CS 466/666, Spring 2007
Timothy Chan

Assignment 1 (due May 23 Wednesday noon)

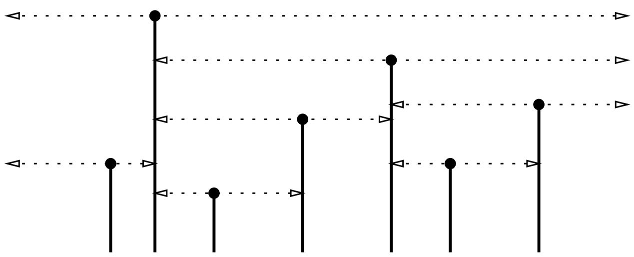
Please read `http://www.student.cs.uwaterloo.ca/~cs466/policies.html` first for general instructions.

1. [*8 marks*] Given a collection of n subsets $S_1, \ldots, S_n \subseteq \{1, 2, \ldots, n\}$, we want to decide whether there exists two subsets S_i and S_j ($i \neq j$) such that $S_i \subseteq S_j$. Show that this problem can be solved in time $O(n^{2.81})$ (or $O(n^{2.376})$).

   Example: if $S_1 = \{1, 2, 4\}$, $S_2 = \{2, 3\}$, $S_3 = \{2, 4, 5\}$, $S_4 = \{1, 3, 4, 5\}$, $S_5 = \{1, 4\}$, then the answer is yes since $S_5 \subseteq S_1$.

2. [*20 marks*] Define a *stick* to be a vertical line segment where the bottom endpoint lies on the x-axis, i.e., a line segment from $(x_i, 0)$ to (x_i, y_i) for some x_i, y_i with $y_i > 0$.

   We are given a collection of n sticks. For every top endpoint (x_i, y_i), we would like to determine the first stick directly to the left of (x_i, y_i) (if one exists) and the first stick directly to the right of (x_i, y_i) (if one exists), as shown in the figure below.



   (a) [*12 marks*] Give an $O(n \log n)$-time algorithm for this problem. Use the following approach: sort the sticks in increasing order of x-coordinate, and insert sticks one at a time in that order.

   Hint: this is similar to Graham's scan; in particular, the analysis involves a similar amortization argument.

   (b) [*8 marks*] Prove an $\Omega(n \log n)$ lower bound for this problem among comparison-based algorithms.

   Hint: use a reduction involving sorting.

3. [*22 marks*] A *stack* is a data structure that supports push (inserting an element to the end of a list) and pop (deleting the last element of the list). There are two implementations you all probably know (from CS240): (i) array or (ii) linked list. Both approaches require only $O(1)$ update time but have drawbacks regarding space: In (i), we need to reallocate/rebuild the array whenever the number of elements n exceeds the array size, so typically a larger-than-necessary array size is used. In (ii), we need an extra pointer per element, resulting in $\Theta(n)$ extra space.

In this question, we suggest a refined approach that requires much less extra space—$O(\sqrt{n})$, as we will see. Our new implementation uses a *linked list of arrays*, denoted $\langle A_1, A_2, \ldots \rangle$, where the j-th array A_j has size exactly j. Elements of A_1 are filled first, then A_2, then $A_3 \ldots$ Let A_k be the last nonempty array. We do not necessarily destroy an array when it becomes empty: we allow for the possibility that A_{k+1} has been created (due to previous operations) but contains no elements and is not yet destroyed. However, we maintain that $A_{k+2}, A_{k+3}, \ldots$ are destroyed. The details are described in the pseudocode below:

initialize():
0. $i = k = \ell = 0$

push(x):
1. if $i < k$ then {
2. $\quad A_k[i+1] = x, \quad i = i + 1$
}
3. else {
4. $\quad$ if A_{k+1} does not exist then create a new array A_{k+1} of size $k+1$
5. $\quad A_{k+1}[1] = x, \quad i = 1, \quad k = k + 1$
}

pop():
6. if $i > 1$ then $i = i - 1$
7. else {
8. $\quad$ if A_{k+1} exists then destroy A_{k+1}
9. $\quad i = k - 1, \quad k = k - 1$
}

(a) [*5 marks*] Show that the amount of space for the entire data structure is at most $n + O(\sqrt{n})$ at any time, where n denotes the current number of elements.
Hint: $1 + 2 + \cdots + k = \Theta(k^2)$.

(b) [*2 marks*] What is the worst-case cost of push and pop as a function of n?
Note: assume that the cost of creating/destroying an array of size j is $\Theta(j)$.

(c) [*10 marks*] Prove that the amortized cost of push and pop is $O(1)$. Use the potential method with the following definition of potential:

$$\Phi = \begin{cases} i & \text{if } A_{k+1} \text{ does not exist} \\ k - i & \text{if } A_{k+1} \text{ exists} \end{cases}$$

Carefully show your work.

(d) [*5 marks*] Show that if we change line 8 to

8′. $\quad$ destroy A_k

(in other words, we always destroy an array as soon as it becomes empty), then the amortized cost of push and pop is no longer $O(1)$, by giving a specific sequence of $O(n)$ operations whose total cost is much larger than $O(n)$.