NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
EXAMINATION FOR

SEMESTER 2 2008/2009

CS1102S—Data Structures and Algorithms

April 2009 **Time Allowed: 2 Hours**

INSTRUCTIONS TO CANDIDATES

This is an open book examination. Any written and printed material may be used during the examination.

Please observe the following items:

1. Enter your matriculation number here: | | | | | | | | |
2. Enter the answers to the questions in this answer book in the provided spaces.
3. This examination booklet has 15 pages, including this cover page, and contains 8 questions.
4. Answer all questions.
5. The maximal marks achievable is indicated for each question. Overall, the maximal number of marks is 100.

Good Luck !

Do not write below this line

Question	1	2	3	4	5	6	7	8	Total
Maximal Score	21	9	9	7	15	12	5	22	100
Student Score									

Question 1: (21 marks) Assume that a set of N persons are represented by distinct positive integers from 0 to $N-1$.
You are given a boolean matrix (2-dimensional array) childOf that indicates that one person is a child of another person. Example:

```
childOf[7][18] == true
```

if and only if Person 7 is the child of Person 18. In your programs below, you can use childOf and N as global variables.

- (6 marks) Give a function

  ```
  boolean childless(int person)
  ```

  in pseudo-code or Java that returns **true** if and only if the given person has no child.

Answer:

```
boolean childless(int person) {

   boolean answer = true;
   for (int i = 0; i < N; i++) {
       if ( childOf[i][person] ) {
          answer = false;
          break;
       }
   return answer;
}
```

Give the runtime of your algorithm in Θ notation, in terms of N.

Answer: $\Theta(N)$

- (7 marks) Give a function

  ```
  boolean grandChild(int person1, int person2)
  ```

  in pseudo-code or Java that returns **true** if and only if the given person1 is a grandchild of the given person person2.

Answer:

```
boolean grandChild(int person1, int person2) {
   boolean answer = false;
   for (int i = 0; i < N; i++) {
       if (childOf(person1,i)) {
          for (int j = 0; j < N; j++) {
              if (childOf(j,person2)) {
                 answer = true;
                 break;
              }
          }
       }
   }
}
```

Give the runtime of your algorithm in Θ notation, in terms of N.

Answer: $\Theta(N^2)$

- (8 marks) Give an efficient function

  ```
  boolean cousins()
  ```

  in pseudo-code or Java that decides whether there is any pair of distinct persons that have a common grandparent.

Answer:

```
boolean cousins() {
   LinkedList<int>[] grandparents = new LinkedList<int>[N];
   for (int i=0; i<N; i++) {
       for (int j=0; j<N; j++) {
           if (childOf(i,j)) {
              for (int k=0; k<N; k++) {
                  if (childOf(j,k))
                     grandparents[k].add(i);
              }
       }
   }
   boolean answer = false;
   for (int i=0; i<N; i++) {
       for (int j=0; j<N; j++) {
           for (int k : grandparents[i]) {
               if (i != j && k in grandparents[j]) {
                  answer = true;
                  break;
               }
           }
       }
   }
}
```

Give the runtime of your algorithm in Θ notation, in terms of N.

Answer: $\Theta(N^3)$ because every person only has at most 4 grandparents, so the second set of nested loops runs in $\Theta(N^2)$.

Question 2: (9 marks) Assume that a company has one printer, and a secretary who delivers the printed material to the offices that issue printing jobs. The printing will be done every hour during office hours to ensure the secretary can attend to other tasks.

- (2 marks) Suggest a data structure that ensures that the printing jobs issued earlier will be executed earlier. (only name the data structure)

  Answer: Queue

- (7 marks) You can assume that the actual printing is done using an existing function

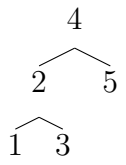
  ```
  public void print(PrintJob p)
  ```

  Using the data structure you are suggesting as a global variable, give the pseudo-code or Java program for the functions that issue the printing jobs, and the function that executes the hourly printing task. You can assume that the accumulated jobs can be printed within an hour.

```
private static  Queue     jobs = new Queue<PrintJob>();


public static void issuePrintJob(PrintJob p) {

   jobs.enqueue(p);

}

public static void hourlyPrinting() {

  while (! jobs.empty) {

     PrintJob job = jobs.dequeue();
     print(job);

  }

}
```

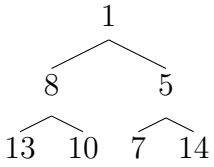
Question 3: (9 marks) A heap is a data structure that provides efficient deletion of the smallest element using the deleteMin operation.
For each of the following complete binary trees, decide whether they are heaps. If the tree is not a heap, give the number of times that function buildHeap has to call compareTo(...) in order to turn the tree into a heap.

- (3 marks)



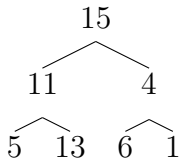
Answer: No. 4 comparisons

- (3 marks)



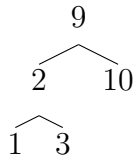
Answer: Yes.

- (3 marks)



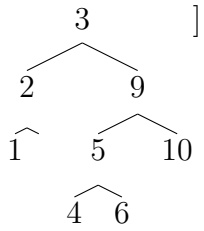
Answer: No. 7 comparisons

Question 4: (7 marks) Consider the following AVL tree.



Draw the AVL tree resulting from inserting first the number 4, then the number 5 and finally the number 6 into the AVL tree!

Answer:



Question 5: (15 marks)

- (2 marks) What is the load factor of a hash table of size 1009 after adding 2018 data records? (no explanation needed)

  Answer: 2

- (3 marks) Is it possible to store 2018 data records in a hash table of size 1009 using separate chaining? Give one complete sentence as explanation.

  Answer: Yes. Separate chaining will result in a linked list of average size of 2 for each hash table entry, and no problems arise.

- (3 marks) Is it possible to store 2018 data records in a hash table of size 1009 using linear probing? Give one complete sentence as explanation.

  Answer: No. If the load factor is larger than 1 (i.e. after inserting 1009 elements), the hash table is full and no more elements can be inserted.

- (7 marks) Assume you have a collection of hundreds of personal data records of Singaporeans living in apartment buildings. The records include a name, a date of birth, street address and unit number. The date of birth is given in the form DDMMYYYY. The name string starts with the surname, followed by the given name. Some names such as "Tan" and "Ng" are very common. Many of the persons in the collection may live in the same building.

  An example record may look like this:

```
{ name         : "Tan Ka Boon, Vincent",
  dateOfBirth  : "24121984",
  address      : "17 Clementi Avenue 4",
  unit         : "07-120"
}
```

  You an assume that the components of a given data record r can be accessed using "." as in r`.address`.

  Give a hash function in pseudo-code or Java that allows storing personal data records of Singaporeans in a hash table of size 1009, such that collisions are minimized.

Answer:

```
public static int hash( Person key, int tableSize) {

   return
   ( String.toInt(key.dateOfBirth) + key.address[0] + key.address[1] )
   %
   tableSize

}
```

Question 6: (12 marks) Assume given an unsorted non-empty array of Comparable objects. Recall that two objects obj1 and obj2 can be compared by calling obj1.compareTo(obj2); the result is negative if obj1 is smaller than obj2, 0 if they are considered equal, and positive if obj1 is larger than obj2.
Consider the following simple sorting algorithm that we shall call *deletion sort*:

> *As long as there are there are elements left in the unsorted array, find the index of the smallest element, copy that element into a helper array, and replace it in the unsorted array by* ***null****. The helper array is filled from left right, starting at index 0, and therefore will be sorted at the end of the process.*

- (9 marks) Give the pseudo-code or Java program of deletion sort by filling the following template. Make sure to avoid applying compareTo to **null**.

Answer:

```
public static <AnyType extends Comparable<? super AnyType>>
void deletionSort(AnyType[] a) {
   AnyType[] helper = new AnyType[a.length];
   helperIndex = 0;
   for (int i=0; i < a.length; i++) {
      int smallest = a[0];
      int smallestIndex = 0;
      for (int j=0; j < a.length) {
         if (if a[j] != null && a[j].compareTo(smallest) < 0) {
            smallest = a[j];
            smallestIndex = j;
         }
      }

      helper[helperIndex++] = smallest;
   }
   for (int i=0; i < a.length; i++) {
      a[i] = helper[i];
   }
}
```

- (3 marks) If the given array a has N elements, how many times does your algorithm execute the compareTo method? Give a direct formula on N, and do not use $O(..)$ or $\Theta(...)$ or $\Omega(...)$ notation.

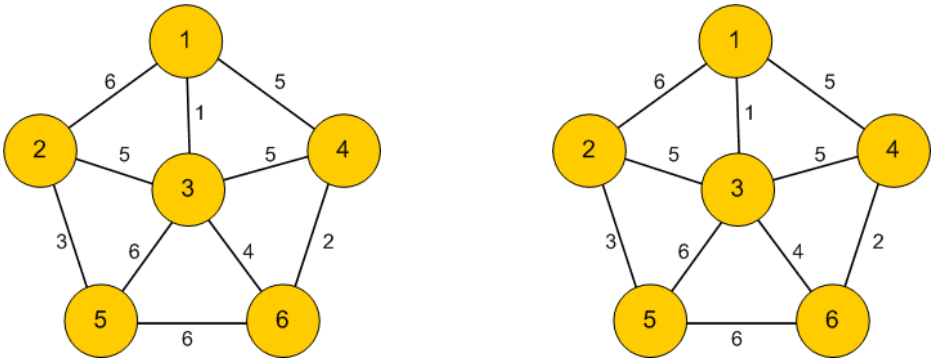
Answer: There will be $N(N-1)/2$ comparisons.

Question 7: (5 marks) In the lectures, methods for sorting large data sets were presented, which use tapes as storage medium. All these external sorting methods were based on mergesort.
Why is mergesort a better choice than quicksort as a basis for external sorting? Give your answer in at most three complete English sentences.

Answer: Mergesort allows for splitting into predictably sizes sections called blocks. This means that the tape space can be predicted, and the algorithm can be controlled precisely how the tapes are moved. Quicksort's sizes of the "blocks" are unpredictable, and therefore the algorithm cannot make effective use of the tapes.

Question 8: (22 marks) A minimum spanning tree of an undirected weighted graph G is a tree formed from edges of G that connects all the vertices of G at lowest total cost.

- (4 marks) Indicate a minimum spanning tree of the undirected weighted graph on the left, by drawing the edges of the tree in the diagram on the right.



- (1 marks) What is the total cost of a minimum spanning tree in this example? (no explanation needed)

Answer: 15

- (7 marks) In the lectures, Prim's algorithm, a greedy method for finding a minimum spanning tree, was presented, using an informal description and examples.

  Give the algorithm in pseudo-code or Java! Make sure that you clearly indicate the input and output and make clear what helper data structures you use (if any).

Answer:

```
Given a graph, G, with edges E of the form (v1, v2) and vertices V

dist : array of distances from the source to each vertex
edges: array indicating, for a given vertex, which vertex in the tree it
       is closest to
i    : loop index
F    : list of finished vertices
U    : list or heap unfinished vertices

/* Initialization: set every distance to INFINITY until we discover a way to
link a vertex to the spanning tree */
for i = 0 to |V| - 1
    dist[i] = INFINITY
    edge[i] = NULL
end

pick a vertex, s, to be the seed for the minimum spanning tree

/* Since no edge is needed to add s to the minimum spanning tree, its distance
from the tree is 0 */
dist[s] = 0


while(F is missing a vertex)
   pick the vertex, v, in U with the shortest edge to the group of vertices in
       the spanning tree add v to F

   /* this loop looks through every neighbor of v and checks to see if that
    * neighbor could reach the minimum spanning tree more cheaply through v
    * than by linking through a previous vertex */
   for each edge of v, (v1, v2)
        if(length(v1, v2) < dist[v2])
            dist[v2] = length(v1, v2)
            edges[v2] = v1
```

```
                possibly update U, depending on implementation
            end if
        end for
    end while
```

- (10 marks) Give a proof of correctness of Prim's algorithm.

Answer: Let P be a connected, weighted graph. At every iteration of Prim's algorithm, an edge must be found that connects a vertex in a subgraph to a vertex outside the subgraph. Since P is connected, there will always be a path to every vertex. The output Y of Prim's algorithm is a tree, because the edge and vertex added to Y are connected. Let Y1 be a minimum spanning tree of P. If Y1=Y then Y is a minimum spanning tree. Otherwise, let e be the first edge added during the construction of Y that is not in Y1, and V be the set of vertices connected by the edges added before e. Then one endpoint of e is in V and the other is not. Since Y1 is a spanning tree of P, there is a path in Y1 joining the two endpoints. As one travels along the path, one must encounter an edge f joining a vertex in V to one that is not in V. Now, at the iteration when e was added to Y, f could also have been added and it would be added instead of e if its weight was less than e. Since f was not added, we conclude that

$$w(f) \geq w(e).$$

Let Y2 be the graph obtained by removing f and adding e from Y1. It is easy to show that Y2 is connected, has the same number of edges as Y1, and the total weights of its edges is not larger than that of Y1, therefore it is also a minimum spanning tree of P and it contains e and all the edges added before it during the construction of V. Repeat the steps above and we will eventually obtain a minimum spanning tree of P that is identical to Y. This shows Y is a minimum spanning tree.

END OF QUESTIONS