OLD FINAL EXAMINATION QUESTIONS — COMPUTER SCIENCE 308-251
INTRODUCTION TO COMPUTER SCIENCE

(Luc Devroye)
(January 4, 1996)

GENERAL QUESTIONS.

Question 1. An operating system must manage a pool of n jobs, where each job in our context is characterized by a pair of numbers, `(imp, time)`, where `imp` is a number reflecting the importance of the owner (small is important, large is unimportant), and `time` is the time of submission of the job. A job is said to be **eligible for processing** if no other job in the pool is both more important and submitted earlier. Picture a job as a point in the `(imp, time)` space. Then eligible jobs are those near the lower left: if you move the coordinate axes to any eligible point, then the lower left quadrant cannot contain any points (jobs). Assume for the sake of simplicity that no two values `imp` and `time` are identical, and that the n pairs of values are stored in an array of records. Give an $O(n \log(n))$ worst-case algorithm for identifying (printing) all eligible jobs in the array. Please use a mixture of PASCAL constructs and prose. It is understood that standard algorithms and data structures are known (e.g., stack and heap operations, sorting procedures, tree traversals, and so forth can be used without further explanation).

Question 2. **Beap.** In 1976, Munro and Suwanda proposed the **beap** (biparental heap) as a possible implementation for a priority queue-like ADT supporting the operations `makenull`, `insert`, `deletemin` and `search`. A beap of n elements should be visualized as a triangular matrix such as the one shown below which has 10 elements. The elements are added in cross-diagonal fashion, so that the inherent order is 1, 2, 4, 6, 5, 9, 21, 8, 20 and 13 in the example shown below.

1	4	9	13	...
2	5	20		...
6	8			...
21				...

The beap has the property that along any row or column, the elements are encountered in increasing order. Each element can be thought of as having two coordinates, say (c, r) where c stands for "column" and r stands for "row". We use the convention that the first element is at $(0,0)$. In our example, the element at $(2,1)$ is 20, and the last element, at $(3,0)$, is 13. If we place elements in cross-diagonal fashion, then the n-th element ends up at coordinates $(c(n), r(n))$, where $c(.)$ and $r(.)$ are certain simple functions of n that are given. Thus, $(c(1), r(1)) = (0,0)$, $(c(2), r(2)) = (0,1)$, $(c(3), r(3)) = (1,0)$, $(c(4), r(4)) = (0,2)$, and so forth. Define a pointer-less super-PASCAL implementation of the beap (assume that n never exceeds "maxnum"):

```
type
beap =
```

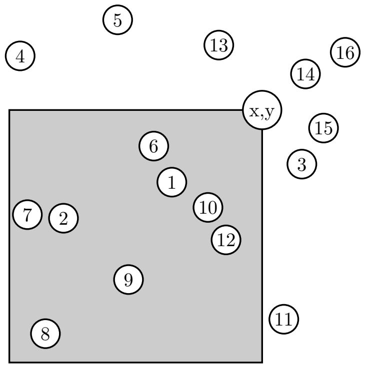
Now think about how you might implement `makenull`, `insert`, `delete` and `deletemin`. Now you should be ready to tackle the following analytical questions.

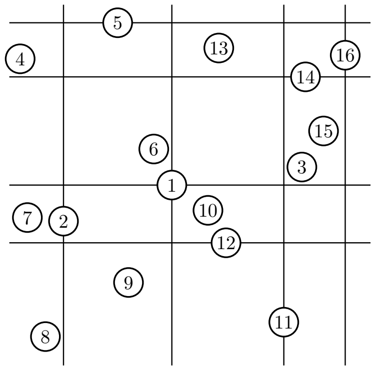
Give O and Ω statements about the worst-case complexity of your operations `insert` and `deletemin`.	
Give the worst-case time complexity of `beapsort`, a sorting method in which n elements are inserted in a beap, and then deleted by n calls to `deletemin`.	
For use as a future event set in discrete event simulation, would you prefer a beap over a heap?	

