

# SAMPLE SOLUTION.

## Fall 2008, Test 1, Data Structures and Algorithms

Name:  
Section:  
Email id:

2nd October, 2008

Answer all Seven questions. You have 100 minutes to complete the exam.  
The formulas you may need are:

### 1 Formulas

$$\sum_{i=0}^{i=n} i = n(n+1)/2$$

$$\sum_{i=0}^{i=n} i^2 = n(n+1)(2n+1)/6$$

$$\sum_{i=0}^{i=n} a^i = (a^{n+1} - 1)/(a - 1), \text{ where } a \neq 1$$

## 1. Algorithm and Recursion

- (a) Write an efficient algorithm (C++ like program) to compute an integer value ( $x$ ) which is  $\log_2(\log_2(n))$  (where  $n$  is the given input positive integer).  $2^{2^{x-1}} < n \leq 2^{2^x}$  of a given positive integer. For example if  $n = 3$ , the output should be 1, if  $n = 16$ , the output should be 2 and if  $n = 690$ , the output should be 4. What is the running time of your algorithm in  $O$  notation. [5 points]

$O(\log \log n)$

```

int loglog(int n)
{
    int i = 0;
    if (n <= 2) return i;
    x = 2;
    while (n > x) {
        x = x * x;
        i = i + 1;
    }
    return i;
}
    
```

- (b) How many times, as a function of  $n$  does the following program print? Hint: Find the values for small values of  $n$  and write a recurrence relation.

```

void f(int n)
{
    if (n > 1)
    {
        cout << "Good Luck\t"
        cout << "Comes with Hard Work " << endl;
        f(n-1);
    }
}
    
```

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

$$T(2) = 2;$$

$$T(3) = 4$$

$$T(4) = 6$$

$$T(n) = 2n - 2$$

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

$$\therefore T(n) = T(n-2) + 4$$

$$\vdots$$

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

## 2. Big Oh Notations

- (a) Take the following list of function and arrange them in ascending order of growth. [5 points]

$$f_1(n) = n^2$$

$$f_2(n) = n^3$$

$$f_3(n) = 100n^2$$

$$f_4(n) = n \log(n)$$

$$f_5(n) = 2^n$$

$$f_6(n) = 2^{2^n}$$

$$f_4(n) < f_1(n) = f_3(n) < f_2(n) < f_5(n) < f_6(n)$$

- (b) If  $F(n) = \sqrt{2^{\log_2(n)}}$  and  $T(n) = \sqrt{\log_2(n)}$ . prove which one of the following is true.  $T(n) = O(F(n))$  or  $T(n) = \Omega(F(n))$  or  $T(n) = \Theta(F(n))$  [5 points]

$$F(n) = 2^{\frac{\log_2 n}{2}}$$
$$= \left(2^{\log_2 n}\right)^{\frac{1}{2}}$$

$$= n^{\frac{1}{2}}$$

$$T(n) = (\log_2 n)^{\frac{1}{2}}$$

$$\lim_{n \rightarrow \infty} \frac{T(n)}{F(n)} = 0$$

$$T(n) = O(F(n))$$

### 3. Recurrence Relation and Proof By Induction

(a) Solve the following recurrence equation

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

in terms of explicit functions of  $n$  and  $T(1)$ . Assume  $n = 2^k$ . [5 points]

$$\begin{aligned} T(2^k) &= T(2^{k-1}) + 2^k && \text{--- (1)} \\ T(2^{k-1}) &= T(2^{k-2}) + 2^{k-1} && \text{--- (2)} \\ T(2^k) &= T(2^{k-2}) + 2^k + 2^{k-1} && \text{Subst (2) in (1)} \\ &\vdots \\ T(2^k) &= T(2^{k-k}) + 2 + 2^2 + \dots + 2^k \\ &= T(1) + 2(1 + \dots + 2^{k-1}) \\ &= T(1) + 2 \frac{2^k - 1}{2 - 1} \\ &= T(1) + 2^{k+1} - 2 \end{aligned}$$

(b) Find the values of  $a$  and  $b$  and prove by induction hypothesis the answer you got.

$$\sum_{i=1}^{i=n} i2^i = 2^{n+1}(n-b) + a$$

[5 points]

$$\begin{aligned} S &= \sum_{i=1}^n i \cdot 2^i && \text{--- (1)} \\ &= 1 \cdot 2 + 2 \cdot 2^2 + \dots + n \cdot 2^n \\ 2S &= 1 \cdot 2^2 + 2 \cdot 2^3 + \dots + n \cdot 2^{n+1} && \text{--- (2)} \end{aligned}$$

Subtracting (1) from (2), we get

$$\begin{aligned} S &= n \cdot 2^{n+1} - 2 - 2^2 - \dots - 2^n \\ &= n \cdot 2^{n+1} - 2(1 + 2 + \dots + 2^{n-1}) \\ &= n \cdot 2^{n+1} - 2(2^n - 1) = n \cdot 2^{n+1} - 2^{n+1} + 2 \\ &= 2^{n+1}(n-1) + 2 \end{aligned}$$

$a = 2, b = 1$

Base:

$$1 \cdot 2 = 2^2(1-1) + 2 = 2$$

Assume Ind. Hypo. upto  $n$

$$S = 2^{n+1}(n-1) + 2$$

for  $i = n+1$

$$\begin{aligned} S &= 2^{n+1}(n-1) + 2 + (n+1)2^{n+1} \\ &= 2^{n+1}(n-1+n+1) + 2 = 2^{n+1}(n) + 2 \end{aligned}$$

$= 2^{n+1}(n+1-1) + 2$  Hence proved.

#### 4. Design of Algorithm

You are doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still do not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with  $n$  rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call the highest safe rung.

It might be natural to try binary search: drop a jar from the middle rung, see if it breaks, and then recursively try from  $n/4$  or  $3n/4$  depending on the outcome. But this has the drawback that you break a lot of jars finding the answer.

If your primary goal were to conserve jars, on the other hand, you could try the following strategy. Start by dropping a jar from the first rung, and then the second rung, and so forth, climbing one higher, until the jar breaks. In this way, you only need a single jar - at the moment it breaks you have the correct answer - but you may have to drop it  $n$  times (rather than the  $\log(n)$  as in the binary search solution.)

So here is a trade-off: It seems that you can perform fewer drops if you are willing to break more jars. To understand how this trade-off works, let us consider this given a fixed "budget" of  $k$  jars. In other words, you have to determine the correct answer - the highest safe rung - and can break at most  $k$  jars.

Your task is to devise a strategy if you are given  $k = 2$  jars. The number of drops should be asymptotically smaller than  $n$ . [10 Points]

Soln: 1

1. Drop at ~~every~~ <sup>steps of</sup>  $\sqrt{n}$  rung. till it breaks.
2. Suppose it breaks at  $i\sqrt{n}$  rung
3. ~~Search~~ <sup>Drop</sup> sequentially from  $(i-1)\sqrt{n}+1$  till  $i\sqrt{n}$  to find the safe rung. complexity  $O(\sqrt{n})$

Soln 2:

1. Drop at ~~every~~ <sup>steps of</sup>  $\frac{n}{\log n}$  rung till it breaks
2. Suppose it breaks at  $i \left(\frac{n}{\log n}\right)$  rung.
3. Drop sequentially from  $(i-1) \cdot \frac{n}{\log n}$  till  $i \cdot \frac{n}{\log n}$  to find the safe rung. complexity  $O\left(\frac{n}{\log n}\right)$

Soln 3:

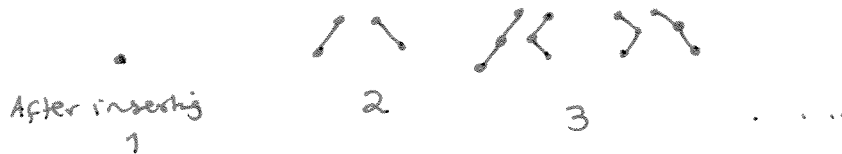
1. Drop at steps of  $n^{\frac{1}{k}}$  rung till it breaks.
2. Suppose it breaks at  $i \cdot n^{\frac{1}{k}}$  rung
3. Drop sequentially from  $(i-1)n^{\frac{1}{k}}+1$  till  $i \cdot n^{\frac{1}{k}}$  to find the safe rung. complexity  $O\left(\min\left(n^{\frac{1}{k}}, n^{\frac{k-1}{k}}\right)\right)$

## 5. Binary Search Tree

- (a) If the number of comparisons is monotonically increasing while inserting keys in a binary search tree. What will be the number of possible shapes of Binary Search Tree, after inserting  $n$  elements. [5 points]

Comparisons ~~is~~ ~~have~~ have to be  $0, 1, 2, 3, \dots, n-1$   
to make it monotonic and the comparisons have to go up by 1.

Number of possibilities =  $2^{n-1}$

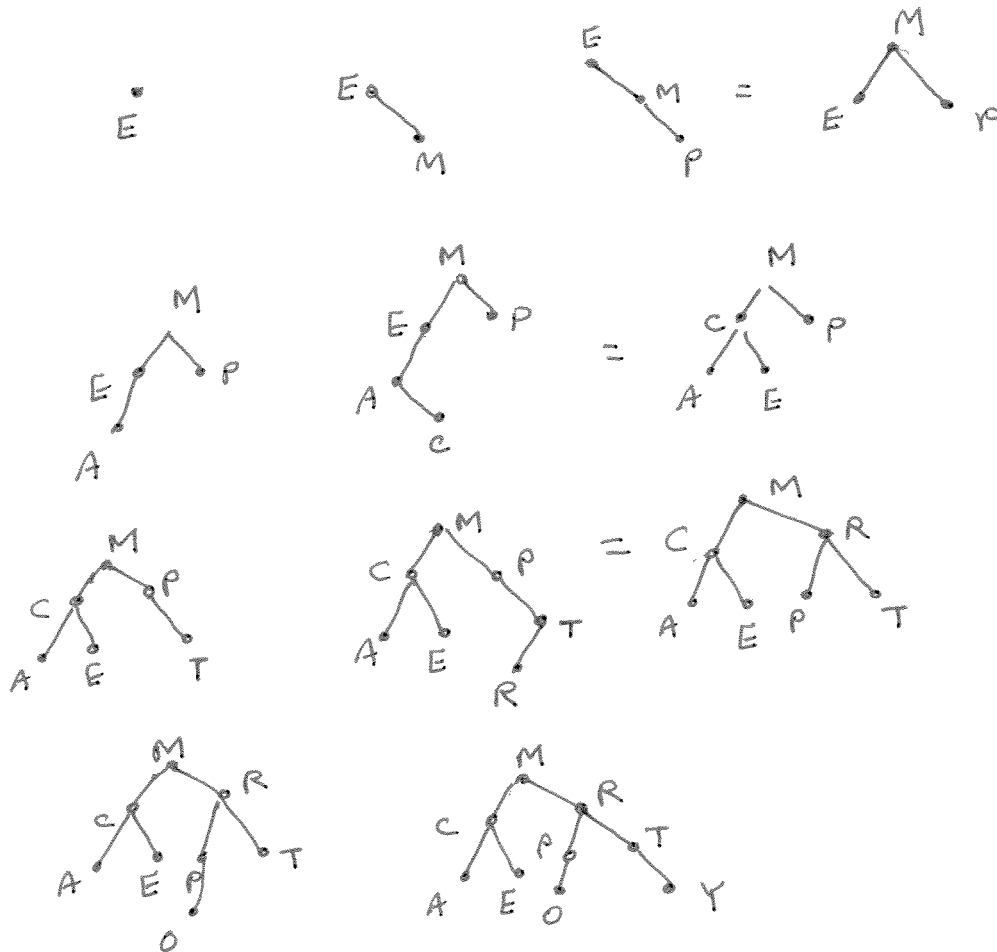


- (b) Write a C++ program to compute the length of the shortest path among all paths from the root node to leaf nodes in a full Binary Tree (that is all internal nodes have degree 2). [5 points]

