Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
36^a Escuela de Ciencias Informáticas

Course: Advanced Data Structures **Date:** July 24–28, 2023
Take-home final exam

Instructions : Write your answers using your favorite editor or markup language (for example, LaTeX) and send me the PDF with your answers by email to `conrado@cs.upc.edu` not later than July 31st, 2023 (23:59 AoE). It is fine to include diagrams or figures in your solutions, but please do not send scans of handwritten solutions. Make sure that your file starts with your full name (first and last name(s)). Every question has 5 points, choose two, produce solutions and send me them as indicated above. Don't forget to put the numer of the questions you are solving. Maximum grade is 10 points, and all grades will be integer numbers in the range [1..10]. It is fine if you want to send more than two solutions: in that case, please indicate clearly which are the two questions that you have selected to answer for the exam.

1. **(5 points)** We have been given a box with n bolts and n nuts of different sizes. Each nut matches exactly one bolt and vice versa. All bolts and nuts are almost of the same size, so we cannot tell if one bolt is bigger than another or if one nut is bigger than another, and we don't have any instrument to measure them precisely. However, we can try to match one nut with one bolt, and the outcome is always clear: the nut is either too big, too small or exactly right for the bolt.

   The obvious algorithm —for each nut test every bolt until we find its right match— is too costly: we would need $\Theta(n^2)$ tests in the worst case to match every nut with its corresponding bolt.

   Devise a randomized algorithm that efficiently solves this problem on average. The algorithm must always solve the problem correctly, that is, it never leaves unmatched bolts or nuts. Explain your algorithm with enough detail to justify its correctness and to analyze is expected cost. Show that the expected number of tests for your algorithm satisfies a recurrence of the form

$$T_n = c \cdot n + \sum_{k=1}^{n} \pi_{n,k} \cdot (T_{k-1} + T_{n-k}),$$

   for some constant c and probabilities $\pi_{n,k}$ to be determined. Solve the recurrence using the continuous master theorem.

Useful formula:

$$\int z\,\ln(z)\,dz = \frac{z^2\ln(z)}{2} - \frac{z^2}{4} + C$$

---

2. **(5 points)** Consider the following algorithm (in pseudo-code):

```
// B: Bloom filter; Z = z1, z2, ..., zN: sequence of N elements
B:= empty filter; count:= 0
while (there are elements in Z) do
   z:= next item in Z
   if (z not in B)
       add z to B; count:= count + 1
   endif
endwhile
```

What can we say about `count`? What is its relation to the number n of **distinct** elements in Z?

Assume that the parameters M (size of the bitvector) and k (number of hash functions) of the Bloom filter have been choosen to guarantee that the rate of false positives is ≤ 0.01. Give bounds relating n and `count`.

---

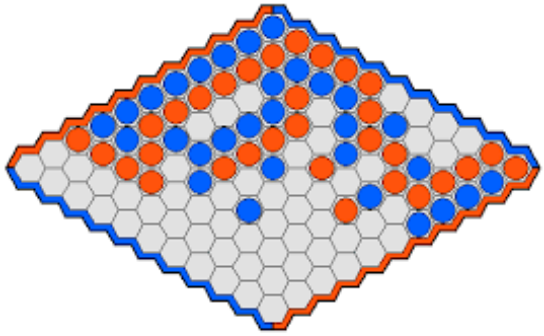
3. **(5 points)** We have implemented a dictionary class using a skip list; but we will need in our application to support two new operations:

   (a) Given a *rank* i, $1 \leq i \leq n = \text{size}(S)$, find the element in the skip list S with the i-th smallest key.

   (b) Report how many elements in the skip list S have a key in the range $[k_1..k_2]$, for two given keys k_1 and k_2. The keys k_1 and k_2 need not be present in S.

   For the first operation we should avoid traversing i elements in the last level: it's a correct but expensive solution. Likewise, for the second operation, our solution must avoid the brute force approach of searching k_1 and then traversing the last level of S from there until we reach an element with key $k > k_2$. Both operations can be done in expected time $\Theta(\log n)$ using extra information at each node/level of the skip list: namely, for a node x at level ℓ we keep together with the pointer to the successor of x at level ℓ the number of elements that can be accessed from that point. Explain how to implement efficient algorithms for the two operations above, and how the insertion and deletion algorithms must be modified so that the extra information can be updated and kept correct at all times.

4. **(5 points)** In the HEX game, invented by John Nash, two players play in an $n \times n$ board of hexagonal cells (i.e., each cell, except those in the boundary, has 6 neighboring cells). On each turn, one player puts a peeble of her color in any empty cell of the board. One player ("Red") has the red peebles and the other ("Blue") has the blue peebles. Each player owns two edges of the board, on opposite sides, and her goal is to connect her two edges of the board by creating a path from one edge to the other. Two cells are connected if they share a common edge and they contain peebles of the same color.



Red won

Design an efficient algorithm which allows us to detect if there is a winner of the Hex game at the end of each turn. Analyze the worst-case complexity of the algorithm as a function of the size of the board n and the number of turns N played so far (notice that $N \leq n^2$). In other words, explain how we can efficiently answer if there is a winner in a given board B. The cost of the algorithm below, in particular the cost of the N calls to `winner` (think in terms of amortized cost) and the cost of the omitted parts (the ... inside the lopp, for example).

```
B:= emptyBoard(n); // <== Cost: Theta(n ^2)
player:= Red;
...
while (not winner(B))
   askPos(B, player, i, j)
   B[i,j]:= player;
   player:= opponent(player);
   ...
endwhile
```

5. **(5 points)** Suppose we have a priority queue P where the elements are integers in the range $[0..n-1]$, each elements having an associated positive real-value priority. We also have a data structure of "handles": each handle is a pointer to the corresponding node (binomial queues, Fibonacci heaps) or an index to the array storing P (binary heaps). Give a high-level description of the operation to remove an element (not necessarily the minimum) given a handle to the element, for each of the three implementations that we have studied. Analyze the cost of the algorithm for each case.

---

6. **(5 points)** Give a high-level description (but detailed enough) of the algorithm to find all elements in a standard K-d tree that are at (Euclidean) distance of q not larger than a given radius R. In your algorithm, take into account that there is no need to store discriminants in the nodes of the standard K-d tree; they cycle along any path: $0, 1, 2, \ldots, K-1, 0, 1, \ldots$

---

7. **(5 points)** Give a high-level description (but detailed enough) of an algorithm to list in ascending lexicographic order the subset of words in a TST (ternary search tree) that begin with a given prefix p. Assume that the given prefix and all words in the TST are strings built from 8-bit ASCII characters, with the usual ordering, furthermore, assume that the end-of-string symbol is \0, that is, the character which ASCII code is 0 (henceforth smaller than any other character).
