CPSC 221: Algorithms and Data Structures
Assignment #2, due Friday, 2014 June 20 at 17:00 (5pm) PST

Submission Instructions

Type or write your assignment on clean sheets of paper with question numbers prominently labeled. Answers that are difficult to read or locate may lose marks. We recommend working problems on a draft copy then writing a separate final copy to submit.

Each submission should include the names and student IDs of the authors at the top of each page. (You are strongly encouraged to work in pairs, but may not work in groups of three or more. Each pair submits a single copy of the assignment.) On your first page, also sign the statement "I have read and complied with the CPSC 221 2014S1 academic conduct policy as posted on the CPSC 221 course website." (See: `http://www.ugrad.cs.ubc.ca/~cs221/2014S1/syllabus.shtml#conduct`.) In keeping with the policy, you should also acknowledge on your first page any collaborators or resources that helped you with the assignment. Finally, staple your submissions pages together! We are not responsible for lost pages from unstapled submissions.

Submit your assignment to Box 31, in room ICCS X235. Late submissions are not accepted.

Questions

[8] 1. Hash Tables.

(a) Consider a hash table of size 9 that uses chaining. Its hash values are modded by the table size ($h(n) = n \mod 9$), the linked-list chains are unsorted, and items are inserted to the front of the chain. Observe the following operations: `insert(10)`, `insert(30)`, `insert(35)`, `insert(54)`, `insert(76)`, `insert(81)`, `delete(54)`, `delete(10)`, `insert(91)`.

Draw the hash table after every insertion that collides, every deletion, and the final state of the table. Do NOT simply draw the table after every operation.

(b) Consider a hash table of size 11 that uses open addressing with double hashing. The first and second hash functions are

$$h_1(n) = (n + 3) \mod 11$$
$$h_2(n) = 5 - (n \mod 5)$$

Observe the following operations: `insert(10)`, `insert (25)`, `insert(40)`, `insert(41)`, `insert(50)`, `delete(25)`, `delete(10)`, `insert (73)`, `insert(82)`.

Draw the hash table after every insertion that collides, every deletion, and the final state of the table. Do NOT simply draw the table after every operation.

(c) Consider the final state of the hash table in (b). What would happen if we ran the operation `insert(43)`?

[13] 2. AVL Trees.

Let $N(h)$ be the smallest number of nodes in an AVL tree of height h. For example, $N(0) = 1$ and $N(1) = 2$. Prove that $N(h) = F(h + 3) - 1$ where $F(i)$ is the ith Fibonnacci number ($F(0) = 0$, $F(1) = 1$, and $F(i) = F(i - 1) + F(i - 2)$). To do this, you need to come up with a recurrence relation that defines $N(h)$ and argue why it is correct. Then you need to prove, by induction, that $N(h) = F(h + 3) - 1$.

[15] 3. Mysterious Dictionaries.

Dictionaries can be implemented in many different ways. Imagine you are given three dictionaries, each implemented with a "mystery" data structure. You know that the three possible underlying data structures are the following:

(a) Resizing hash table with linear probing

(b) Resizing hash table with quadratic probing

(c) Unsorted linked list

For each of the three dictionaries, explain what strategy you would use to determine the corresponding data structure. You may assume the following:

- `find(key)`, `insert(key)`, and `delete(key)` operations may all be used.
- You have a tool that tells you how long each operation takes to complete
- All the keys are unique; inserting/removing a key already present in the dictionary will have no effect on its contents.
- The unsorted linked list adds to the end of the list, and moves most recently "found" nodes to the front of the list.

[6] 4. Priority Queues.

The queue ADT is often described as FIFO (first in, first out). The stack ADT is often described as LIFO (last in, first out). One of the following is a *good* description of the priority queue ADT. Indicate which one it is and why. One of the others almost always describes priority queues. Identify it amd give two reasons why it's not the best description.

- Last In Last Out
- Only Best Out
- Never Worst Out

[15] 5. Sorting.

(a) *Only Sort of Helpful?*

A common mistake made by those new to sorting is to unnecessarily sort all data. Although many operations can be performed faster on sorted data than unsorted data, this is not always the case. State the

- time complexity on unsorted data,
- time complexity on sorted data, and
- if the time complexities differ, provide a brief (1-2 sentences) explanation why

for the following scenarios:

i. Determining if a given value is larger than the largest value in a singly linked list of integers.
ii. Determining if the current index contains the smallest value in an array of integers.
iii. Determining whether or not a value already exists within a given array of integers.
iv. Determining the depth of a binary tree.
v. Computing the average of a set of integers.
vi. Finding the median of a set of integers.

Be sure to state any assumptions you have made.