The **heap**, the **sorted array**, the **unsorted array** and the **beap** are four structures capable of holding elements with the property that the position of an element in the structure contains information about the relative size of the element vis-a-vis some other elements. This property is not shared by pointer-based structures. Of the four structures cited above in boldface, which one is best, in the worst-case complexity sense, for the following tasks:

`makenull`, followed by n `insert`'s, and then n `deletemin`'s.	
One operation `member` in a structure holding n elements.	
`makenull`, followed by n `insert`'s and n `member`'s.	

Question 3. ALGORITHMS. This question deals with an ADT for two-dimensional data sets. In particular, we are given n two-dimensional vectors that we shall denote by $(x_1, y_1), \ldots, (x_n, y_n)$. No two x or y values are identical. What is needed is a simple structure for handling queries of the form `belowleft (x,y)`, where `belowleft` returns the number of data points for which jointly $x_i \leq$`x` and $y_i \leq$`y`. Without any special set-up we can always perform `belowleft` in $O(n)$ time. As we will see below, one can do much better if one is willing to bear a one-time $O(n \log n)$ initialization cost for `set-up`. After that, each `belowleft` operation takes time well below $O(n)$. For `set-up`, we sort the data by x-coordinates and by y-coordinates separately. This yields two arrays of size n. Assume for simplicity that n is a perfect square, and define $k = \sqrt{n}$. In each array, mark elements $k, 2k, 3k, \ldots, n$ and rename these elements $X_1, \ldots, X_k$, and $Y_1, \ldots, Y_k$ respectively. A cell in the plane is a rectangle of the form $(X_{i-1}, X_i] \times (Y_{j-1}, Y_j]$. In other words, if you take the k by k grid defined by verticals at the X_i's and horizontals at the Y_j's, you obtain $k^2 = n$ cells.





1	1	1	1
0	1	0	3
2	1	1	0
1	1	2	0

0	3	6	9
0	3	5	8
0	1	2	4
0	0	0	0

In this example, n=16 and k=4. Top left figure shows the cloud of data points. The points counted in `belowleft (x,y)` are highlighted—the function returns 8 in this example. At the bottom left, you first find the partition of the space induced by X_1,X_2,X_3,X_4 and Y_1,Y_2,Y_3,Y_4. In the middle, we show the cardinalities of the cells. Clearly, all rows and all columns have totals that must be k=4. At the right, the cell number indicates the total number of elements in all cells whose row and column are below and to the left of the given cell.

A. Show how (starting from scratch) you can determine in $O(n \log n)$ time the cardinalities of all cells, that is, the number of (x_i, y_i) points that fall in each cell.

B. Fix a cell c. Let N be the number of data points in cells that are both strictly to the left and strictly below cell c (thus, the row number and column number of all these cells must be strictly less than that of c). In the rightmost figure above, the numbers N are shown for all cells. Show how you can find all values of N for all n cells in $O(n)$ time after you have finished the computations suggested under part A.

C. The numbers N are stored with the cells in part B. This completes the set-up of the data structure. It will be used in a static fashion to answer only one type of query, `belowleft (x,y)`. Give a type definition for a PASCAL data structure for the whole thing so that `belowleft (x,y)` can be performed efficiently. Coordinate your answer with that of part D.

D. Make a claim about the worst-case time of `belowleft (x,y)`, and give an algorithm and an explanation of your claim. Coordinate your response with that of part C.

QUESTIONS ON STACKS.

Question 4. When writing a PASCAL program involving array implementations of a stack, how should a good programmer pass the stack to the procedure MAKENULL:

As a VAR parameter.
As a regular (not VAR) parameter.
Through a global variable (no parameter).

When writing a PASCAL program involving array implementations of a stack, how should the stack be passed to the function EMPTY by a good programmer:

As a VAR parameter.
As a regular (not VAR) parameter.
Through a global variable (no parameter).

QUESTIONS ON PRIORITY QUEUES (NOT HEAPS).

Question 5. **Heaps.** In a standard heap, we would like to implement the operation `delete`. Our choices are:

A. Mark the element as in `lazy-delete`, and remove the element when it reaches the root.
B. Give the element the value $-\infty$, let it trickle up to the root, and remove it as in a `deletemin`.
C. Give the element the value $+\infty$, let it trickle down to the leaf level, remove it, fill the hole with the last element, and let this element trickle up again.

Question 6. Priority queues. A sophisticated computer system must be able to accomodate many users simultaneously. Each job is characterized by three integer numbers:

ID:	user's identification number;
TIME:	time of job submission (integer between 0000 and 9999);
PR:	priority of job (integer in $1 \ldots 6$, where 6 is the highest priority).

Jobs are submitted at random time intervals and are put in a waiting line. The computer takes jobs from the waiting line (if it is not empty) according to the rule: highest priority job comes first; if there is a tie in priority, take the job that was submitted earliest. After execution of a job, a new job is fetched. The operating system will need two subprograms:

1) ADD(ID,TIME,PR) (addition to a waiting line).
2) TAKE(ID) (take ID from the waiting line). Assume that the waiting line is known to be limited to 400 jobs at any given time. Give a good data structure (with drawing) for organizing the job submission/job fetching processing. Initialize it, and give very brief descriptions (prose is acceptable) of your subprograms.

DEFINITIONS.

Question 7. Definitions. Define the following terms (be precise but concise): pixel double hashing Hu-Tucker algorithm future event set digital search tree path compression Polish notation digraph traversal height of a node AVL tree Ackermann's function topological sorting `dag` linear congruential sequence `disjoint_set` open addressing

ANALYSIS.

Question 8. Euclid's famous algorithm for determining the greatest common divisor of two integers m, n is given below.

```
function euclid (n,m:  integer); note:  n ≥ m
  begin
  while m > 0 DO (n,m) ← (m,n  (mod m));
  return n
  end
```

Example: if we start with $(72, 66)$, then we get the sequence $(66, 6)$, $(6, 0)$. The answer is 6. The following is always true: $n \pmod m < n/2$. Using this fact, give the worst-case complexity as a function of n and m in $O(.)$ format.

Question 9. Please indicate the worst-case complexities of the following operations. Just fill in n or $\log(n)$ or whatever is appropriate. It should be understood that all answers are both O and Ω of this function of n. In all cases, assume that the algorithm used in the answer is the best algorithm in existence.

OPERATION	YOUR ANSWER
Time needed to give an ordered listing of all elements stored in a given (but possibly badly balanced) binary search tree.	
Quicksort.	
Computing the rank of the root in a red-black tree in which we store only left, right and parent pointers as well as colors of the nodes.	
Membership check in red-black tree tree.	
Membership check in an AVL tree.	
Evaluation of a postfix expression of length n.	
Finding the lefmost leaf in a leftist heap.	
Finding the rightmost leaf in a leftist heap.	
Union of two possibly overlapping sets implemented as unsorted linked lists.	
Time needed to evaluate the function $f(n, 3)$ when $f(n, m)$ is computed based on a recursive algorithm with $f(n, m) = n * f(n-1, m) + (f(n-1, m-1))^2$, and $f(n, n) = f(n, 0) = 1$ for all n.	
Construction of a complete binary search tree (array implementation) from a sorted array.	
Membership check in a complete ternary tree.	

QUESTIONS ON DUAL DATA STRUCTURES.

Question 10. Give a type definition for a data structure that can solve an equivalence class problem, i.e., one that is geared towards the ADT `disjoint_set`, where the elements are all of the type "string", and the total number of elements is initially unknown.

Question 11. We need to implement the following ADT: the mathematical model is a collection of countries, which in turn are just sets of cities. The operations are

`join`: join two countries into a new (bigger) country.
`makenull`: create a country with no cities.
`makeone`: create a country with one given city.
`listing`: provide a list of all cities in a given country.

Explain in a few lines of prose how you would implement this and why you picked your implementation.

QUESTIONS ON SORTING.

Question 12. **Quicksort** was repeatedly timed on problems of size $1024 = 2^{10}$, and the average time was 1000 μs. What average time would you predict for problems of size $8192 = 2^{13}$?

Question 13. Show that the linear worst-case median-of-five algorithm remains linear if the five is replaced by seven. I am looking for the solution of a recursive equation, together with two lines of explanation of the terms in the equation.

Question 14. Consider two sorted lists of lengths m and n respectively. The implementation of these lists is not specified. Someone has written a comparison-based algorithm for for merging these lists into one sorted list of length $m+n$. In the worst case, the number of comparisons is at least equal to some function of n and m. (To draw an analogy, the number of comparisons for any sorting method is $\geq \lfloor \log_2 n! \rfloor$ in the worst case.) Give as good a lower bound as you can cook up.

QUESTIONS ON TRIES.

Question 15. Draw the `patricia` tree based upon the binary expansions of the numbers 7/64, 1/8, 19/64, 3/16 and 3/32 when inserted in that order.

Question 16. Draw the digital search tree based upon the binary expansions of the numbers 7/32, 5/64, 1/4, 5/16 and 3/32 when inserted in that order.

Question 17. Consider a trie based upon a decimal alphabet, used for storing 8-digit integers. The formal `TYPE` definition is

```
TYPE
    cell=array[0..9] of @cell;
    TRIE=@cell;
```

Write a one-line PASCAL statement which returns the value "true" in a boolean variable x when the structure contains at least one integer starting with 609 (such as 60987873, or 60900318), and the value "false" otherwise:

QUESTIONS ON HEAPS.

Question 18. Heaps. Heaps are often used when `deletemin` and `insert` are important operations. Are the worst-case complexities of these operations preserved if we replace the heap by

a 2-3 tree	Yes	No
a binary search tree	Yes	No
an ordered linked list	Yes	No
an AVL tree	Yes	No
a leftist heap	Yes	No

(answer yes or no in each row).

Question 19. Consider the following array with 7 elements:

7	3	1	4	6	2	5

Element one has value 7, and so forth. When we apply the standard procedure **buildheap** to this array to make it into a heap with minimum on top (minimum at root), then the heap, in array form, is

QUESTIONS ON DISJOINT SETS.

Question 20. Consider the tree-of-parentpointers implementation for the abstract data type `disjoint_set`. True or false:

QUESTION	ANSWER
The worst-case time of FIND can always be improved by path compression.	
The MERGE operation is $O(1)$ in the worst case.	
With the weighting rule, used when $n = 2000$, it is possible to have parentpointer trees of height at least 13.	

Question 21. Disjoint set ADT. The disjoint set ADT is implemented with the aid of a forest of parentpointer trees: in our case, the elements are drawn from $\{1, \ldots, n\}$, and the parentpointers are stored in an `array[1..n]` called `p`. Give a PASCAL algorithm for the function `findset (x: integer): integer` when we implement path compression.

QUESTIONS ON HASH TABLES.

Question 22. A sparse array is one in which most entries are zero. Assume that we want to store a sparse $n \times n \times n$ array of real numbers such that the operations `insert` and `delete` are efficient—these operations insert or delete an element in a given position (i, j, k) in the array. Deleting an element thus means making it zero. Explain which space-efficient data structure you would use if you knew that approximately all but n of the n^3 possible elements are zero at any time. We would like to get on the average about $O(1)$ execution times for `insert` and `delete`, just as if we would with an ordinary array.

Question 23. Hash table. Provide an estimate of the expected number of element comparisons needed to insert a new element in a **hash table** (closed hashing) of size $1,000,000$ holding $990,000$ elements. Assume that the hash function and the probing strategy are both "perfect" (i.e., totally random and uniform).

1 10 100 1000 1.99 1.445 0.995 0.99

Question 24. Let x be an integer, representing a 9 decimal digit Canadian Social Insurance Number, to be used for hashing. We wish to construct a hash function $h(x)$ in PASCAL, which extracts in one statement the third and eighth digits. Example: if x is 254076318, then $h(x)$ should be the integer 41, since "4" is the third digit, and "1" is the eighth digit. However, you are only allowed to fill in one assignment statement:

$h(x) =$

Question 25. Assume that we have a closed hash structure with n slots and that n different elements are inserted by means of a perfect hash function and a perfect collision resolution strategy (i.e., if a slot is occupied, try a new but not necessarily different randomly selected slot). Give as accurate an estimate as possible for the expected number of slot verifications needed to insert the elements (clearly, this is at least n.)

