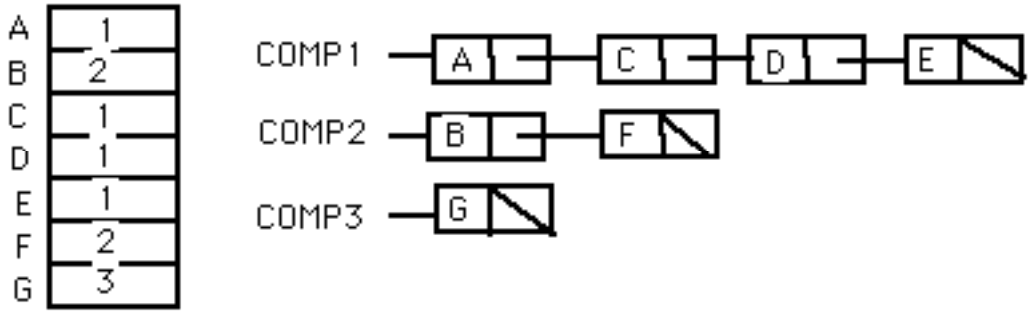
CS2223
HW#6

DUE: Tuesday, December 5

1. (10 points) One implementation of a data structure to store disjoint sets which supports MAKE-SET, UNION and FIND-SET is the following data structure that we discussed in class. The data structure stores, for each set, a linked list of the elements which belong to the set. It also stores, for each set, the number of elements in the set. It performs a UNION by copying the smaller list into the larger list. Following is the state of the data structure after having implemented:

MAKE-SET(A)
MAKE-SET(B)
MAKE-SET(C)
MAKE-SET(D)
MAKE-SET(E)
MAKE-SET(F)
MAKE-SET(G)
UNION(A,C)
UNION(B,F)
UNION(D,E)
UNION(C,E)



a What is the worst-case time to implement a sequence of *n* MAKE-SET operations followed by a sequence of *n*-1 UNION operations? Justify your response.

b Show that your bound of part ***a*** is tight by showing a sequence of *n* MAKE-SET operations followed by a sequence of *n*-1 UNION operations which require this much time.

2. (7 points) *Arbitrage* (http://en.wikipedia.org/wiki/Arbitrage) Suppose you know an arbitreur who can convert each of n currencies intro exactly one other currency, in a fixed sequence. For example, for n=4 and currencies Hungarian Forints, c_1, Euros, c_2, Jordanian Dinars, c_3 and US \$, c_4, the conversion rates on November 26, 2006 were $1*c_1=0.003855*c_2$, $1*c_2=0.927635*c_3$ and $1*c_3=1.41143*c_4$. She could convert $\$\lambda$ to $198.1*\lambda$ Forints, and then buy $0.003855*198.1*\lambda$ Euros, and then buy $0.927635*0.003855*198.1*\lambda$ Dinars, and then buy $\$1.41143*0.927635*0.003855*198.1*\lambda = \$\ 0.999883331*\lambda$. In helping her maximizde her money, she has to solve the following subproblem:

For input n and list $(a_1, a_2, ..., a_n)$, $a_i \in \Re^+, 1 \le i \le n$, we want to determine $i, j, 1 \le i \le j \le n$ to satisfy $\max_{1 \le i \le j \le n} \prod_{i \le k \le j} a_k = \max_{1 \le i \le j \le n} a_i * a_{i+1} * ... * a_j$.

Determine a linear time algorithm, $O(n)$, to solve the above subproblem, which will help the arbitreur maximize her money.

3. (10 points) You are helping design a scheduler for two computers, C_1 and C_2. Your input is $n \ge 1$ processess, and associated with each process is the time for the process to execute, $(a_1, ..., a_n), a_i \in \mathbb{Z}^+, 1 \le i \le n$. The execution of the processes can not be preempted, that is, if a process needing a_i units of time is executed on C_j, then it must stay on C_j for the entire a_i units of time. As output you must determine whether or not there is a *viable schedule*, that is, a schedule such that the processes can be assigned to a computer and they all finish execution in $\sum_{1 \le i \le n} a_i / 2$ units of time. For example, if n=5 and the execution times are $(a_1, a_2, a_3, a_4, a_5) = (6, 4, 3, 6, 5)$, then a viable schedule is to execute jobs 1 and 4 on C_1 and jobs 2, 3 and 5 on C_2, but if $(a_1, a_2, a_3, a_4, a_5) = (6, 4, 6, 6, 5)$ then there does not exist a viable schedule. Your algorithm (pseudocode suffices) must execute in worst case time in $O\left(n \sum_{1 \le i \le n} a_i\right)$.

Hint: Be lazy.

CS2223
HW#6 SOLUTIONS

1. In the worst-case, we always merge (via UNION) two sets of equal size.

a The worst-case time for the n MAKE-SETs is in $\Theta(n)$, and the worst-case time for the n-1 UNIONs is in $O(n \lg n)$, since each element is merged into a set as least as large, and hence has the name changed of the component to which it belongs at most lg$n$ times.

b For this example, we assume that n is a power of 2.

MAKE-SET(x_1)	n of these
MAKE-SET(x_2)	
....	
MAKE-SET(x_n)	
UNION(x_1,x_2)	n/2 of these
UNION(x_3,x_4)	
....	
UNION(x_{n-1},x_n)	
UNION(x_1,x_3)	n/4 of these
UNION(x_5,x_7)	
...	
UNION(x_{n-3},x_{n-1})	
	...
UNION(x_1,$x_{n/2+1}$)	1 of these

2. *Best_to*[0] $\leftarrow$ 1
for $j \leftarrow 1$ **to** n **do**
Best_to[j] $\leftarrow$ max(1, a_j***Best_to*[j-1])

3. There is a viable schedule if and only if there's a solution to the instance of the KNAPSACK PROBLEM with $v_i = w_i = a_i, 1 \le i \le n$ and capacity $W = \sum_{1 \le i \le n} a_i / 2$ of value $\sum_{1 \le i \le n} a_i / 2$.