ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Midterm Exam, Algorithms 2015-2016

- 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 them. In particular, do not only give pseudo-code without explanations. A good guideline is that a description of an algorithm should be so that a fellow student can easily implement the algorithm following the description.
- **Do not touch until the start of the exam.**

Good luck!

Name: ______________________________ **N° Sciper:** __________________

Problem 1 / 16 points	Problem 2 / 15 points	Problem 3 / 23 points	Problem 4 / 23 points	Problem 5 / 23 points

Total / **100**

1 *(16 pts)* **Recurrences and Stacks.**

1a *(8 pts)* Give tight asymptotic bounds for the following recurrences (assuming that $T(1) = \Theta(1)$). You need not justify your answers.

(i) $T(n) = 2T(n/4) + \Theta(\sqrt{n})$

(ii) $T(n) = 10T(n/3) + \Theta(n)$

(iii) $T(n) = T(n/2) + T(n/3) + T(n/3) + \Theta(n^2)$

Solution:

1b *(8 pts)* Consider the following procedure UNKNOWN that takes as input an array $A[1 \ldots n]$ consisting of n letters and returns true or false.

```
UNKNOWN($A, n$)

Let $S$ be an empty stack
for $i = 1$ to $\lfloor n/2 \rfloor$
    PUSH($S, A[i]$)
for $j = \lceil n/2 \rceil + 1$ to $n$
    if $A[j] \neq$ POP($S$)
        return false
return true
```

What does UNKNOWN(A, n) return on input $A =$ and $n = 4$?

A	B	B	A

Solution:

What does UNKNOWN(A, n) return on input $A =$ and $n = 3$?

O	L	A

Solution:

In general, give a succinct characterization of the inputs for which the procedure returns true.

Solution:

2 *(15 pts)* **Divide and Conquer.** The increasing popularity of the Merge-Sort algorithm is largely due to it being parallelizable. This is a significant advantage when dealing with large data sets. Here, we will analyze a new variant of merge-sort, called Merge-Sort-Delux, that could potentially be even better for parallelization. Indeed, instead of partitioning the array recursively into two subproblems, we will partition the array into $\sqrt{n}$ subproblems (all of which could potentially be sorted recursively on different computers). The pseudo-code is as follows.

```
Merge-Sort-Delux($A, p, r$)
1. Let $n = r - p + 1$
2. if $n > 1$
3.       $k = \lceil \sqrt{n} \rceil$
4.       for $i = 1$ to $k$
5.             Merge-Sort-Delux($A, p + \lfloor \frac{n}{k} \cdot (i-1) \rfloor, p + \lfloor \frac{n}{k} \cdot i \rfloor - 1$)
6. Merge the $k$ sorted arrays

             $A\left[p \ldots \left(p + \lfloor \frac{n}{k} \rfloor - 1\right)\right]$
             $A\left[\left(p + \lfloor \frac{n}{k} \rfloor\right) \ldots \left(p + \lfloor \frac{n}{k} \cdot 2 \rfloor - 1\right)\right]$
                 $\vdots$
             $A\left[\left(p + \lfloor \frac{n}{k} \cdot (k-1) \rfloor\right) \ldots r\right]$

   into one sorted array $A[p \ldots r]$.
```

Recall from the exercises that k sorted arrays containing n elements in total can be merged in time $\Theta(n \log k)$ and thus Step 6 takes time $\Theta(n \log k)$. We also assume that $\sqrt{n}$ can be calculated in constant time.

2a *(6 pts)* Let $T(n)$ be the time it takes to execute Merge-Sort-Delux(A, p, r) on a single computer, where $n = r - p + 1$ is the number of elements in the array. **Give the recurrence relation** for $T(n)$. To simplify notation, you may assume that $\sqrt{n}$ and n/k are integers.

Solution:

2b *(9 pts)* **Prove** that $T(n) = O(n \log n)$ using the substitution method.

Solution:

3 *(23 pts)* **Binary Search Trees.**

3a *(7 pts)* We consider the task of *reconstructing* a binary search tree from the output of a postorder walk. **Illustrate/draw** the binary search tree T for which the output of Postorder-Tree-Walk($T.root$) is $3, 2, 1, 9, 8, 13, 14, 19, 12, 5$.

Solution:

3b *(16 pts)* **Design** and **analyze** an algorithm for the following problem:

Input: An array $A[1 \ldots n]$ consisting of n *different* integers.

Output: A binary search tree T of height $\lfloor \log_2(n) \rfloor$ with the integers in A as keys (the tree should contain exactly one key for each element of A).

Your algorithm should run in time $O(n \log n)$.

Solution:

Continuation of the solution to 3b:

4 *(23 pts)* **Dynamic Programming.** In this problem you are going to help Mourinho get the Chelsea football club back on track. In particular, you should design an algorithm for buying the cheapest set of players so that they form a "good team": their sum of skills should be at least a threshold T. In our abstract model, we assume that a player's skill can be characterized by a single integer. The formal definition of our problem is as follows:

INPUT: A set $\{1, 2, \ldots, n\}$ of n players where each player i is characterized by the following data:

- $s_i \geq 0$ — an integer describing player i's skill,
- $p_i \geq 0$ — an integer describing the price of buying player i.