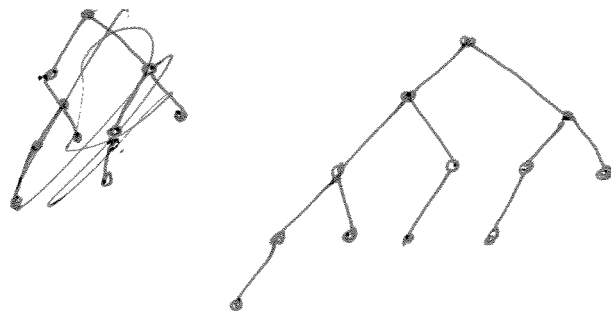
```
int Shortest_Path (Binary Tree * T)
{
    if (T == NULL) return 0;
    else
    {
        int a, b;
        a = Shortest_Path (T->left);
        b = Shortest_Path (T->right);
        if (a < b) return 1+a;
        else return 1+b;
    }
}
```

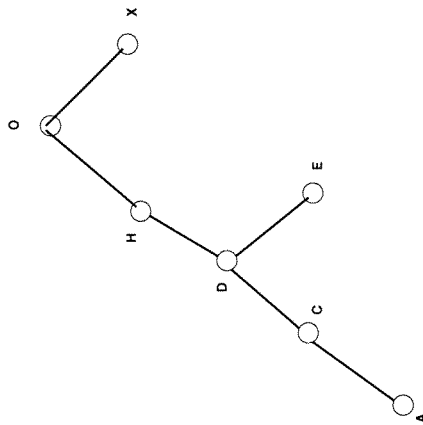
## 6. AVL Trees

- (a) Show the AVL tree when the following characters (E, M, P, A, C, T, R, O, Y) are inserted one at a time. Draw the AVL tree at the end of each insertion. [6 points]



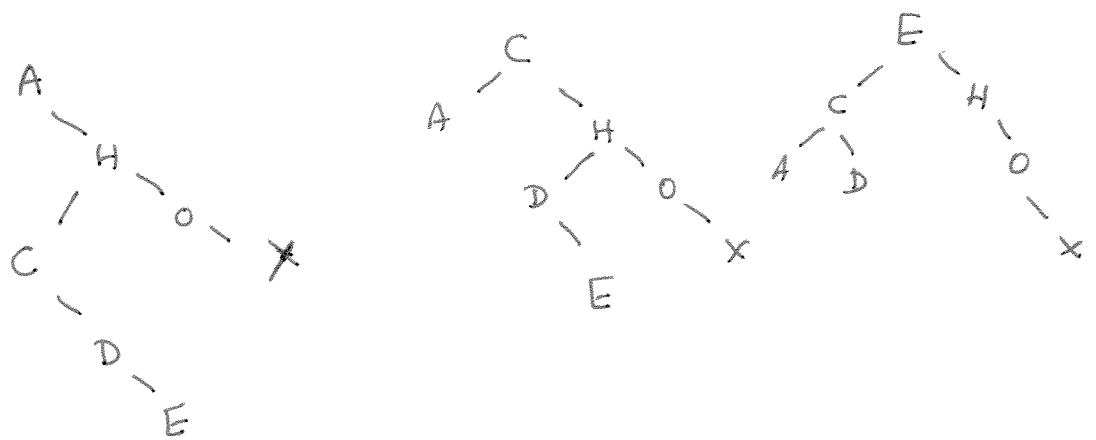
- (b) Draw an AVL tree with 12 nodes and height 4. [4 Points]





7. Splay Trees

(a) Display the splay tree after successively accessing A, C and E. [5 Points]



- (b) Instead of inserting an element as a leaf node (as in Binary Search Tree or AVL tree), we insert it as a new root node (maintaining a Binary Search Property). Draw a tree after inserting the keys 1, 6, 8, 0, 3, 9 and 7. [5 Points]