(b) *Context.*

For each of the following scenarios, state which sorting algorithm you would use and why. You may choose from one of: selection sort, insertion sort, merge sort, or quick sort. You may only use an algorithm once. Answers without justification will receive no marks.

i. You're working for a telemarketing company that keeps hundreds of thousands of phone numbers. You need to write a function that can take a list of phone numbers and sort them by area code. You have no idea how large this list will be, but you do know that the salespeople who put them together don't do it in any particular order.

ii. For the final question at your technical interview at Google, youre asked to write a sorting algorithm for an array of integers. Youre told that the size of the array will be no more than 30, and you have 5 minutes until your interview time is up.

iii. NASA needs you to write a very quick sorting algorithm that will have to deal with a large amount of information, but run on a microchip with a very small amount of main memory.

[15] 6. Graphs.

(a) Draw the undirected graph whose adjacency matrix is:

	A	B	C	D	E
A	0	1	1	1	1
B		0	1	0	0
C			0	1	0
D				0	1
E					0

(b) Draw the undirected graph whose adjacency list is:

A	B, E, F
B	A, C, F
C	B, D, F
D	C, E, F
E	A, D, F
F	A, B, C, D, E

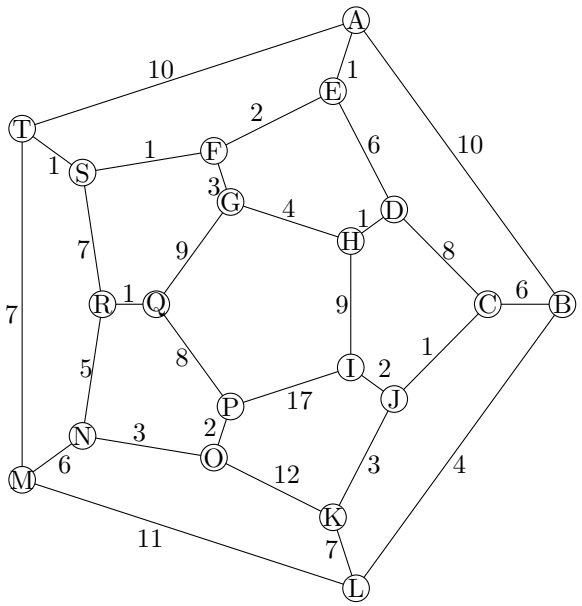
(c) Perform Depth-first search, Breadth-first search, Dijkstra's algorithm, and Kruskal's algorithm on the following undirected, weighted graph. Use A as the source vertex for the first tree and, when there is a choice for which node to pick next, always chose the one that comes first in the alphabet. Show the resulting

i. DFS-tree

ii. BFS-tree

iii. shortest-path tree

iv. minimum spanning tree.



[10] 7. Combinatorics and Logic.

(a) A coin is tossed ten times. Each toss results in heads or tails. Give the following probablities and justify your answers.

i. What is the total number of outcomes?

ii. How many have exactly five heads?

iii. How many have at least eight heads?

iv. How many have at most one head?

(b) How many different strings can you form from the letters: "ALFALFA"? Explain.

(c) For how many numbers in 1 to 999 is the sum of their digits 7? Explain.

(d) Find the least number of cables to directly connect 8 computers to 4 printers so that any 4 computers can directly access 4 different printers. Prove that your answer is correct (i.e., show how to do it with that number and why it can't be done with fewer).

[18] 8. Proof.

Suppose that we have a *binary search tree* defined with a `Node` structure identical to the one from Lab 5 (still with integer keys – here we will use `int` instead of `KType`, however note that Lab 5 uses a `typedef` to make these equivalent). Furthermore, suppose we are given a `size(Node *root)` function that calculates the number of nodes in the tree rooted at the node `root`.

Then, consider the following function that uses the order property of binary search trees to extract the k-th smallest value stored in the tree rooted at the node `root`. We include the precondition that the tree at `root` has size greater than or equal to `k`, ensuring that the problem is well defined and there always is a k-th smallest value.

```
int find_kth_smallest_value(Node *root, int k) {

   Node *target = root;
   int p = k;
   int size_left_branch;

   while(true)
   {
      size_left_branch = size(target->left);

      if (size_left_branch == p-1)
         return target->value;
      else if (size_left_branch > p-1)
         target = target->left;
      else {
         target = target->right;
         p -= size_left_branch+1;
      }
   }
}
```

Take some time to make sure you understand what the function is doing before answering the following questions, in which you prove that the function works as you expect it to.

(a) Come up with and prove a loop invariant involving `k`, `p`, `root`, and `target`. (Hint: you may want to draw out a large BST and step through the function. As you do so, keep track of `p` and `target`).

i. State the loop invariant;

ii. Verify that it is true when the loop begins to execute; and

iii. Show that if it holds at the beginning of the loop, it must also hold at the end.

(b) Explain why the loop must terminate.

(c) Use parts (a) and (b) to construct your argument as to why the function `find_kth_smallest_value(Node *root, int k)` returns the k-th smallest value in the tree at `root`.