Final Exam, Algorithms 2013-2014

- You are only allowed to have a handwritten A4 page written on both sides.
- Communication, calculators, cell phones, computers, etc... are not allowed.
- Your explanations should be clear enough and in sufficient detail so that a fellow student can understand it. For example, a description of an algorithm should be so that a fellow student can easily implement the algorithm following the description. In particular, do not only write pseudocode without additional explanation.
- **Do not touch until the start of the exam.**

Good luck!

Name: ______________________ **N° Sciper:** ______________

Problem 1 / 15 points	Problem 2 / 20 points	Problem 3 / 15 points	Problem 4 / 20 points	Problem 5 / 10 points	Problem 6 / 20 points

Total / 100

1 *(15 pts)* **Asymptotics and Recursions.**

Suppose you are choosing between the following five Divide-and-Conquer algorithms:

Algorithm A solves problems of size n by dividing (in constant time) them into two subproblems each of size $n/2$, recursively solving each subproblem, and then combining the solutions in $\Theta(n^3)$ time.

Algorithm B solves problems of size n by dividing (in constant time) them into nine subproblems each of size $n/3$, recursively solving each subproblem, and then combining the solutions in $\Theta(n^2)$ time.

Algorithm C solves problems of size n by dividing (in constant time) them into ten subproblems each of size $n/3$, recursively solving each subproblem, and then combining the solutions in $\Theta(n)$ time.

Algorithm D solves problems of size n by dividing (in constant time) them into eight subproblems each of size $n/2$, recursively solving each subproblem, and then combining the solutions in constant time.

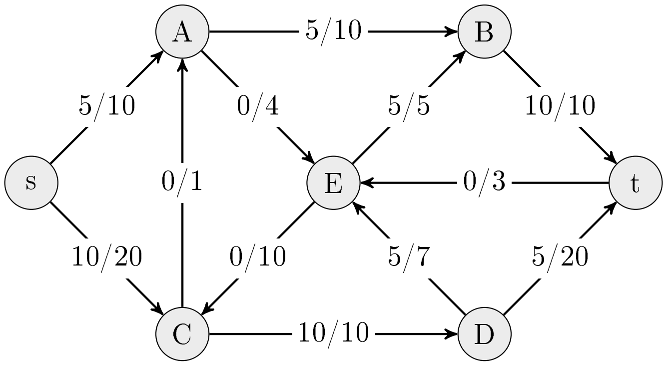
Algorithm E solves problems of size n by dividing (in constant time) them into two subproblems each of size $n - 1$, recursively solving each subproblem, and then combining the solutions in constant time.

What are the running times of each of these algorithms (in Θ notation), and which would you choose?

Solution:

(This page is intentionally left blank.)

2 *(20 pts)* **Flows and Cuts.** Assume the following flow network and corresponding flows (the numbers on an edge determine its current flow and its capacity).



2a *(5 pts)* What is the net-flow across the cut $(\{s, A, C, E\}, \{t, B, D\})$? What is the capacity of the same cut?

Solution:

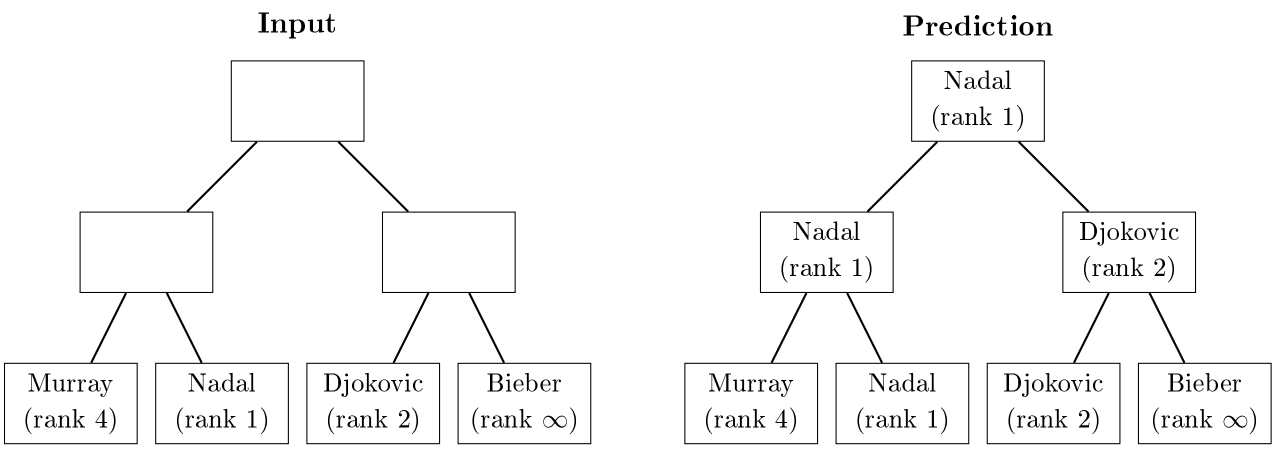
2b *(15 pts)* Starting with the depicted flow, *find a max-flow and a min-cut* by running the Ford-Fulkerson algorithm that in each iteration chooses the fattest augmenting path (the one that can carry the maximum amount of flow). In each iteration draw the residual network and *explain how you found the min-cut.* Finally, *write down the value of the found max-flow and the capacity of the min-cut.*

