UNIVERSITY OF OSLO

Faculty of Mathematics and Natural Sciences

Examination in	**:**	INF 110 — Algorithms and Data Structures
Examination date	**:**	Wednesday, December 3, 2003
Examination hours	**:**	09.00 a.m. - 3.00 p.m.

This examination set consists of 6 pages including the appendix.

Appendix	**:**	One sheet with multiple questions answer array and frequency table for Huffman coding
Permitted aids	**:**	All printed and hand written material

Ensure that your examination set is complete before you attempt to answer it. Detach the appendix sheet, write your number on it, provide the required answers, then staple it together with the rest of your exam answers.

Problem 1 (20 %)

The following set of 10 problems is multiple choice. Each problem has the same weight in terms of percentages, but not necessarily the same difficulty. Please provide your answers on the appendix sheet in the space provided for them.

1. Using the formal definition of O, Ω, and Θ-notation, which of the choices below is true?

   A. $n^3 \in \Theta(2^n)$ B. $n^3 \in \Omega(2^n)$ C. $n^3 \in O(2^n)$ D. A and B E. A and C

2. Which of the functions below grows the fastest for large values of n?

   A. $n^2 2^n (\log n)$ B. $n^2 3^n (\log n)$ C. $n^2 2^n (\log n)^5$ D. $n^2 3^n (\log n)^2$ E. $n^4 2^n$

3. Suppose that we are performing double hashing. Our table size is $m = 7^4 = 2401$. Recall that the first hash function gives the initial location to be probed and the second hash function gives the size of the steps we should jump as we follow the probe sequence. Which of the following values for the second hash function will guarantee that, when doing an insertion, we will find an empty table location unless the entire table is full?

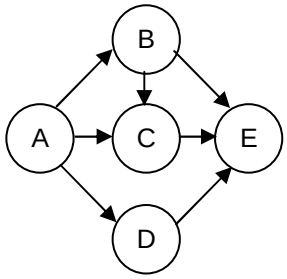
   A. 12 B. 35 C. 49 D. 63 E. 98

4. The bucketsort algorithm was able to sort n integers in the range 0 to n-1 in worst Θ(n) time. Why is this not a contradiction of the Ω(n log n) lower bound we gave

on sorting?

A. That lower bound was for the algorithms that exchange adjacent elements
B. That lower bound was for algorithms that work by comparing keys
C. That lower bound was for NP-sorting
D. That lower bound was for inputs with range 0 to n^2

5. Which of these orders is not a possible order in which Depth First Search could visit the vertices of the directed graph shown below?



A. ADEBC B. ADECB C.ABEDC D. ABECD E. ABCED

6. Which of these orders is not a valid topological order for the same graph?

A. ABCDE B. ABDCE C. ADCBE D. ADBCE

7. Suppose that for some NP-complete problem L, someone was able to show that L was in the class P. What conclusions could you draw?

A. P = NP
B. There is an algorithm to determine in polynomial time whether or not a given graph can be 3-colored.
C. There is no polynomial-time algorithm to decide whether a given boolean expression, built from boolean variables and the operators and, or, and not, is satisfiable.
D. A and B
E. B and C

8. What is the smallest possible height of a decision tree with 200 leaves? Note that the height of a leaf node is 0!

A. 9 B. 8 C. 7 D. 6 E. 5

9. Suppose that we are storing a heap in an array A. This is the type of heap used for quickly finding the maximum element, so values increase as we go from a leaf to the root. Assuming A is as shown below, and we delete the largest element, in what position will the value 20 be after we reheapify? (Use the algorithm discussed in class for deleting from a heap, this is the same as the deletion that occurs during the second phase of heapsort.)

i:	1	2	3	4	5	6	7	8	9	10	11
A[i]:	90	80	70	50	60	30	40	5	10	15	20

A[5] B. A[6] C. A[7] D. A[9] E. A[10]

10. Which of the choices below correctly describes the amount of time used by the following code:

```
for (i = 1; i ≤ n; i++)
    for (j = 1; j ≤n; j = 2 *j)
        for (k = 1; k < n; k = 2 * k)
            x = x + 1;
```

A. Θ (n)
B. Θ (n log n)
C. Θ (n(log n)^2)
D. Θ (n^2)
E. Θ (n^2 log n)

Problem 2 (15 %)

The people of Ognad on planet Zur utilize a very simple alphabet. Monitoring their transmissions we discover the frequency table given in the Appendix.

Problem 2A

Create a Huffman tree to determine efficient binary codes for each character. Find Huffman codes for the letters and fill them into the table from the appendix.

Problem 2B

Encode the following Ognad words KELB DUME.
Decode the following sequence: 100011101010100001110101.

Problem 2C

On average how many bits would be transmitted if 1,000,000 characters were transmitted? What is the minimum number of bits used per character if the above character set were encoded using fixed-length encoding?

Problem 3 (30 %)

The object of the ***Oracle of Connery*** game is to start with any actor or actress who has played one of the leading roles in any movie and connect them to Sean Connery in the smallest number of links possible.

Two people are linked if they've been in a movie together. An actor/actress has Connery number equal to one if he/she has had a leading role in a movie with Sean Connery. For example, Catherine Zeta Jones has a Connery number equal to 1, as she played with him in the movie ***Entrapment***. You might wonder now what Brad Pitt's Connery number is. Under the assumption that they have never played in a movie together (we could not think of one), one may proceed as follows: Brad Pitt was in the ***Legends of the Fall*** with Anthony Hopkins, and Anthony Hopkins was in the film ***A Bridge Too Far*** with Sean Connery. Thus Brad Pitt's Connery number is 2.

Problem 3A

Given a database of movies with their actors (male or female), describe in ***English*** how you could compute the Connery number of all actors.

Carefully state which major data structures and algorithms you would use and how you would use them in order to solve this problem as efficiently as possible. IMPORTANT NOTE: You will need to have chosen the main data structure correctly for the solution in part 3C. Think well and think which design paradigm you will choose!

Assume that each record in the database is a movie name, followed by the number of actors in that movie and then the names of the actors. One record may look like: ***Legends of the Fall, 3, Anthony Hopkins, Brad Pitt, Aidan Quinn***. You may assume that no two actors and no two movies have the same names.

Problem 3B

Let M be the number of movies in the database, A be the number of actors, and S be the average number of actors in each movie. Write an asymptotic expression for the running time of your algorithm, and explain why it is so.

Problem 3C

IMPORTANT NOTE: You will use the main data structure you have chosen in part 3A.

Write the pseudocode for **FindLink (D, A)** where **D** is the reference to the data structure to be processed (i.e., the data structure you decided to use to represent the records in your database), **A** is the actor whose Connery number you are looking for.

Convert the database into an adjacency-list representation of an unordered graph, with a vertex for each actor and for each movie. Read each record and use a hash table to map each name to a vertex. (The vertex should also keep a copy of the name.) Insert edges between a movie vertex and the vertices for actors in the movie. Apply Dijkstra's shortest-path algorithm starting at the vertex for Sean Connery. (Alternatively, since this is an unweighted graph, simply use Breadth First Search.) The Sean Connery number is the shortest path length / 2 for each actor vertex.

The number of vertices is M+A. The number of edges is M*S. Dijkstra runs in O(|E| log |V|), so the answer is O(M S log(M+A)). If you used Breadth First Search instead of Dijkstra, then the run time is O(|E| + |V|), so the answer is O(M + A + MS).

For answer on part C) one could write a pseudocode for Dijkstra, starting at Connery and looking at the shortest path from Connery to A, returning Connery number as integer n. OR one could do the breath first search, with Connery as a root.

Problem 4 (15 %)

Suppose you have been given an implementation of Quicksort which always picks the first element of the sub-array as the pivot. You know that this can lead to very bad performance on sorted or nearly-sorted input.

You are not allowed to modify the Quicksort code or write a new sorting algorithm from scratch. However, you are allowed to process the input before you call Quicksort.

Problem 4A

Describe in English how you can put these pieces together to sort *any* input in average O(n log n) run time.

Randomly reorder the input before calling Quicksort.

Problem 4B

Write the pseudocode for your routine, Bettersort, which takes as input an array A and the length of the array N. Your routine may allocate additional data structure(s) of size N. It should make one call to the Quicksort routine. You have the use of the rand() function which returns a random number.

```
external void Quicksort(int A[], int N); void
Bettersort(int A[], int N){…

int * B;
int i, j, k;
B = new int[N];
k = 0;
for (i=N; i>0; i--){
j = rand() % i;
B[k++] = A[j];
A[j] = A[i-1];
}
for (i=0; i<N; i++) A[i] = B[i];
delete B;
Quicksort( A, N );
}
```

There are many ways to shuffle the input. This way is based on treating A like a "grab bag". A cleverer solution would avoid allocating the array B. Instead, put the shuffled data at the end of the array A. That is, when you pull out the random element and swap in the last element, put that element where the last element used to be.

Problem 5 (20 %)

IMPORTANT NOTE: You will be using a **Java**-like pseudo programming language in this problem, implying that perfect Java-syntax is not expected, but that the language should be recognizably Java, that your logic should be sound and understandable, and that programming details like references etc. should be in a Java-like and understandable syntax.

We will be looking at the B-tree ADT in this problem.

Problem 5A

Write in **Java** the data-structure for a B-tree of arbitrary order **m = 3**. (or a 2-3 tree). You may assume that the values stored in the structure are integer values as usual. Note only the data-structure is required and not the operations.

SOLUTION proposal for 5A

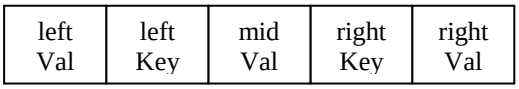
It can be solved in many ways, depending upon the choice of underlying basic structure – i.e., whether single attributes, pointer-chains (lists), arrays etc are used. It also depends upon whether one uniform structure for all types of nodes are used or whether internal and leaf nodes are represented separately.

One possible implementation (among many) of a 2-3 tree using single attributes and single node structure for both node types is given below. Note that it is Java-like pseudo-code and does show details like "public" etc.

```
class Node
{ boolean leaf;   // Tells whether leaf-node or not.
  int leftVal;    // Leftmost value. Used as left key
                  //   in internal node, and as leftmost
                  //   value in leaf node.
  int midVal;     // Middle value. Used only in a leaf
                  //   node.
  int rightVal;   // Rightmost value. Used as right key
                  //   in internal node, and as rightmost
                  //   value in leaf node.
  Node leftNode;  // Referance to leftmost node if this
                  //   is an internal node.
  Node midNode;   // Leftmost value. Used as left key
                  //   is an internal node.
  Node rightNode; // Leftmost value. Used as left key
                  //   is an internal node.
  Node mother;    // It makes it easier to traverse if it
                  //   points back to mother (but is not
                  //   required).
}
```

Alternatively one could use two arrays of three elements, one integer array for the three values leftVal, midVal, rightVal, another for the three references instead of leftNode, midNode, rightNode. This makes managing operations like insert and delete (and hence merge) easier.

Yet another alternative is just using an array that reserves 5 cells for each node (regardless of type), where the block of 6 cells would look like this



The 5-cell block would function as an internal node indexing (pointing to) left, mid and right children and housing two key values (left and right keys).

The 5-cell block would function as a leaf node with up to three values (left, mid and right values), not using the keys as keys but marking (after some convention) of the fact that this is a leaf node and not an internal node.

One may start using from cell 1, and use index value 0 in internal node left, mid and right values as a mark of “null reference” (or “no index”), and -1 (or some unlikely value) as meaning “this is a leaf node) etc.

The student should preferably have noted that array-implementation is efficient but may be a problem if not possible to extend the size dynamically.

A mix of pointers (references) and arrays is also possible. We will use a structure where all the leaves are in one single array.

```
// The array that contains all leave-nodes. It is known to
// the Node class below, and to the “program”.
int Leaf[MAXLEAVES];


// The Node class that represents only internal nodes.
class Node
{ boolean pointsToLeaf; // Tells whether the node points to
                        // a leaf node or not. If not,
                        // it points to another internal
                        // node.
  int indexToLeaf[3];   // Index into the array Leaf
                        // where all the leaf nodes are, if
                        // this does point to a leaf node.
  int key[2];     // If not leaf, its left-key is in
                  // key[0],and right-key is in key[1].
                  // Value = -1 is “no key”.
  Node child[3];  // Its leftmost child is in child[0] etc.
                  // Built-in value “null” is “no child”.
  // Note it is usually recommended that each level is a
  // doubly linked-list giving access to siblings.
}
```

Problem 5B

We will assume a B-tree structure of order **m** = 3 which is also referred to as a 2-3 tree. Assume also that the integer values recorded in the B-tree are ages of people. Write **Java** code to search for the ages within a range **lo** ≤ age ≤ **hi** (where **hi** and **lo** are given) and output each age in a sorted manner. You can choose the sorting order, but you must state explicitly your choice of sorting order. Example: The program would output 11 and 13

(in ascending order) if the search is for $7 \leq age \leq 17$ and if the structure contains only those two ages recorded in that range.

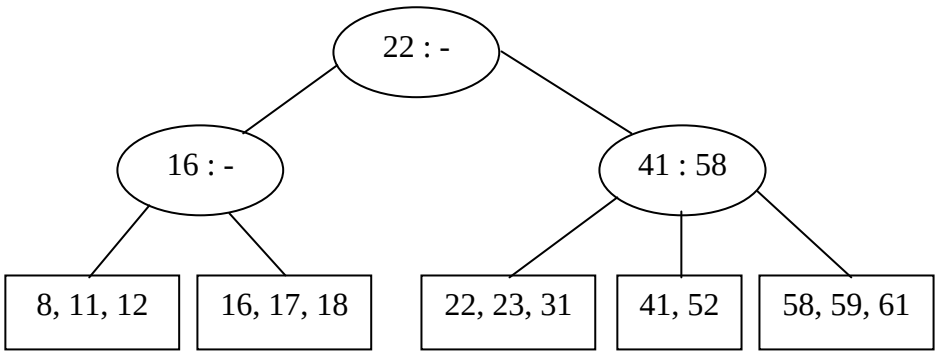
SOLUTION proposal for 5B

The signature is not given, and can be assumed to be

```
void queryAge( Node root, int lo, int hi);
     or
boolean queryAge(Node root, int lo, int hi);
     or
int queryAge(Node root, int lo, int hi);
```

The second one may be expected to return success or failure type of information. This can also be extended with a Unix-like convention to return an integer value indicating success (= 0) or different types of failure as in the third one. The last one may also return number of nodes found.

Assume ascending sorting order and given 2-3 tree with smaller values on the left like this one from the book:



We are assuming a query like “find all people with ages from 10 to 20”, and with the 2-3 tree example above, it should list out 5 values, i.e., 11, 12, 16, 17 and 18. We are going to implement the signature that returns the number of nodes found. With chose structure, the pseudo-code is very simple:

```
int queryAge( Node root, int lo, int hi)
{
  int nodeCount = 0;

  // Find the node that refers to the leaf with the lo
  // value (assuming correct input, and that the value
  // exists in the structure).
  // It would be the one with 8, 11 and 12 in the example.
  int theLoIndex = findLeafIndex (root, lo); // See below.

  // Start rounds to print write out until hi is written.
  // Note that i < MAXLEAVES is just “security measure”.
  // The loop terminates with a break.
  for ( int i = theLoIndex; i < MAXLEAVES; i++)
  {
    if ( Leaf[i] >= lo && Leaf[i] <= hi )
    {
      System.out.ptintln( Leaf[i] );
      nodeCount++;
    }
    else break;
  }
  return nodeCount;
}
```

The other method used, i.e.,

```
  Node findLeafIndex( Node n, int val)
```

is for locating the index to the leaf that contains the value “val”. It is trivial (book/lecture material), but the logic is as follows: Start form the node n, checking keys against “val”, and pick one of three pointers if pointsToLeaf is false and repeat, else return with the index.

Appendix – answer sheet for questions 1 and 2

Problem 1

Please insert the letter that corresponds to the correct answer to questions 1-10 into the table bellow:

1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
C	D	A	B	C	C	A	C	A	C

The correct answer to problem 7 is D. But in the context of the pensum (only Hamiltonian circuit and traveling salesman problems were worked with), A is accepted as correct as well.

Problem 2

You may use a clad sheet, but please draw the final tree in the space bellow (left of the table). In order that as many of you as possible obtain the same answer, always arrange the leaves and the sub-trees of the Huffman tree in increasing order from left to right. If two leaves have the same weight, place as leftmost the one that appears in the table first (from top to bottom). Fill the codes in.

(SEE THE TREE ON THE NEXT PAGE)

Frequency Table		
Character	**Frequency**	**Huffman code**
space _	.13	101
U	.15	110
K	.10	010
B	.10	011
E	.18	111
M	.05	0000
D	.05	0001
R	.10	001
Z	.09	1000
L	.06	1001

KELB DUME code is: 0101111001011000111000001111

Decoding of 100011101010100001110101 is: ZEK_MEK.

Average number of bits transmitted per 1000000 characters is: 3400000.

Minimum number of bits for fixed – length encoding is: 4

There is more then one correct answer – so if you followed the right procedure, but your tree does not look quite the same, it is OK.