Maharishi University of Management

MIDTERM SOLN

Course Title and Code: *CS 435 - Design and Analysis of Algorithms*
Instructo: *Dr. Emdad Khan*
Date: *Friday 07/03/2015*
Duration: *10am - 12 pm*

Student Name:

Student ID:

Total Mark
_____
46

1. This is a closed book exam. Do not use any notes or books!
2. Show your work. Partial credit will be given. Grading will be based on correctness, clarity and neatness.
3. We suggest you to read the whole exam before beginning to work any problem.
4. There are 4 questions worth a total of 46 points, on 5 pages (including this one)

Question 1: 10 points (3 + 5 + 2)

a. Complete the following table. Explain your answer as appropriate.

Algorithm	Worst Case / Stable / In-Place
Insertion Sort	$\Theta(n^2)$ / Yes / Yes
Selection Sort	$\Theta(n^2)$ / Not in general / Yes
Merge Sort	O(nlgn) / Yes / No
Quick Sort	$O(n^2)$ but O(nlgn) much more likely / No / Yes if Partition is In-Place
Radix Sort	O(n) / Yes / No
Counting Sort	O(n) / Yes / No

b. Determine whether f is O, o, Big omega or small omega of g where

f = n ^ k and g = c ^ n; c > 1 and k >= 1.

Ans. Lt (n -> inf) f/g -> Lt (n -> inf) f' / g' (Using L'Hopital rule) -> k.n^ (k-1) /(lg(c) c ^ n). After successive derivatives, we have, k(k-1)(k-2).....1. (n ^ 0) / ([lg(c)]^k. c ^ n) which is 0 as the denominator becomes infinity.

Thus, f is o (g) where o means small o. It is not small omega (g).

c. Show that $6n^2$ is $\Theta(n^2)$.

Ans. f(n) is $\Omega(g(n))$ iff g(n) is O(f(n)) iff there is a constant c > 0 and an integer constant $n_0 \geq 1$ such that $g(n) \leq c \bullet f(n)$ for all $n \geq n_0$

So $6n^2$ is $\Omega(n^2)$ iff n^2 is $O(6n^2)$, which is obviously true.

To show $6n^2$ is $\Theta(n^2)$, must show also that $6n^2$ is also $O(n^2)$ – this is also obvious.

Therefore $6n^2$ is $\Theta(n^2)$.

Question 2: 11 points (3 + 3 + 5)

For each of the following recurrences, derive an expression for the running time using iterative, substitution or Master Theorem.

a. $T(n) = 2\,T\left(\frac{n}{2}\right) + n^3$

Ans. a=2, b=2, c =1, k =3, and a < b ^k [f(n)=n^3>0]. So we can apply the Master theorem. Hence, we have,

T(n) = Θ(n^3)

b. $\begin{cases} T(n) = T(n-1) + n \\ T(1) = 1 \end{cases}$

Ans. Using iterative method, we have, T(n) = T(n-2) + (n-1) + n
T(n) = T(n-3) + (n-2) + (n-1) + n
T(n) = T(n-4) + (n-3) + (n-2) + (n-1) + n
T(n) = T(n-5) + (n-4) + (n-3) + (n-2) + (n-1) + n

So now rewrite these five equations and look for a pattern:

T(n) = T(n - 1) + n
T(n) = T(n-2) + (n-1) + n
T(n) = T(n-3) + (n-2) + (n-1) + n
T(n) = T(n-4) + (n-3) + (n-2) + (n-1) + n
T(n) = T(n-5) + (n-4) + (n-3) + (n-2) + (n-1) + n

Generalized recurrence relation at the kth step of the recursion:

$T(n)=T(n-k)+(n-k+1)+(n-k+2)+\cdots+(n-1)+n$

We have T(n-k) and we want T(1). So, we let n – k = 1. Also, solve for k, k = n – 1. Now, plug this in all across the board:
$T(n)=T(1)+2+3+\cdots+(n-1)+n]$
$T(n)=n(n+1)/2=O(n2)$

c. Consider the following recurrence algorithm

```
procedure T(n,x)
  if(n==1)
    return true
  else
  {
  x = x + n
```

```
    Call T(n-1,x)
    }
end procedure
```

i. (2 point) Write a recurrence equation for T(n)

Solution:
i. T(n) = 1 if n = 1,

T(n) = 1 + T(n-1) if n > 1

ii. (3 points) Solve recurrence equation using iterative method i.e. give an expression for the runtime T(n).

