CS211 Spring 2005
Prelim 2
April 19, 2005

Solutions

Instructions

Write your name and Cornell netid above. There are 8 questions on 11 numbered pages. Check now that you have all the pages. Write your answers in the boxes provided. Use the back of the pages for workspace. Ambiguous answers will be considered incorrect. The exam is closed book and closed notes. Do not begin until instructed. You have 90 minutes. **Good luck!**

	1	2	3	4	5	6	7	8	Σ
Score	/15	/10	/20	/10	/15	/10	/5	/15	/100
Grader									

1. (15 points) Write a recursive binary search method to find an object `v` in a sorted array `a` of comparables. The procedure should return `true` if object `v` is found in the array, and `false` if it is not found. Be sure to handle special cases.

```
//lo and hi are endpoints of the search interval in the array
public static <T> boolean binarySearch (Comparable<T>[] a,
                                        int lo, int hi, T v) {
   if (hi - lo < 1) return false;
   if (hi - lo == 1) return a[lo].compareTo(v) == 0;
   int mid = (hi + lo)/2;
   if (a[mid].compareTo(v) <= 0) return binarySearch(a, mid, hi, v);
   else return binarySearch(a, lo, mid, v);
}

//search interval is inclusive of lo, exclusive of hi
//method assumes that 0 <= lo <= hi <= a.length



//alternative solution, inclusive of both endpoints
//assumes that 0 <= lo <= hi < a.length
public static <T> boolean binarySearch (Comparable<T>[] a,
                                        int lo, int hi, T v) {
   if (hi - lo < 0) return false;
   if (hi == lo) return a[lo].compareTo(v) == 0;
   int mid = (hi + lo + 1)/2;
   if (a[mid].compareTo(v) <= 0) return binarySearch(a, mid, hi, v);
   else return binarySearch(a, lo, mid - 1, v);
}
```

2. (10 points) One advantage of quicksort is that it sorts in place. Manually perform quicksort on the following array of integers (sort in increasing order), using the first element of each subarray as the pivot. In the spaces provided show the values in the array as quicksort proceeds down each level. Circle the pivots used for each subarray. Hint: your answer should have one pivot circled in the first row, two pivots circled in the second row, etc. You may not need to use all the rows provided.

20	31	24	19	45	56	4	65	5	72	14	99
19	4	5	14	20	31	24	45	56	65	72	99
4	5	14	19	20	24	31	45	56	65	72	99
4	5	14	19	20	24	31	45	56	65	72	99
4	5	14	19	20	24	31	45	56	65	72	99
4	5	14	19	20	24	31	45	56	65	72	99
4	5	14	19	20	24	31	45	56	65	72	99
4	5	14	19	20	24	31	45	56	65	72	99

3. (20 points) Asymptotic Complexity

(a) (5 points) If $f(n) = O(n \log n)$ and $g(n) = O(n \log n)$, prove that $f(n) + g(n) = O(n \log n)$. Argue in terms of witness pairs.

Suppose $f(n) = O(n \log n)$ with witness pair (a_f, n_f) and $g(n) = O(n \log n)$ with witness pair (a_g, n_g). Thus for all $n \geq n_f$, $f(n) \leq a_f n \log n$, and for all $n \geq n_g$, $g(n) \leq a_g n \log n$. Define $a_{f+g} = a_f + a_g$ and $n_{f+g} = \max n_f, n_g$. Then for all $n \geq n_{f+g}$, $f(n) + g(n) \leq a_f n \log n + a_g n \log n = (a_f + a_g) n \log n = a_{f+g} n \log n$, therefore $f(n) + g(n) = O(n \log n)$ with witness pair (a_{f+g}, n_{f+g}).

(b) (5 points) Sort the following in order of increasing asymptotic complexity: $O(n \log n)$, $O(n^2 \log n)$, $O(n(\log n)^2)$, $O(n)$, $O(1)$, $O(n^n)$, $O(n^{1.618})$, $O(1.618^n)$, $O(\log n)$, $O(2^n)$.

