Name:

Penn State access ID (xyz1234) in the following box:

Instructions:

- All questions are weighted equally.
- Please clearly write your name and your PSU access ID (i.e., xyz1234) in the box on top of **every page**.
- Write your solutions only in the space provided. There is one answer sheet for each problem. **Do not write outside of the black bounding box**: the scanner will not be able to read anything outside of this box.
- We are providing one extra page at the end if you need extra space. Make sure you mention in the space provided that you answer continues there.
- **Important:** Brief but precise description of algorithms is required (try less than 6 sentences if you can, but don't fret if you go over). Pseudocode is not required, but you may provide it if you think it helps your exposition. Pseudocode alone will not receive any credits. Do not forget to provide the running time analysis of all your algorithms.

You have three hours. Good luck!

1. Fibonacci numbers

Recall that the Fibonacci numbers are defined by $f_0 = 0$, $f_1 = f_2 = 1$ and the recurrence relation $f_{n+1} = f_n + f_{n-1}$ for all $n \geq 1$.

(a) Show that $f_{n+1} < \left(\frac{7}{4}\right)^{n+1}$ for all $n \geq 1$.

(b) Show that $f_1 + f_2 + f_3 + ... + f_n = f_{n+2} - 1$ for all $n \geq 1$.

Hint: use mathematical induction.

2. Simple proofs

Prove the following statements.

(a) Show that $\log(n!) = \Theta(n \log n)$.

(b) Suppose we toss a fair coin independently n times. Therefore the probability of head for each coin toss is $1/2$. Show that the probability of having even number of heads after n-many tosses is exactly $1/2$ for any $n > 0$. *Hint: use mathematical induction.*

3. Rates of growth

List the following six functions so that they are in order from smallest to largest when n is very large. Indicate which functions are Θ of each other.

$$\begin{aligned} f_1(n) &= (\log n)^{\log n} \\ f_2(n) &= \log(n^n) \\ f_3(n) &= 2^n \\ f_4(n) &= n^3 \cdot n^{\log\log\log n} \\ f_5(n) &= (\log\log n)^n \\ f_6(n) &= n^{\log n} \end{aligned}$$

Note: To receive partial credit for a not entirely correct answer, you must show your work neatly, and your work should not simply consist of evaluating the functions at a large value of n in the hope that this value is large enough to reveal the correct answer.

4. Recurrences

Solve each of the following recurrences. Giving your solution in O-notation suffices. You may assume that n is of some special form (e.g., a power of two or of some other number), and that the recurrence has a convenient base case that is $\Theta(1)$.

(a) $T(n) = 5T(n/4) + n$

(b) $T(n) = T(n/2) + 1.5^n$

(c) $T(n) = \sqrt{n}T(\sqrt{n}) + n$

5. Graph algorithms

For a graph $G = (V,E)$, the second-best Minimum Spanning Tree is a spanning tree with the second minimum weight sum of all edges out of all spanning trees of graph G. Design an algorithm to find the second-best Minimum Spanning Tree of a given graph G. Include a justification of why your algorithm is correct and a run-time analysis.

Note: For full-credit your algorithm should run in $O(|V|^3)$ time or better.

6. Almost sorted

An array of distinct integers is called *decreasing k-sorted*, if each element of the array only needs to be moved at most k positions in order to sort the array in decreasing order. For example, the array given by $[9,10,6,8,7,3,2,5,4,1]$ is decreasing 2-sorted. Formally, an array is decreasing k-sorted if for every $i = 0,\ldots,n-1$, the position j of the element $A[i]$ when A is sorted in decreasing order is such that $|i-j| \leq k$.

Provide an algorithm that sorts an decreasing k-sorted array with n elements with running time $O(n\log k)$ time.

7. Finding Closest Point

Suppose we are given a list of n many 2-dimensional points, $(x_1,y_1),\ldots,(x_n,y_n)$ which all lies on a line parameterized by a linear equation $\alpha x + \beta = y$. Therefore for all $i \in [n]$, $\alpha x_i + \beta = y_i$. Furthermore, suppose we have $y_1 < \ldots < y_n$.

Suppose we are given an input point (x^*,y^*) which is guaranteed to be on the line, that is $\alpha x^* + \beta = y^*$. Define the distance between two points (x_i,y_i) and (x_j,y_j) as the usual Euclidean distance, that is $\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}$.

Give an algorithm which finds the closest point among $(x_1,y_1),\ldots,(x_n,y_n)$ to (x^*,y^*) with running time $O(\log n)$ time.

8. Dynamic Programming You are given a $4 \times n$ grid that has an integer value written in each square. You are also given a set of $2n$ rocks, and you want to place some of these on the grid squares (each rock can be placed on exactly one square) so as to maximize the sum of the integers that are covered by rocks. There is one constraint: for a placement of pebbles to be legal, no two of them can be on horizontally or vertically adjacent squares (diagonal adjacency is fine).

(a) Determine the number of legal patterns that can occur in any column (in isolation, ignoring the pebbles in adjacent columns) and describe these patterns.

Call two patterns compatible if they can be placed on adjacent columns to form a legal placement. Let us consider subproblems consisting of the first k columns $1 \leq k \leq n$. Each subproblem can be assigned a type, which is the pattern occurring in the last column.

(b) Using the notions of compatibility and type, give an $O(n)$-time dynamic programming algorithm for computing an optimal placement.

Name:	**PSU Access ID (xyz1234):**

Problem 1.

Name:	**PSU Access ID (xyz1234):**

Problem 2.

Name:	**PSU Access ID (xyz1234):**

Problem 3.

Name:	**PSU Access ID (xyz1234):**

Problem 4.

Name:	**PSU Access ID (xyz1234):**

Problem 5.

Name:	**PSU Access ID (xyz1234):**

Problem 6.

Name:	**PSU Access ID (xyz1234):**

Problem 7.

Name:	**PSU Access ID (xyz1234):**

Problem 8.

Name:	**PSU Access ID (xyz1234):**

Extra.