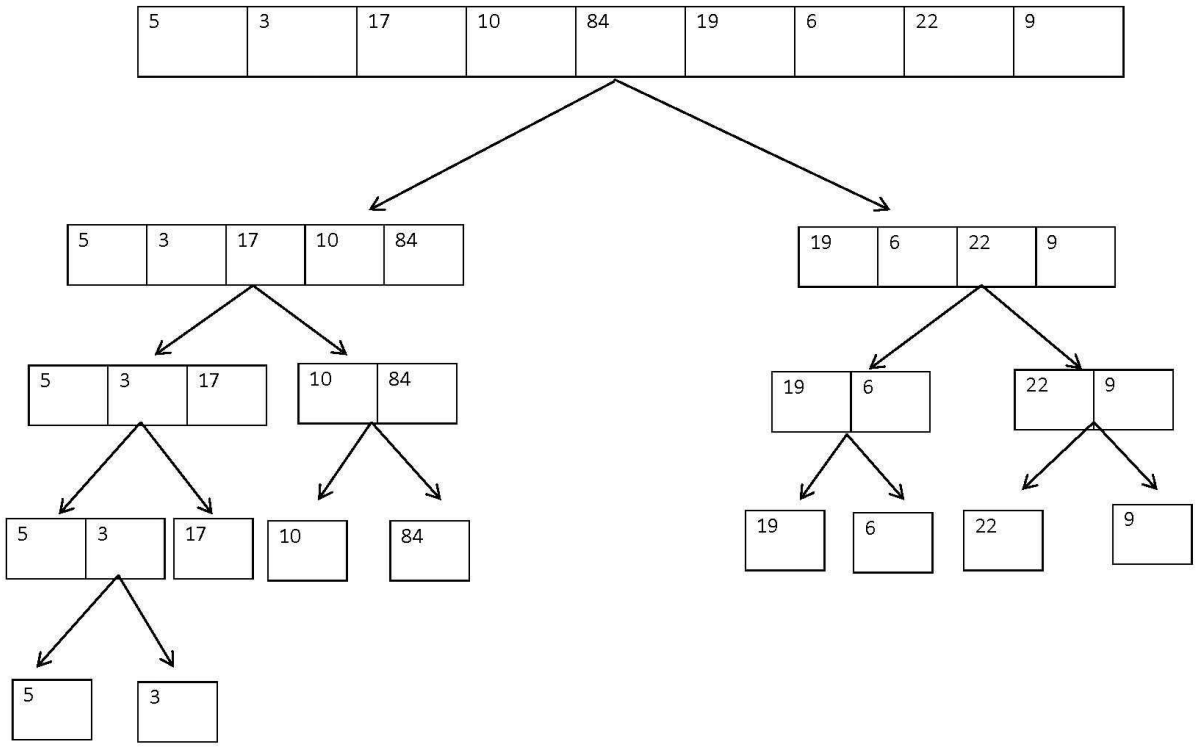
Solution:
T(n) = T(n □ 1) + 1
T(n) = ((T(n □ 2) + 1) + 1
T(n) = ((T(n □ 3) + 1) + 1) + 1
:
:
T(n) = T(1) + 1 + 1 + ::::: + 1
T(n) = 1 +
$\sum_{k=0}^{n} X \square 1$
1
T(n) = O(n).

Question 3: 12 points (2 + 5 + 5)

a. Illustrate (by drawing) the operations of the mergesort algorithm on the array A={5,3,17,10,84,19,6,22,9}.

Ans.

(Next page)



Then sort recursively all the elements

b. Use the QuickSelect algorithm to manually compute the 5th smallest element of the array [1, 5, 23, 0, 8, 4, 33]. Assume that the rightmost element is used as the pivot in each case. Show what happens in each self-call, indicating the new input array and the current value of *k*.

Ans. QS (1,5, 23, 0, 8, 4, 33), k = 5

L = [1,5,23,0,8,4], E = [33], G = [ ]

k = 5 <= |L|

QS (1,5,23,0,8,4), k = 5

L = [1, 0], E = [4], G = [5, 23, 8]

k' = k – |L| – |E| = 5 – 2 – 1 = 2

QS (5, 23, 8), k = 2

L = [5], E = [8], G = [23]

|L| < k ≤ |L| + |E|, so **return 8**

c. Use RadixSort, with two bucket arrays and radix = 6, to sort the following array:

[1, 8, 3, 2, 34, 21].

Show all steps of the sorting procedure. Then explain why the running time is O(*n*). What would be the running time if you used ONE bucket?

Ans.

r[] =

	1	2 8	21 3	34	
0	1	2	3	4	5

q[]=

3 2 1	8		21		34
0	1	2	3	4	5

The formulae used are

1. Bucket r - use input values x and remainder using mod 6.

2. Bucket q needs to be filled by reading values (left to right) from bucket r but divided by 6.

Return: [1, 2, 3, 8, 21, 34]

Running Time: O(n):

- Initializing each of the bucket arrays costs O(n)
- Populating r[] and q[] costs O(n)
- Reading output from q[] is O(n)

If one bucket is used, then we would need n^2 size as the range is close to 6^2 where 6 is the number of inputs. So, the time would be O(n^2).

Question 4: 13 points (3 + 6 + 4)

a. Is mathematics decidable? Explain the Halting problem in your own words (no need to prove).

Ans. No, mathematics is not decidable. This is can be verified by using the Halting Problem which states that given a program P, there is no way one can determine whether P will halt, terminate normally or will continue to run for ever. An algorithm to define the Halting problem exists but there is no algorithm to solve the Halting Problem.

b. Given an array A of n numbers, suggest an O(n) **expected time** algorithm to determine whether there is a number in A that appears more than $n / 2$ times.

Ans. **Use an array C of size k, where k is the range of the elements stored in A. Use a single loop, and increment the entries of C indexed by A[i] for i running from 0 to n. After each increment check whether the C[A[i]] >= n/2, if yes print A[i] and exit the program.**

```
for (i=0; i<n; i++)
{
        C[A[i]]++;
        if(C[A[i]] >= n/2)
        {
                cout << A[i] << "is such a number" << endl;
                exit(0);
        }
}
```

c. Use Decision tree and binary tree basic ideas to prove the following theorem:

"Every comparison based sorting algorithm has, for each n, running on input of size n, a worst case in which its running time is Ω(nlog n)".

Ans.

We know that on a decision tree there are n! leaves. We also know that a tree with height h has max 2^h leaves. So we have,

$n! \leq 2^h$

Taking logarithms, we get

$\lg(n!) \leq h$

Stirling's approximation tells us: $n! > \left(\frac{n}{e}\right)^n$

Thus, we have,

$$h \geq \lg\left(\frac{n}{e}\right)^n$$

$$= n\lg n - n\lg e$$

$$= \Omega(n\lg n)$$

Thus, the minimum height of a decision tree is $\Omega(n \lg n)$. So, the running time for comparison based algorithm is Ω(nlog n).