============