Solution:

(This page is intentionally left blank.)

3 *(15 pts)* **Australian Open.** The draw of Australian Open was recently announced. In tennis each match is between two players and the winner progresses to the next round. This naturally leads to a complete binary tree structure of the tournament. At the leaves, we have all the players in the tournament. At the next level, we have those that won their first match, and so on. In particularly, the root of the tree contains the winner of the tournament. We are interested in predicting the outcome of *every match* in Australian Open 2014. To do this we use the following simplifying assumption: a better ranked player always wins over a player with worse rank.

Consider the figure below for an example. We have four players entering the tournament of various rankings. The prediction tree then predicts the winner of each match. For example, as Rafael Nadal is currently ranked number one and Andy Murray is ranked number four, Rafael Nadal wins against Andy Murray. Similarly, we predict that Nadal wins against Djokovic in the final.



Design and analyze an efficient algorithm for the Australian Open prediction problem:

Input: The root of a complete binary tree (the draw) with n players as leaves. Each player/node has a *name* and a *ranking* that is initially empty for nodes that are not leaves. In addition, each node has pointers to its *left child*, its *right child* and *its parent*. Finally, no two players have the same ranking.

Output: A complete binary tree (the prediction) where each node contains the player (his name and rank) that has reached this stage assuming that better ranked players always win over worse ranked players.

Your algorithm should run in *linear time* in the number of players.

Solution:

(This page is intentionally left blank.)

4 *(20 pts)* **Probabilistic analysis**. We shall analyze a randomized procedure RANDOMIZED-SELECT for the select problem: given an array A consisting of n unique integers and an integer $k \leq n$, output the kth smallest integer of A.

For example, if the input is $A =$ | 89 | 14 | 16 | 28 | 51 | 25 | and $k = 3$ then the correct output is 25.

To simplify the description of RANDOMIZED-SELECT, we let $|A|$ denote the length of the array. The pseudocode is as follows:

RANDOMIZED-SELECT(A, k)

1. Pick *pivot* uniformly at random from the numbers in A.
2. Compare each number in A with *pivot* to obtain arrays S and L:
   S contains all numbers of A strictly smaller than pivot.
   L contains all numbers of A strictly larger than pivot.

3\. **if** $|S| = k - 1$
4\.   **return** *pivot*
5\. **else if** $|S| \geq k$
5\.   **return** RANDOMIZED-SELECT(S, k)
6\. **else** (we have $|S| < k - 1$)
7\.   **return** RANDOMIZED-SELECT$(L, k - (|S| + 1))$

The idea of the algorithm is very similar to RANDOMIZED-QUICKSORT that we saw in class. As in that algorithm, we first select a number *pivot* uniformly at random from the numbers in A. We then partition A into two arrays S and L that contain all numbers of A that are strictly smaller and strictly larger than *pivot*, respectively[1]. The time it takes to execute these steps (Steps 1 and 2) of the algorithm is $\Theta(|A|)$ which is also proportional to the number of "$\leq$"-comparisons we make to find S and L. After that, the algorithm recurses on the array where the kth smallest element can be found or simply returns the kth smallest element if the pivot equals it.

We shall now analyze the running time of RANDOMIZED-SELECT.

4a *(4 pts)* Suppose that we are extremely lucky: every time we select a pivot at random, the pivot that *minimizes* the running time is selected. What is the asymptotic running time of RANDOMIZED-SELECT in this lucky case? Motivate your answer.

Solution:

[1] As we saw in class, we can do this without using the extra space S and L. We have presented the algorithm in this way to make the description clearer.

4b *(6 pts)* Suppose that we are extremely unlucky: every time we select a pivot at random, the pivot that *maximizes* the running time is selected. What is the asymptotic running time of RANDOMIZED-SELECT in this unlucky case? Motivate your answer.

Solution:

4c We shall now analyze the *expected* running time of RANDOMIZED-SELECT on an array of length n. Similarly to RANDOMIZED-QUICKSORT, the running time is proportional to the total number of comparisons. As we saw in class, if we let X be the random variable that equals the total number of comparisons then

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{E}[X_{ij}],$$

where X_{ij} is the random indicator variable that takes value 1 if the ith smallest number was compared to the jth smallest number of the array, and 0 otherwise.

Give a tight asymptotic analysis of the *expected* running time of RANDOMIZED-SELECT by analyzing the above expression.

(Hint: Distinguish between the following three cases: $i < j \leq k$, $k \leq i < j$, and $i < k < j$.)

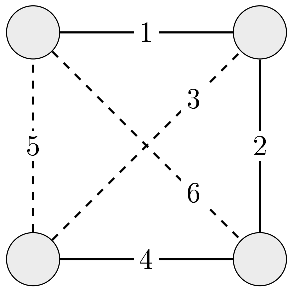
Solution:

(This page is intentionally left blank.)

5 *(10 pts)* **Spanning trees.** Consider three undirected edge-weighted connected graphs $G_1 = (V, E_1), G_2 = (V, E_2)$, and $H = (V, E_1 \cup E_2)$ with non-negative weights $w : E_1 \cup E_2 \to \mathbb{R}_+$ on the edges. Note that they are all graphs on the same vertex set but their edges differ: G_1 has only the edges in E_1, G_2 has only the edges in E_2, and H has all the edges $(E_1 \cup E_2)$.

Let T, T_1, T_2 be minimum spanning trees of H, G_1, and G_2, respectively. Assuming that the weights of the edges are unique, i.e., no two edges have the same weight, *prove that* $T \subseteq T_1 \cup T_2$.

For an example of the statement see the figure below. The solid edges are E_1 and the dashed edges are E_2. Note that the minimum spanning tree of G_1 is $T_1 = \{1, 2, 4\}$, the minimum spanning tree of G_2 is $T_2 = \{3, 5, 6\}$, and the minimum spanning tree of H is $T = \{1, 2, 3\}$. We have thus that $T \subseteq T_1 \cup T_2$ in this case. You should prove that it holds in general.



Solution:

6 *(20 pts)* **Dynamic programming.** The capacitated increasing subsequence problem is defined as follows:

Input: a sequence of integers $a_1, a_2, \ldots, a_n$, sizes $s_1, s_2, \ldots, s_n$ that are either $1, 2$, or 3, and an integral capacity C.

Output: the length of a longest subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ satisfying:

- It is an increasing subsequence: $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ and $a_{i_1} < a_{i_2} < \cdots < a_{i_k}$.
- It satisfies the capacity: $s_{i_1} + s_{i_2} + \cdots + s_{i_k} \leq C$.

For example, consider the following input: $a_1 = 8, a_2 = 7, a_3 = 9$ and $s_1 = 1, s_2 = 2, s_3 = 1$ and capacity $C = 2$. The correct output is 2 as a capacitated increasing subsequence of maximum length is a_1, a_3 which has size $s_1 + s_3 = 2 = C$.

6a *(10 pts)* Let $R(j, c) =$ "length of longest increasing subsequence which ends in a_j and has capacity at most c". If $s_j > c$, we let $R(j, c) = -\infty$. Find a recursive formulation of $R(j, c)$.

Solution:

6b *(10 pts) Design and analyze* an efficient algorithm for solving the capacitated increasing subsequence problem.

Solution: