Please print your name at the top of the page. The following four pages (numbered 2-5) contain four problems with weight (approximately equal to the time needed, in minutes) given within brackets [50]. Please write your answers in the space provided; the amount of space corresponds to the expected length of the answer. Please write legibly! If you want to show intermediate steps for partial credit, do it on the reverse and indicate it in your answer. A good strategy may be to read all the problems first, and then start with those that seem to be easiest.

Please write legibly!
1. [10] What is a **tight** bound on the time complexity of each of the following program segments? (Hint: The asymptotic growth rate of the final value of \( m \) as a function of \( n \) reflects the time complexity. You can show your work by writing a summation formula for \( m \).)

(a) \( m:=0; \)
\[ \text{for (int } i=0; i<n; i++) \]
\[ \text{for (int } j=0; j<i*i; j++) \]
\[ m++; \]
\[ T(n) \in \]

(b) \( m:=0; \)
\[ \text{for (int } i=0; i<n*n; i++) \]
\[ \text{for (int } j=0; j<i; j++) \]
\[ m++; \]
\[ T(n) \in \]
2. [12] *Binary heap* is an almost complete binary tree with the **min**-heap property. It is stored in an array $A$.

a) Draw the sequential implementation of the binary heap resulting from the bottom-up construction in the array $A$ containing elements with priorities 15..7, in this order, in locations 1 through 9. Use the space below to indicate contents of the array after each swap in the bubbling-down process. The construction starts with location $A[4]=12$. (The first row represents the initial content of the array.)

(b) Draw the contents of the heap, as represented by the array $A$, in each swap after the deletion of the minimum priority element from the final heap in (a) above.
3. [13] Give pseudo-code which, given an unsorted array $A$ containing integers in locations 1 to $n$, will return the $k^{th}$ smallest element. Use heap operations - the running time of your algorithm should be $O(n + k \log n)$. 
4. [15] The balance factor of a node \( v \) in a binary tree is defined to be the height of the left subtree of \( v \) minus the height of the right subtree of \( v \). The height of a null node is defined to be \(-1\). Suppose you are given a binary tree \( T \) whose nodes have a balance factor field \( v.balfac \) which has not yet been initialized. Write pseudo-code which will correctly assign values to the balance factor fields of all the nodes of \( T \).

Your algorithm should run in linear time. In addition to \( v.balfac \), you will be using the fields \( v.lchild, v.rchild, \) and \( T.root \).

You may not assume that the computed height of the subtree rooted at \( v \) is stored in \( v \). You may assume that \( v \) has a field \( v.height \) if this helps, but it is not necessary for the algorithm.