ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Midterm Exam, Algorithms 2014-2015

- 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. In particular, do not only give pseudocode 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 / 25 points	Problem 2 / 25 points	Problem 3 / 25 points	Problem 4 / 25 points

Total / 100

1 *(25 pts)* **Recurrences, Stacks, and Trees.**

1a *(9 pts)* Give tight asymptotic bounds for the following recurrences (assuming that $T(1) = \Theta(1)$):

(i) $T(n) = 2T(n/4) + \Theta(1)$

(ii) $T(n) = 2T(n/4) + \Theta(n)$

(iii) $T(n) = 8T(n/4) + \Theta(n)$

(iv) $T(n) = 32T(n/4) + \Theta(n^{2.5})$

Solution:

1b *(8 pts)* Consider the following procedure UNKNOWN that takes as input an integer $n \geq 0$.

```
UNKNOWN(n)
Let S be an empty stack
PUSH(S, 1)
PUSH(S, 1)
while n > 1
      tmp1 = POP(S)
      tmp2 = POP(S)
      PUSH(S, tmp1)
      PUSH(S, tmp1 + tmp2)
      n = n − 1
return POP(S)
```

Write a recursive formulation of UNKNOWN(n), i.e., write UNKNOWN(n) as a function of UNKNOWN(0), UNKNOWN(1), …, UNKNOWN($n-1$) whenever $n \geq 2$. Also indicate the value of UNKNOWN(n) when $n = 0$ and $n = 1$.

Solution:

1c *(8 pts)* **Illustrate/draw the binary search tree** obtained by executing

1. Let T be an empty binary search tree
2. Tree-Insert$(T, 9)$
3. Tree-Insert$(T, 5)$
4. Tree-Insert$(T, 2)$
5. Tree-Insert$(T, 12)$
6. Tree-Insert$(T, 13)$
7. Tree-Insert$(T, 7)$
8. Tree-Insert$(T, 10)$

Is it a good binary search tree for the given set of keys? Motivate your answer.

Solution:

2 *(25 pts)* **Divide-and-Conquer.** Consider the procedure POWER that takes as input a number a, a non-negative integer n and returns a^n:

```
POWER(a, n)
if n = 0
      return 1
if n = 1
      return a
q = ⌊n/4⌋ + 1
return POWER(a, q) · POWER(a, n − q)
```

2a *(10 pts)* Let $T(n)$ be the time it takes to invoke POWER(a, n). Then the recurrence relation of $T(n)$ is

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 0 \text{ or } n = 1, \\ T(\lfloor n/4 \rfloor + 1) + T(n - \lfloor n/4 \rfloor - 1) + \Theta(1) & \text{if } n \geq 2. \end{cases}$$

Prove that $T(n) = O(n)$ **using the substitution method**.

In your proof you may ignore the floor function, i.e., you can replace $\lfloor n/4 \rfloor$ by $n/4$.

Solution:

Do not forget subproblem 2b on page 6.

Continuation of the solution to 2a:

2b *(15 pts)* **Design** and **analyze** a modified procedure FASTPOWER(a, n) that returns the same value a^n but runs in time $\Theta(\log n)$.

A solution that only works when n is a power of 2, i.e., $n = 2^k$ for some integer $k \geq 0$, gives partial credits.

(Note that a^n is *not* a basic instruction and should therefore not be used.)

Solution:

3 *(25 pts)* **Dynamic Programming.** Consider the weighted version of the classic change making problem that addresses the following question: how can a given amount of money be made with the least weight of coins of given denominations and weights? The formal definition is as follows:

> **INPUT:** a set of n integer coin values $\{v_1, v_2, \ldots, v_n\}$ with associated weights $\{w_1, w_2, \ldots, w_n\}$ and a positive integer T. The coin values satisfy $v_1 = 1$ and $v_i \leq v_{i+1}$ for $i = 1, \ldots, n-1$.
>
> **OUTPUT:** The smallest weight W such that there exist non-negative integers $x_1, x_2, \ldots, x_n$ satisfying
>
> $$\sum_{i=1}^{n} x_i \cdot v_i = T \qquad \text{and} \qquad \sum_{i=1}^{n} x_i \cdot w_i = W.$$
>
> Here x_i stands for the number of times the coin of value v_i is used to achieve the total value T.

An example input is the following:

$T = 7$ and there are $n = 3$ coin values:

i	1	2	3
v_i	1	2	5
w_i	6	14	29

The correct output to the above input is 41 as the smallest weight change is to use $x_1 = 2$ coins of value v_1 and $x_2 = 0$ coins of value v_2 and $x_3 = 1$ coin of value v_3.

3a *(10 pts)* Let $r[t]$ equal the minimum weight W_t required to achieve a total value of t, i.e., such that there exist non-negative integers $x_1, x_2, \ldots, x_n$ satisfying

$$\sum_{i=1}^{n} x_i \cdot v_i = t \qquad \text{and} \qquad \sum_{i=1}^{n} x_i \cdot w_i = W_t.$$

Complete the recurrence relation for $r[t]$ that can be used for dynamic programming.

Solution:

$$r[t] = \begin{cases} \infty & \text{if } t < 0 \\ \rule{8cm}{0.4pt} & \text{if } t = 0 \\ \rule{8cm}{0.4pt} & \text{if } t > 0 \end{cases}$$

3b *(15 pts)* Use either the bottom-up approach or top-down with memoization to **design an efficient algorithm** for the weighted change making problem. You should also **give a tight analysis** of the running time of your algorithm.

(write your solution to 3b on next page)

Solution to 3b:

4 *(25 pts)* **Heaps.** Consider the following problem:

> **INPUT:** A positive integer k and an array $A[1 \dots n]$ consisting of $n \geq k$ integers that satisfy the max-heap property, i.e., A is a max-heap.
>
> **OUTPUT:** An array $B[1 \dots k]$ consisting of the k largest integers of A sorted in non-decreasing order.

Design and **analyze** an efficient algorithm for the above problem. Ideally your algorithm should run in time $O(k \log k)$ but the worse running time of $O(\min\{k \log n, k^2\})$ is also acceptable.

Slightly slower algorithms give partial credits.

Solution:

Continuation of the solution to 4: