SPRING MID SEMESTER EXAMINATION-2019

Design & Analysis of Algorithms

[CS-2008]

Full Marks: 20 **Time: 1.5 Hours**

Answer any <u>four</u> 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 <u>all parts of a question should be answered at one place only</u>.

DAA SOLUTION & EVALUATION SCHEME

Q1 Answer the following questions: (1 x 5)

a) Define Ω–Notation.

<u>Scheme:</u>

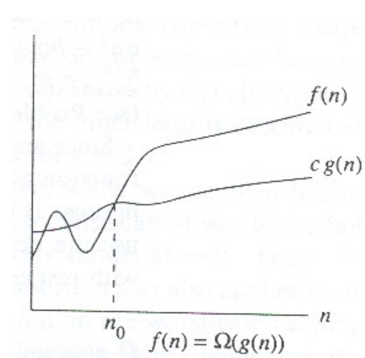
- Correct definition : 1 mark
- No proper definition, but proper explanation through example: 0.5 mark
- Any other: 0 mark

<u>Answer:</u>

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 omega of g(n), $f(n) = \Omega(g(n))$, if there exists two positive constants c and n0 such that

$$0 \leq cg(n) \leq f(n) \text{ for all } n \geq n0$$



b)
```
int fun ()
{
    int i, s = 0;
    for (i  = 1; i ≤1024; i=2*i)
        s=s+i;
    return s;
}
```

What is the time complexity of the above function fun?

A) Θ(1) B) Θ(log n) C) Θ(n) D) Θ(nlog n)

Answer:
A

Scheme:
- Correct answer : 1 mark
- Wrong answer or any other explanation: 0 Mark

c)
```
int fun1(int n)
{
     int i, j, k = 0;
    for (i  = n; i ≥ 1; i=i/2)
        for (j = 1; j ≤ i; j++)
              k = k + 1/n;
     return k;
 }
```
What is the returned value of the above function fun1?

A) $\Theta(1)$ B) $\Theta(\log n)$ C) $\Theta(n)$ D) $\Theta(n\log n)$

Answer:
A

Scheme:
- Correct answer : 1 mark
- Wrong answer or any other explanation: 0 Mark

d) The performance of merge sort depends on which type of data?

A) Increasing order data
B) Decreasing order data
C) 50% data are increasing and 50% data are decreasing
D) Does not depend upon the data set.

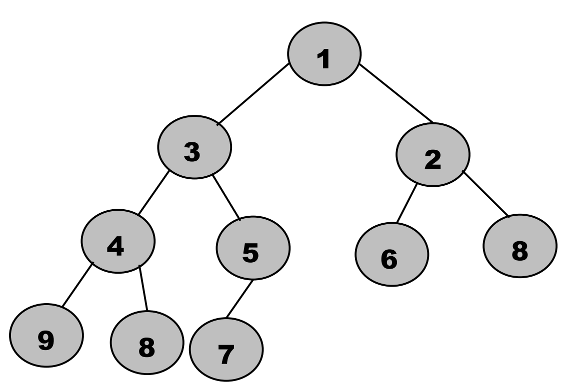
Answer:
D

Scheme:
- Correct answer : 1 mark
- Wrong answer or any other explanation: 0 Mark

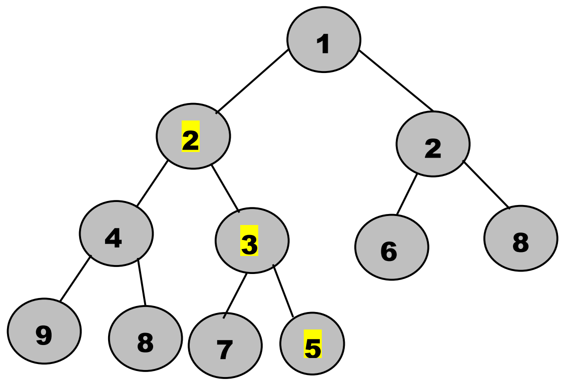
e) Find out the min-heap (show in diagram) evolved, after inserting 2 and 0 in that order into the heap={1, 3, 2, 4, 5, 6, 8, 9, 8, 7}.

Scheme:
- Correct answer (Final diagram) : 1 mark
- Partial Correct: 0.5 mark
- Incorrect answer: 0 mark

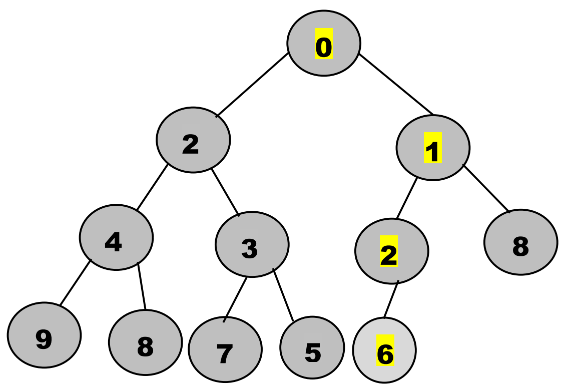
Answer:
Given heap



Inserting key 2



Inserting key 0



(Final Min-Heap)

Q2 Write a sorting algorithm for the following elements stored in a n-element array A such that the execution time will be least. The array A is given as, A ={1, 2, 3, 4, 5, 6, …….., 99, 100 }, if n=100. Find the execution time in asymptotic notation. (5)

Scheme:

- Correct insertion sort/bubble sort algorithm or any other algorithm with execution time O(n) : 5 marks
- Correct insertion sort/bubble sort algorithm without mentioning execution time as O(n) : 4 marks
- Other sorting algorithm with more than O(n) time : step marks (0 to 1 mark)

Answer:

/*Insertion Sort Algorithm*/

```
INSERTION-SORT(A, n)
{
   for (j←2 to n)
   {
      key←A[j]
      //Insert key into the sorted sequence A[1..j-1]
      i←j-1
```

```
            while(i>0 and A[i]>key)
            {
                A[i+1]←A[i]
                i←i-1
            }
            A[i+1]←key
        }
}
```

Q3 Solve the following recurrences (5)

a) $T(n) = 2T(\sqrt{n}) + 1$, $T(1)=1$

b) $T(n) = 4T(n/2) + n \log_2 n$, $T(1)=1$

Scheme:

- Correct answer with proper explanation by any method : 2.5 mark (each)
- Partial Correct: step marks (0.5 to 2 marks)
- Wrong answer with no proper approach: 0 mark

Answer:

Q.3.a => $T(n)=\Theta(\log_2 n)$

Given Recurrence, $T(n) = 2T(\sqrt{n}) + 1$ ………………(1)

Let $m=\log_2 n \Rightarrow n=2^m \Rightarrow n^{1/2}= 2^{m/2}$

Substituting these values in equation (1),

$T(2^m) = 2T(2^{m/2}) + 1$ …………………………………..(2)

Let rename $S(m) = T(2^m)$, substituting these values in equation (2), it produces the new recurrence as follows:

$S(m) = 2S(m/2) + 1$ …………………………………….(3)

Recurrence of equation (3) is similar to master theorem. As per master theorem,

a	b	k	p	b^k	a ® b^k	Case	Solution
2	2	0	0	1	>	1	$S(m)=\Theta(m^{\log_b a}) = \Theta(m^{\log_2 2})=\Theta(m)$ $\Rightarrow T(2^m)=\Theta(m)$ **$\Rightarrow T(n)=\Theta(\log_2 n)$**

Q.3.b => $T(n)=\Theta(n^2)$

Given Recurrence, $T(n) = 4T(n/2) + n \log_2 n$, $T(1)=1$

Type:-1 (Master Theorem as per CLRS)	**Solution of recurrence** **$T(n) = 4T(n/2) + n \log_2 n$**
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	Here, a=4, b=2, $f(n)=n \log_2 n$ $n^{\log_b a} = n^{\log_2 4} = n^2$ Step-1:Comparing $n^{\log_b a}$ with f(n), we found $n^{\log_b a}$ is asymptotically larger larger than n^2. So we guess case-1 of master theorem. Step-2: As per case-1,

follows: There are 3 cases: **a) Case-1:** If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $\mathbf{T(n) = \Theta(n^{\log_b a})}$ **b) Case- 2:** If $f(n) = \Theta(n^{\log_b a})$, then $\mathbf{T(n) = \Theta(n^{\log_b a} \log n)}$ **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 $\mathbf{T(n) = \Theta(f(n))}$	$f(n)=O(n^{\log_b a-\epsilon})$ must be satisfied first. Let it be true. $f(n) = O(n^{\log_b a-\epsilon})$ => $f(n) \leq c^* n^{\log_b a-\epsilon}$ => $n \log_2 n \leq c^* n^{2-\epsilon}$ ---------------- (1) Taking $c=1$ and $\epsilon=0.5$, the above inequality is valid for $n_0=1$ So as per case-1 of master theorem, $\mathbf{T(n)=\Theta(n^{\log_b a})=\Theta(n^2)}$

OR

Type:-2 (Master Theorem)	**Solution of recurrence** $\mathbf{T(n) = 4T(n/2) + n \log_2 n}$
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=4, b=2, k=1, p=1 $b^k = 2^1 = 2$ Comparing a with b^k, we found a is greater than b^k, so this will fit to case-1. As per Case-1 solution is the recurrence solution. $\mathbf{T(n)= \Theta(n^{\log_b a})=\Theta(n^2)}$

Q4 In a social gathering, there are m boys and n girls ($m > n$) of different ages. You have two unsorted arrays giving their ages (one for the boys, the other for the girls). Devise an efficient $O(m \log n)$ algorithm to find out the ages that are common between both the boys and girls. (5)

Scheme:

- Correct algorithm : 5 mark
- Partial Correct: step marks (0.5 to 4 marks)

Answer:

```
COMMON-AGE (B, m, G, n)
{
   //Sort the smallest array G with n elements
   MERGE-SORT(G, n);    ........................................... O(n log2 n)
   //Search each boy's age in girls' array  ⎫
   for (i←1 to m)                            ⎪
   {                                         ⎬ ........................ O(m log n)
      k←BINARY-SEARCH(G, n, B(i));           ⎪
      if (k!=-1)                             ⎭
        print G[k];
   }
}
```

Time Complexity=O(n $\log_2$ n) + O(m log n) = O(m log n) as m>n

Q5 Write an algorithm MAX-HEAP-CHANGE-KEY(A, n, i, key), to re-build a n-element max-heap A, after the value at node with index i has been changed to a new value key. Illustrate the operation of MAX-HEAP-CHANGE-KEY(A, 12, 2, 2) on the heap A={15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1}. Assume root is at index 1. (5)

Scheme:

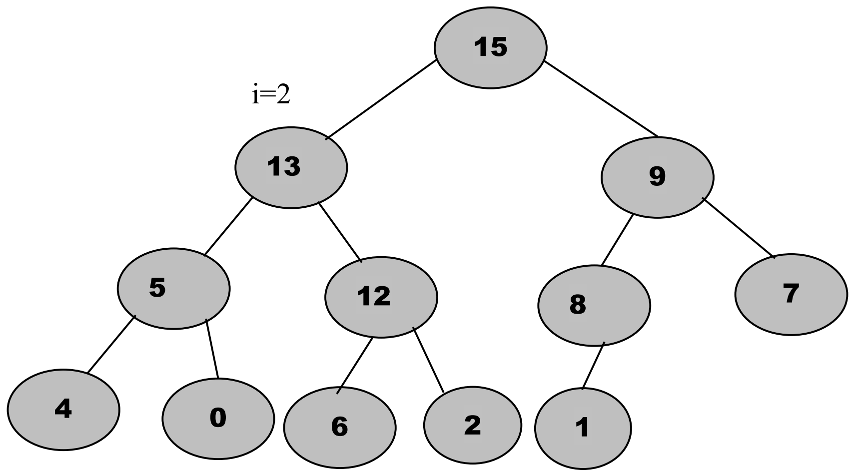
- Correct algorithm answer : 3 marks
- Illustration operation of MAX-HEAP-CHANGE-KEY(A, 12, 2, 2): 2 marks
- Partial Correct: step marks (0.5 to 3 marks)
- Wrong answer with no proper approach: 0 mark

Answer:

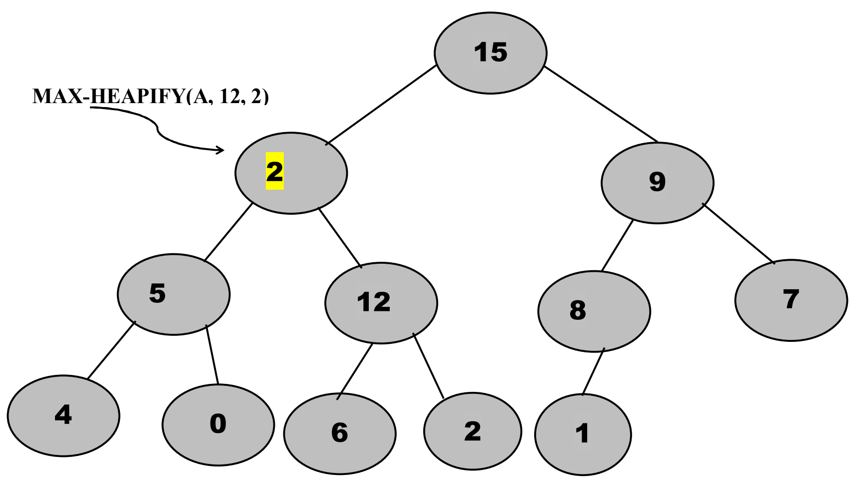
```
MAX-HEAP-CHANGE-KEY(A, n, i, key)
{
    if (key==A[i])
      return;
    else if (key>A[i])
   {
        A[i]←key;
        while (i>1 and A[PARENT(i)] < A[i])
         {
             A[i] ↔ A[PARENT(i)];
             i ← PARENT(i);
        }
   }
   else
   {
        A[i]←key;
        MAX-HEAPIFY(A, n, i);
   }
}
```

<u>Illustrate the operation of MAX-HEAP-CHANGE-KEY(A, 12, 2, 2)</u>

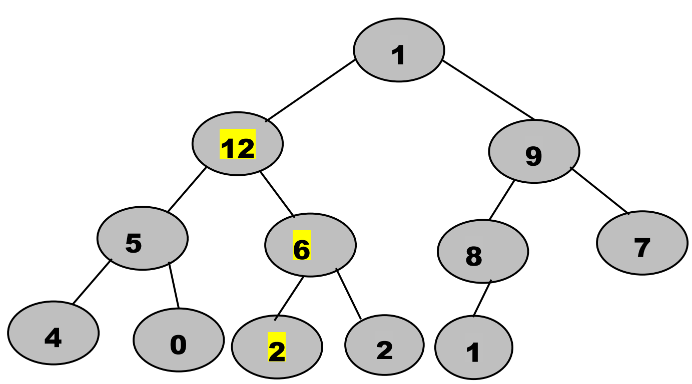
Given Heap



Now 2<13 => replace 13 by 2 and apply MAX-HEAPIFY(A, 12, 2)



After applying MAX-HEAPIFY(A, 12, 2), the max-heap becomes



(Final Max-Heap)