Complexity Ordering	Function From The List Above
least complex	$O(1)$
2	$O(\log n)$
3	$O(n)$
4	$O(n \log n)$
5	$O(n(\log n)^2)$
6	$O(n^{1.618})$
7	$O(n^2 \log n)$
8	$O(1.618^n)$
9	$O(2^n)$
most complex	$O(n^n)$

(c) (5 points) What is the asymptotic complexity of the following code fragment? Give justification.

```
int count = 0;
for (int i=0; i<n; i++) {
  for (int j=0; j<n; j++) {
    if ((j % 2) == 0) {
      for (int k=i; k<n; k++)
        count++;
    }
    else {
      for (int l=0; l<j; l++)
        count++;
    }
  }
}
```

$O(n^3)$. The outer two loops run through all n^2 values of i and j between 0 and $n-1$, inclusive. For each i, j, the first inner loop executes $n-i$ times if j is even and the second executes j times if j is odd. In either case the number of executions is at most n. For nested loops, the running time is multiplicative, which gives $O(n^3)$.

This is also a lower bound: for $j \geq n/2$ and $i \leq n/2$, regardless of the parity of j, the inner loop executes at least $n/2$ times. Thus the total number of executions is at least $n(n/2)^2 = n^3/4$.

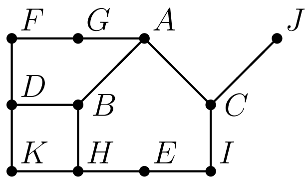
(d) (5 points) Consider the following algorithm to randomly permute an array with all permutations equally likely.

(i) split the array into two roughly equal-size subarrays;
(ii) recursively permute the two subarrays;
(iii) randomly merge the two subarrays.

To randomly merge two arrays, we iteratively take an element from the front of one or the other, each with probability 1/2, until one of the arrays is exhausted, then take the remaining elements. What is the asymptotic complexity of this algorithm? Give justification.

$O(n \log n)$, the same as mergesort. In fact the algorithm is identical to mergesort except for the random selection in (iii). The recurrence governing the running time is $T(n) = 2T(n/2) + cn$. The term $2T(n/2)$ is for the two recursive calls in (ii) and the term cn is for the merge in (iii). The merge is linear because we spend a constant amount of time for each element. The solution of this recurrence is $cn \log n$.

4. (10 points) List the sequence of nodes visited by depth-first and breadth-first traversals of the following graph starting at node A. When there is an arbitrary choice to make in either traversal, expand nodes in alphabetical order.



(a) (5 points) Depth-First

A	B	D	F	G	K	H	E	I	C	J

(b) (5 points) Breadth-First

A	B	C	G	D	H	I	J	F	K	E

5. (15 points) True or false?

(a) $\boxed{\text{T}}$ F Quicksort has worse asymptotic complexity than mergesort.

(b) $\boxed{\text{T}}$ F Binary search is $O(\log n)$.

(c) T $\boxed{\text{F}}$ Linear search in an unsorted array is $O(n)$, but linear search in a sorted array is $O(\log n)$.

(d) $\boxed{\text{T}}$ F Linear search in a sorted linked list is $O(n)$.

(e) $\boxed{\text{T}}$ F If you are only going to look up one value in an array, asymptotic complexity favors doing linear search on the unsorted array over sorting the array and then doing binary search.

(f) $\boxed{\text{T}}$ F If all arc weights are unique, the minimum spanning tree of a graph is unique.

(g) $\boxed{\text{T}}$ F Binary search in an array requires that the array be sorted.

(h) T $\boxed{\text{F}}$ Insertion into an ordered list can be done in $O(\log n)$ time.

(i) $\boxed{\text{T}}$ F A good hash function is one which tends to distribute elements uniformly throughout the hash table.

(j) $\boxed{\text{T}}$ F In practice, with a good hash function and non-pathological data, objects can be found in $O(1)$ time if the hash table is large enough.

(k) T $\boxed{\text{F}}$ If a piece of code has asymptotic complexity $O(g(n))$, then at least $g(n)$ operations will be executed whenever the code is run with parameter n.