In addition, we are given a non-negative integer T, which is the required sum of skills of the new players.

OUTPUT: The smallest cost C such that there exists a subset $P \subseteq \{1, 2, \ldots, n\}$ of players satisfying

$$\sum_{i \in P} p_i = C \qquad \text{and} \qquad \sum_{i \in P} s_i \geq T.$$

If no subset P with the required sum of skills exists, the algorithm should output ∞.

4a *(13 pts)* Let $c[i, t]$ be the minimum cost of a solution to the instance consisting only of the first i players $\{1, 2, \ldots, i\}$ and with total skill requirement t. In other words, $c[i, t]$ is the smallest cost such that there exists a subset $P \subseteq \{1, 2, \ldots, i\}$ satisfying

$$\sum_{i \in P} p_i = c[i, t] \qquad \text{and} \qquad \sum_{i \in P} s_i \geq t.$$

(Or, simply ∞ if no such subset P exists.)

Complete the recurrence relation for $c[i, t]$ that can be used for dynamic programming. Also motivate your answers by explaining your reasoning.

Solution:

Continuation of the solution to 4a:

4b *(10 pts)* Consider the following instance of our problem:

i	1	2	3	4	5	6
s_i	2	1	3	4	4	2
p_i	3	1	7	7	6	2

$n = 6$ and $T = 5$.

Use the recurrence relation to return the optimal solution by **filling in the table** of $c[i, t]$ values below (in a bottom-up dynamic programming fashion). Also, in general, **what is the running time** (in Θ-notation) of a bottom-up dynamic programming implementation as a function of n and T?

Solution:

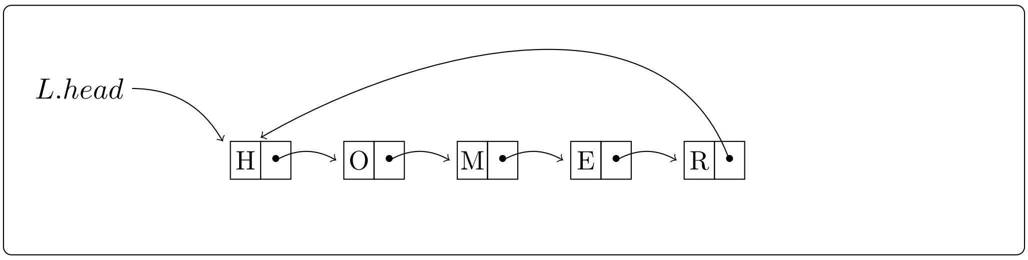
Table:

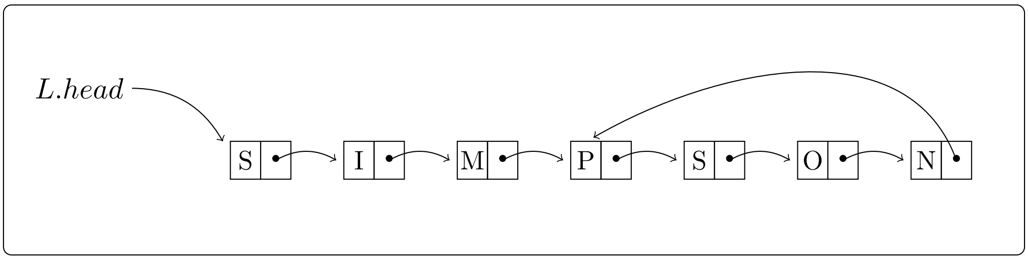
T i	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						
6						

The asymptotic running time (as a function of n and T) of solving the problem in a bottom-up fashion is

$$\Theta\Big(\underline{\qquad\qquad\qquad\qquad}\Big)$$

5 *(23 pts)* **Homer Simpson's crazy-lists.** Homer Simpson is not exactly known as the Einstein of Springfield. In spite of this and unfortunately for us, he has decided to design a linked-list data structure, which we call crazy-lists. A crazy-list is like a single-linked list with the following important exception: the last element's pointer points to a previous element in the list instead of being nil. Two examples of crazy-lists are as follows:





Design and **analyze** an algorithm that takes as input a crazy-list (i.e., a pointer $L.head$) and outputs the number n of elements in that list. The algorithm is *not* allowed to modify the input. In addition, your algorithm should **run in time $O(n)$** and **use a constant amount of *extra* space** (not counting the memory for storing the list).

If you do not solve the whole problem, you are encouraged to write down your best (partial) solution.

Hints: Use the "tortoise/turtle (slow) and hare (fast)" technique to detect the cycle. Then calculate the length of the cycle. After that calculate the length of the path up to the cycle.

Solution:

Continuation of the solution to 5: