Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn

Exam Algorithm Theory

Wednesday, February 27 2013, 16:00 – 17:30

Name:	....
Matriculation Nr.:	....
Signature.:	....

Do not open or turn until told so by the supervisor!

Instructions:

- Write your name and matriculation number on the cover page of the exam and sign the document! Write your name on all sheets!
- Your signature confirms that you have answered all exam questions without any help, and that you have notified exam supervision of any interference.
- Write legibly and only use a pen (ink or ball point). Do **not** use **red**! Do **not** use a pencil!
- You are **not** allowed to use any material except for a dictionary, a pocket calculator, and a self-written summary of at most 5 A4 pages (corresponds to 5 single-sided A4 sheets!).
- There are 3 problems (with several questions per problem) and there is a total of 90 points. The number of points for each question is given.
- Use a separate sheet of paper for each of the 3 problems.
- Only one solution per question is graded! Make sure to strike out any solutions that you do not want to be considered!

- **Do not leave your seat at the end of the examination until all exams have been collected!**

Question	Achieved Points	Max Points
1		37
2		27
3		26
Total		90

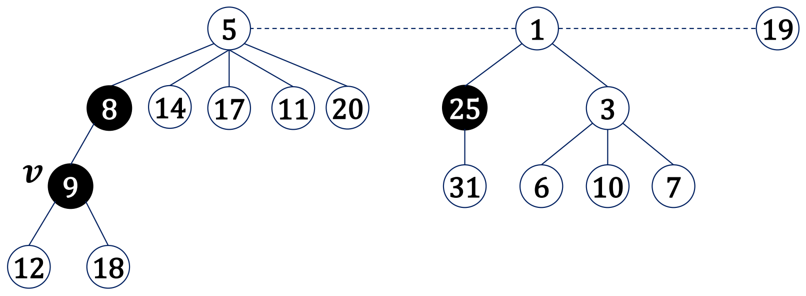
Problem 1: Short Questions **37 points**

(a) **(7 points)** Consider an unsorted array containing n numbers. A majority element is a number x that occurs more than $n/2$ times in the array. Consider the following recursive divide-and-conquer algorithm to find the majority element of an array (if there is a majority element).

1. Divide the array into a left and a right half and solve the problem recursively for the two halfs. If the left half has a majority element, let x be this element and let y be the majority element of the right half if the right half contains a such an element.
2. If the left half has a majority element x, count the number of occurrences of x in the array. If x occurs more than $n/2$ times, return x. Otherwise if the right half has a majority element y, count the number of occurrences of y in the array. If y occurs more than $n/2$ times, return y, otherwise return that no majority element exists.

Argue why the above algorithm is correct and analyze its asymptotic running time.

(b) **(8 points)** Consider the following Fibonacci heap (black nodes are marked, white nodes are unmarked). How does the given Fibonacci heap look after a *decrease-key*$(v, 2)$ operation and how does it look after a subsequent *delete-min* operation?



(c) **(7 points)** The maximum 3-coloring problem asks for assigning one of the colors $\{1, 2, 3\}$ to each node $v \in V$ of a graph $G = (V, E)$ such that the number of edges $\{u, v\} \in E$ for which u and v get different colors is maximized. A simple randomized algorithm for the problem would be to (independently) assign a uniform random color to each node. What is the expected approximation ratio of this algorithm?

(d) **(7 points)** In the lecture, we discussed the random contraction algorithm to obtain a minimum edge cut. One could try to use the same algorithm to also find a maximum edge cut (partition $A \subset V, B = V \setminus A$ of the nodes so that the number of edges connecting nodes in A and B is maximized). Show that for some graphs, the probability that the contraction algorithm returns a maximum cut is 0.

(e) **(8 points)** Give a PRAM algorithm that uses the all-prefix-sums operation to find the position of the first positive entry in an array of n integers. What is the running time of your algorithm if you use p processors for the computation?

Problem 2: Knapsack **27 points**

In the lecture, we have considered the knapsack problem: There are n items with positive weights $w_1, \dots, w_n$ and values $v_1, \dots, v_n$ and a knapsack (a bag) of capacity W. A feasible solution to the problem is a subset of the items such that their total weight does not exceed W. The objective is to find a feasible solution of maximum possible total value.

a) **(7 points)** For the case where all weights are positive integers, we have discussed a dynamic programming solution that solves the knapsack problem in time $O(nW)$. Briefly describe this algorithm.

b) **(10 points)** Assume that instead of the weights, the *values of all items are positive integers*. The weights of the items can be arbitrary positive real numbers. Describe a dynamic programming algorithm that solves the knapsack problem if all values are positive integers and give the running time of your algorithm.

   Hint: Consider only the first $k \in \{0, \dots, n\}$ items and study solutions that achieve a total value of exactly $\ell \in \{0, \dots, X\}$, where X is the sum of the values of all the items.

c) **(10 points)** Assume that you are given a knapsack instance where both the weights and the values are arbitrary positive real numbers. In order to be able to use the algorithm developed in b) to get an approximate solution, you decide to round all values to the next larger integer (i.e., each element $i \in \{1, \dots, n\}$ gets a new value $v_i' = \lceil v_i \rceil$). Assuming that all original values $v_i \geq 1$, show that the optimal solution for the rounded values v_i' is a 2-approximation for the original problem.

Problem 3: Assigning Cooking Days **26 points**

During their studies, 7 friends (Alice, Bob, Carmen, Dave, Eve, Fritz, and George) live to together in a house. They agree that each of them has to cook dinner on exactly one day of the week. However assigning the days turns out a bit tricky because each of the 7 students is unavailable on some of the days. Specifically, they are **unavailable** on the following days (1=Monday, 2=Tuesday, ..., 7=Sunday):

- Alice: 2, 3, 4, 5
- Bob: 1, 2, 4, 7
- Carmen: 3, 4, 6, 7
- Dave: 1, 2, 3, 5, 6
- Eve: 1, 3, 4, 5, 7
- Fritz: 1, 2, 3, 5, 6
- George: 1, 2, 5, 6

a) **(9 points)** Transform the above problem into a maximum flow problem and draw the resulting flow network.

b) **(11 points)** The 7 students start experimenting with an assignment of the evenings and they arrive at the following partial solution:

Alice: 7, Bob: 3, Carmen: 1, Eve: 6, George: 4

Unfortunately, the solution cannot be extended to a complete solution where each day of the week is assigned to exactly one of the students. Apply the Ford Fulkerson algorithm (augmenting paths) to transform the given partial solution into a complete solution (give the residual capacities and the augmenting paths you use for all steps of the algorithm).

c) **(6 points)** Now assume that instead of 7 students and 7 days, there are n students and n days. Assume that you have a partial solution which assigns a day to all except 2 students so that no day is assigned to more than one student. What is the asymptotic (worst-case) time needed to transform the given solution into a complete solution or to find out that no such solution exists? Explain your answer!