Birla Institute of Technology and Science Pilani
Dept. of Computer Science & Information Systems
Design and Analysis of Algorithms (CS 364)

9–10.30AM
Friday Mar 10, 2017

Time: 1½hour *Mid-Semester Test (Closed Book)* Maximum Marks: 60

INSTRUCTIONS:

1. There are **6** questions spread over **2** pages, make sure you have all of them. Attempt **ALL** questions
2. State clearly, any assumption that you make while solving the problem

1. Suppose you are given two sets A and B, each containing n positive integers. You can choose to reorder each set however you like. After reordering, let a_i be the ith element of set A, and let b_i be the ith element of set B. You then receive a payoff of $\sum_{i=1}^{n} a_i^{b_i}$. Design an algorithm that will maximize your payoff. Give a formal correctness proof for your algorithm and find its time complexity. [10]

2. Consider the `randomized-quick sort` algorithm discussed in the class. You have seen that the worst case behavior of this algorithm is no better then the `merge sort` algorithm which takes $O(n \log n)$ time in the same setting. The analysis that leads to this conclusion assumes that `partition` in every step leaves a constant number of items (typically single) in one of the sides. Explain how this analysis would change when the assumption is *balanced partitioning*. In other words analyze the worst case behavior of `randomized-quick sort` algorithm assuming balanced partitioning in every step. [10]

   By balanced partitioning, one means that there would be at least a constant fraction of items on each side (the constant being positive real number, whatever small).

3. Consider `Median of Medians` algorithm discussed in the class that was useful for order statistics operations. Following algorithm shows how Median of Medians can be used to determine an item with rank k [10]

Algorithm 1: mom-Rank-I(A, k)

```
1 j=Median-of-Medians(A)
2 if $j = k$ then
3     return A[j]
4 if $k < j$ then
5     return mom-Rank-I( $A_<$, k)
6 else
7     return mom-Rank-I( $A_>$, k-j)
```

Clearly show all the steps in execution of this algorithm while determining a rank 12 item from a list of items given below:

25, 56, 77, 3, 21, 71, 29, 27, 93, 6, 47, 87, 9, 39, 2, 1, 32, 5, 67, 82, 91, 72, 7, 95, 53, 34, 23, 24, 43, 68, 7, 8, 65, 44, 4.

4. Consider `open addressing` where all elements occupy their position in the hash table itself. Therefore, table entry either contains an element or NIL value. [10]

   (a) Define probe (not more than 3 lines) sequence and algorithm for insertion and searching in a hash table (write algorithm only)

   (b) What improvement Quadratic probing and Double hashing brings to Linear probing? (in 50 words)

   (c) Insert the following keys (in order of their appearance) to an open address table:

   72, 83, 30, 8, 19, 3, 9, 31, 50, 15, 8.

   Let the number of slots be 20, and double hashing is implemented with the following two hash functions

$$h_1(x) = 2x + 3 \pmod{7}$$

$$h_2(x) = 3x + 5 \pmod{9}$$

Show the open address table finally obtained.

5. Let $G = (V, E)$ be an undirected graph with n nodes. Recall that a subset of the nodes is called an *independent set* if no two of them are joined by an edge. Finding large independent sets is difficult in general; but here we will see that it can be done efficiently if the graph is "simple" enough. Call a graph $G = (V, E)$ a *path* if its nodes can be written as $v_1, v_2, ..., v_n$, with an edge between v_i and v_j if and only if the numbers i and j differ by exactly 1. With each node v_i, we associate a positive integer weight w_i. Consider, for example, the five-node path drawn in Figure 1. The weights are the numbers drawn inside the nodes. The goal in this question is to solve the following problem: [10]
Find an independent set in a path G whose total weight is as large as possible.
Design an algorithm that takes an n-node path G with weights and returns an independent set of maximum total weight. The running time of your algorithm should be polynomial in n, independent of the values of the weights. Give a formal correctness proof for your algorithm. Find the time complexity of your algorithm and show its working on the given example (Figure 1).

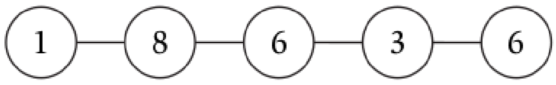

Figure 1: A path with weights on the nodes. The maximum weight of an independent set is 14.

6. Recall the `B-Tree` algorithm discussed in the class. We have discussed that B-Tree is a generalization of `binary search tree` that is balanced and has height $h \leq \log_t \frac{n+1}{2}$ when n number of keys stored. One of the important properties of any B-Tree is its **minimum degree** $t \geq 2$, that specifies that non-root nodes must have $t - 1$ to $2t - 1$ number of keys. Any node x has $x.n$ keys and $x.n + 1$ pointers. Having the value of minimum degree to be 3, construct a B-Tree by incremental insertion of the following keys: [10]

26, 83, 92, 73, 8, 96, 54, 35, 24, 25, 88, 10, 57, 78, 40, 3, 2, 33, 6, 68, 44, 69, 8, 9, 66, 45, 5, 4, 22, 72, 30, 28, 94, 7, 48.

Show all the intermediate steps where number of nodes change.