In O and Ω notation, what is the expected complexity of the same operation for an open hash structure with n buckets?

Question 26. Give a correct PASCAL definition for an open hashing structure which has B buckets (or chains), and an unspecified number of elements of type "datatype". Since some data structures are compositions of other data structures, it is possible that you will have to define several types.

QUESTIONS ON GRAPHS.

Question 27. The signature sign A of a square $(n \times n)$ matrix A is another matrix of the same dimensions in which an element is zero or one according to whether the corresponding element in A is zero or nonzero. Let A have only positive or zero elements to start with. The object is to study how fast we can compute sign A^k, the k-th power of A (i.e. $A \times A \times \cdots \times A$). Your answers should be in $O(.)$ format. In each case, state in one line the key to your solution.

Time needed to compute sign A^2	
Time needed to compute sign A^n	

Question 28. Minimal spanning tree. A Euclidean minimal spanning tree (EMST) is the minimal spanning tree for a graph with n vertices, each representing a point in space, and with all edges present. The cost associated with each edge is the Euclidean distance between the two vertices involved. If all distances between points are pairwise unequal, is it true or false that each vertex is connected directly to its nearest neighbor via an EMST edge?

Question 29. Consider a complete undirected graph with the following edge length matrix; the vertices are A, B, C, D, E, F.

$$\begin{array}{c c} & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array} & \begin{pmatrix} & 7 & 6 & 1 & 3 & 12 \\ & & 4 & 5 & 8 & 10 \\ & & & 9 & 11 & 15 \\ & & & & 2 & 14 \\ & & & & & 13 \\ & & & & & \end{pmatrix} \end{array}$$

In Kruskal's method for minimal spanning trees, edges are added to the minimal spanning tree in a given order. Show the edges in chronological order:

Edge 1	
Edge 2	
Edge 3	
Edge 4	
Edge 5	

Question 30. In Dijkstra's shortest path algorithm, which data structure would you use in a graph with 10000 nodes and 100000 edges? (Choose one.)

Unordered array of edges	
Standard binary heap of edges	
10-ary heap of edges	
Ordered array of edges	

QUESTIONS ON CODING, REPRESENTATIONS.

Question 31. To convey to someone the shape of a full binary tree with n nodes, how many bits suffice? Give as accurate an answer as you can (thus, don't use the $O(.)$ notation).

Question 32. A hypercube in d-dimensional space is a graph with 2^d vertices, each vertex having d binary-valued coordinates. For $d = 2$, we have the vertices of the unit square, and for $d = 3$, the vertices are the corners of a unit cube. In the graph, every vertex is connected to its neighbors at distance one; for $d = 3$, the edges are the lines you draw when you draw a cube. A partial hypercube is one in which not all edges are present. Suppose the family of all such partial hypercubes in dimension d is to be binary-coded for storage. How many bits are necessary in this coding scheme?

QUESTIONS ON HUFFMAN TREES.

Question 33. How would you merge n (n is large) sorted linked lists of length n each into one gigantic sorted linked list of n^2 elements?

A. Merge all linked lists pairwise (they are all of length $2n$ now), and repeat this procedure until all lists are joined into one.
B. Put all linked lists together into an array of size n^2. Apply buildheap to the array. Empty the heap into a new linked list.
C. Define n traveling pointers, one for each linked list. Find the minimum top element by looking at all top elements. Transfer it to the new linked list. Update the traveling pointer in question, and repeat all of this until all linked lists are exhausted.

QUESTIONS ON TREES, SEARCH TREES.

Question 34. In a binary search tree stored in the usual fashion (shown below), with no extra fields in the record called "cell", we would like to efficiently implement the function `lca (x,y,t)`, where `t` is a tree root, and `x,y` are key values of some nodes. The function returns a pointer to the least common ancestor of `x` and `y` (that is, of all the common ancestors, take the one that is furthest from the root).

Question 35. Assume that the standard dictionary operations `member (x,t)`, `insert (x,t)`, `delete (x,t)`, `empty (t)` and `makenull (t)` are being supplemented with the operation

`locate_by_rank (k,t),`

where x is an element, t is a tree, and k is an integer between 1 and n. `locate_by_rank` is supposed to return the k-th smallest element in the tree. We want to use a standard binary search tree. How would you modify the type definition? Tell me (in prose) how the standard `insert` has to be modified: Next, give a nonrecursive PASCAL implementation of the function given below:

```
function locate_by_rank (k:integer; t:tree):  elementtype;
```

Question 36. A **Fibonacci tree** of height h is the most unbalanced AVL tree of that height; for $h = 0$, it consists of one root. For $h = 1$, we have one root with one left child (for a total of two nodes). Write a short **nonrecursive** PASCAL function for `makenull` (h:integer; VAR t:tree), which creates a Fibonacci tree of height h, where each cell in the tree has fields "left", "right" and "element", and "tree" is a pointer to a cell (as are "left" and "right"). The fields "left" and "right" are thus initialized by `makenull`. The field "element" should not be initialized.

Question 37. Give a very simple nonrecursive algorithm for filling in the "parent" fields in an ordinary binary tree with cells of the type

```
TYPE   cell= RECORD
         left, right, parent:  @cell;
         element:  elementtype
         END;
       tree= @cell;
```

The other fields in the cell are already filled in.

```
VAR t:  tree;
  {other definitions}
BEGIN
  {other stuff}





END
```

Question 38. A binary tree with n nodes numbered 1 through n is to be stored in an array of n records in such a way that in constant time, we can determine whether node u is an ancestor of node v for any nodes u, v. The total storage should be $O(n)$. We are not interested in any other operations. Explain what you would store.

Question 39. Consider a complete k-ary tree stored in the usual fashion in an array. Fill in the following table.

Who is the parent of node i?	
How many leaves are there? (Tricky!)	

Question 40. We are given an ordered sequence $x_1 < x_2 < \cdots < x_n$ in linked list format. These elements have to be put into a complete binary search tree with n nodes, stored in the standard array format for complete trees. Explain how you can do this in time $O(n)$ using just $O(1)$ extra space.

Question 41. We are given a threaded binary tree. Check off all correct statements:

The node pointed to by a thread always is a leaf or a root	
A thread always originates in a leaf	
A threaded tree takes more storage than an unthreaded one	

Question 42. Draw the highest red-black search tree with the 5 nodes named A, B, C, D, E you can think of. In your drawing, show for each node the label and the "rank".

Question 43. A **binary tree** with n nodes numbered 1 through n is to be stored in an array of n records in such a way that in constant time, we can determine whether node u is an ancestor of node v for any nodes u, v. The total storage should be $O(n)$. We are not interested in any other operations. Explain what you would store and why.

Question 44. Consider a **red-black tree** with four full levels of nodes. (Plus, if you wish, a fifth level of external nodes.) The values stored at the nodes are the integers $\{1, \ldots, 15\}$. All the leaves (level 4 nodes) and all the level 2 nodes are red, while the others are black.

A. What is the rank of the root?

B. Draw the tree (showing values and colors of nodes) that results after the element with value 0 is inserted.

Question 45. A **game tree** is given in which each node has k children. It is a two-player game in which the root corresponds to a position you are confronted with. You want to maximize the return. Assume that the k^2 grandchildren of the root correspond to final game positions with values $1, 2, \ldots, k^2$ from left to right. What is the value of the root?

Question 46. The **timed binary search tree** (or **timbin** for short) is an ordinary binary search tree in which besides the standard fields label, left and right, we also store time, where "time" is an integer storing the time of insertion. The first element inserted has time = 1, the second one time = 2, and so forth. Give a PASCAL algorithm for the operation `deleteroot`, in which the root is deleted in such a way that the timbin is restored to what it would be if the root had never been inserted. Do not forget the type definition either. The restoration must be consistent with the "time" stamps stored at the nodes. `deleteroot` should be efficient, i.e., it should take expected time $O(\log n)$ on a random timbin. Hint: when designing your solution, it helps to draw the tree in a time versus value coordinate plane.