(l) T $\boxed{\text{F}}$ Given good implementations for different algorithms for some process such as sorting, searching, or finding a minimum spanning tree, you should always choose the algorithm with the better asymptotic complexity.

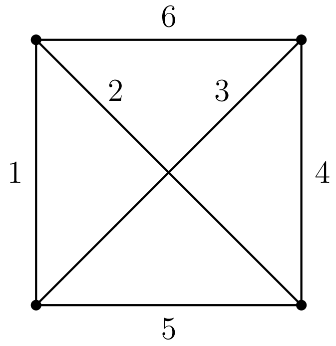
(m) T $\boxed{\text{F}}$ It is not possible for the depth-first and breadth-first traversal of a graph to visit nodes in the same sequence if the graph contains more than two nodes.

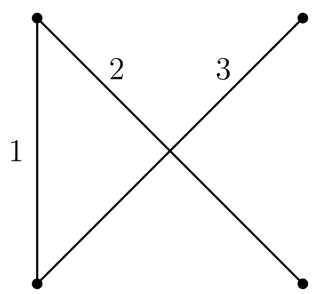
(n) $\boxed{\text{T}}$ F The maximum number of nodes in a binary tree of height H ($H = 0$ for leaf nodes) is $2^{H+1} - 1$.

(o) $\boxed{\text{T}}$ F In a complete binary tree, only leaf nodes have no children.

6. (10 points)

(a) (5 points) Draw the minimum spanning tree for the following graph:





(b) (2 points) Is this graph planar? $\boxed{\text{yes}}$

(c) (3 points) What is the minimum number of colors needed to color this graph? $\boxed{4}$

7. (5 points) Enter the items A-K (in order) into the hash table given the hash values specified for each item.

```
hash(A) = 3
hash(B) = 1
hash(C) = 8
hash(D) = 1
hash(E) = 4
hash(F) = 1
hash(G) = 8
hash(H) = 7
hash(I) = 3
hash(J) = 8
hash(K) = 3
```

Hash Line	1st	2nd	3rd	4th	5th	6th	7th
0							
1	B	D	F				
2							
3	A	I	K				
4	E						
5							
6							
7	H						
8	C	G	J				
9							

8. (15 points) Recall that the `Enumeration` interface is an older form of `Iterator`. An object of type `Enumeration<V>` has methods

```
boolean hasMoreElements()
V nextElement()
```

that correspond to the methods boolean `hasNext()` and `V next()`, respectively, of `Iterator<V>`.

Each of the classes `java.util.Hashtable` and `java.util.Vector` has a method `elements()` that returns an `Enumeration` of its data elements. The `Enumeration` of a `Vector v` enumerates the elements in the order in which they occur in the underlying array. For `Hashtable`, the elements are returned in no particular order.

On the next page, define a class `OrderedHashtable` with methods `put`, `get`, `containsKey`, `size`, and `elements` that are asymptotically no less efficient than the corresponding methods of `java.util.Hashtable`, but for which the `Enumeration` is guaranteed to enumerate the elements in the same order in which they were inserted into the `OrderedHashtable`. To add a new element `x` to the end of a `Vector` `v`, use `v.add(x)`. Assume no duplicate elements are ever inserted.

Hint. Store the data in a `Vector`, and use the `Hashtable` to store indices into the `Vector`.

(write answer on next page)

```
import java.util.Hashtable;
import java.util.Vector;
import java.util.Enumeration;

class OrderedHashtable<K,V> {
   private Hashtable<K,Integer> indices = new Hashtable<K,Integer>();
   private Vector<V> data = new Vector<V>();

   public void put(K key, V value) {
      if (containsKey(key)) { //not necessary, by assumption
         data.setElementAt(value,indices.get(key));
      } else {
         indices.put(key, data.size());
         data.add(value);
      }
   }

   public V get(K key) {
      if (!containsKey(key)) return null;
      return data.get(indices.get(key));
   }

   public boolean containsKey(K key) {
      return indices.containsKey(key);
   }

   public int size() { return data.size(); }

   public Enumeration<V> elements() { return data.elements(); }
}
```

End of Exam