NAME (print): _____

Honor Acknowledgment (signature): _____

DO NOT SPEND MORE THAN 10 OR SO MINUTES ON ANY OF THE OTHER QUESTIONS! If you don't see the solution to a problem right away, move on to another problem and come back to it later.

Before starting, make sure your test contains 10 pages.

If you think there is a syntax error, then ask. You may assume any include statements are provided.

	value	grade
Problem 1	12 pts.	
Problem 2	12 pts.	
Problem 3	16 pts.	
Problem 4	8 pts.	
Problem 5	6 pts.	
Problem 6	4 pts.	
Problem 7	12 pts.	
TOTAL:	70 pts.	

Good Luck!

PROBLEM 1 : (*And the winner is ...*(12 points))

Assume you are considering the implementation of a priority queue that will always give you the smallest value. Three different means of implementation are being contemplated.

1. Unsorted Vector
2. Sorted Linked List
3. Min Heap

Evaluate the performance of each of these three implementations for the following two operations:

A. Delete Min (Get min item and remove from system).

B. Insert (Add a new item to the system.)

Use Big O notation in describing the expected performance. Label clearly (with 1.A, 1.B, 2.A, 2.B, 3.A, 3.B), the combination you are talking about.

PROBLEM 2 : (*Reeling in the big one* (12 points))

Part A (3 points) Explain why removing the largest (or smallest) node from a binary search tree is always very simple.

Part B (9 points) Complete the function `BSTRemoveMax` whose header is shown below. It should find and remove the largest node from a binary search tree. That node should not be deleted, but rather a pointer to it should be returned. The resulting tree should be a proper binary search tree with one less node.

```
template <class type>
TNode<type> * BSTRemoveMax(TNode<type> * t)
// pre: t is a binary search tree, possibly empty
// post: Largest node removed from t.
//       Returns a pointer to the largest node in the tree, but
//       returns null if the tree is empty.
{
```

PROBLEM 3 : (*Pass the hash* (16 points))

Part A (8 points) You are inserting the strings below into a hash table. Assume the hash value for each string computes as shown

string	hash value
"pea"	12
"leek"	26
"bean"	20
"yam"	37
"corn"	33
"kale"	10
"okra"	31

If the values are inserted in this order (i.e., "pea" first and "okra" last) into a hash table with 11 entries, fill out the table assuming that collisions are resolved by chaining. Sketch the results of chaining.

0	1	2	3	4	5	6	7	8	9	10

Repeat the process with the table below, but assume collisions are resolved by linear probing.

0	1	2	3	4	5	6	7	8	9	10

Part B (4 points) Deletion for systems using linear probing can cause problems. Give an example of a `delete` followed by a `search` that illustrates this for the example show immediately above. What is the error that occurs? **Be brief. Only a sentence or two are required.**

Part C (4 points) Assume that we are using hashing with linear probing and our vector has 1000 entries. What is the absolute maximum number of entries the system can contain without changing the vector?

Explain why such an absolute maximum doesn't make sense when discussing chained hashing.

PROBLEM 4 : (*Sort of ...* (8 points))

Part A (4 points) A radix sort could be described as a series of bucket sorts. What determines the number of bucket sorts (or passes) required? Make sure that repeated keys are considered in your discussion. (I.e., sorting N items does not require N unique keys.)

Part B (4 points) Name the fastest, comparison based, stable sort. Give its worst case and average big O performance. Note any shortcomings this sort has.

PROBLEM 5 : (*Don't lose your balance.*(6 points))

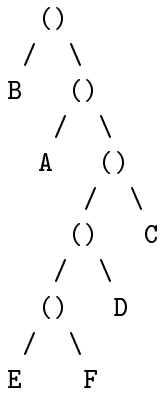
To implement AVL binary search trees, you use a set of operations to re-balance the tree after an insertion has made it unbalanced.

Part A (3 points) What is the criterion used to decide if the tree is balanced or not?

Part B (3 points) For AVL trees, after an un-balancing insertion, one or at most two "rotations" are required to re-balance the binary search tree. What is the big O for each of these rotations.

PROBLEM 6 : (*Being in good shape.*(4 points))

As shown in lecture, we can use binary trees to represent varying length codes. Usually we use the path from the root to represent the code with a left branch standing for a 0 and a right branch standing for a 1. For example, the following represents a Huffman code for the symbols A thru F.



We can also use trees to represent uniform length codes. ASCII is one example of such a code. Characterize the shape of the binary tree representing a uniform length code. (A single phrase of few words is sufficient for an answer.)

PROBLEM 7 : (*A sorted affair ...* (12 points))
Part A (8 points)

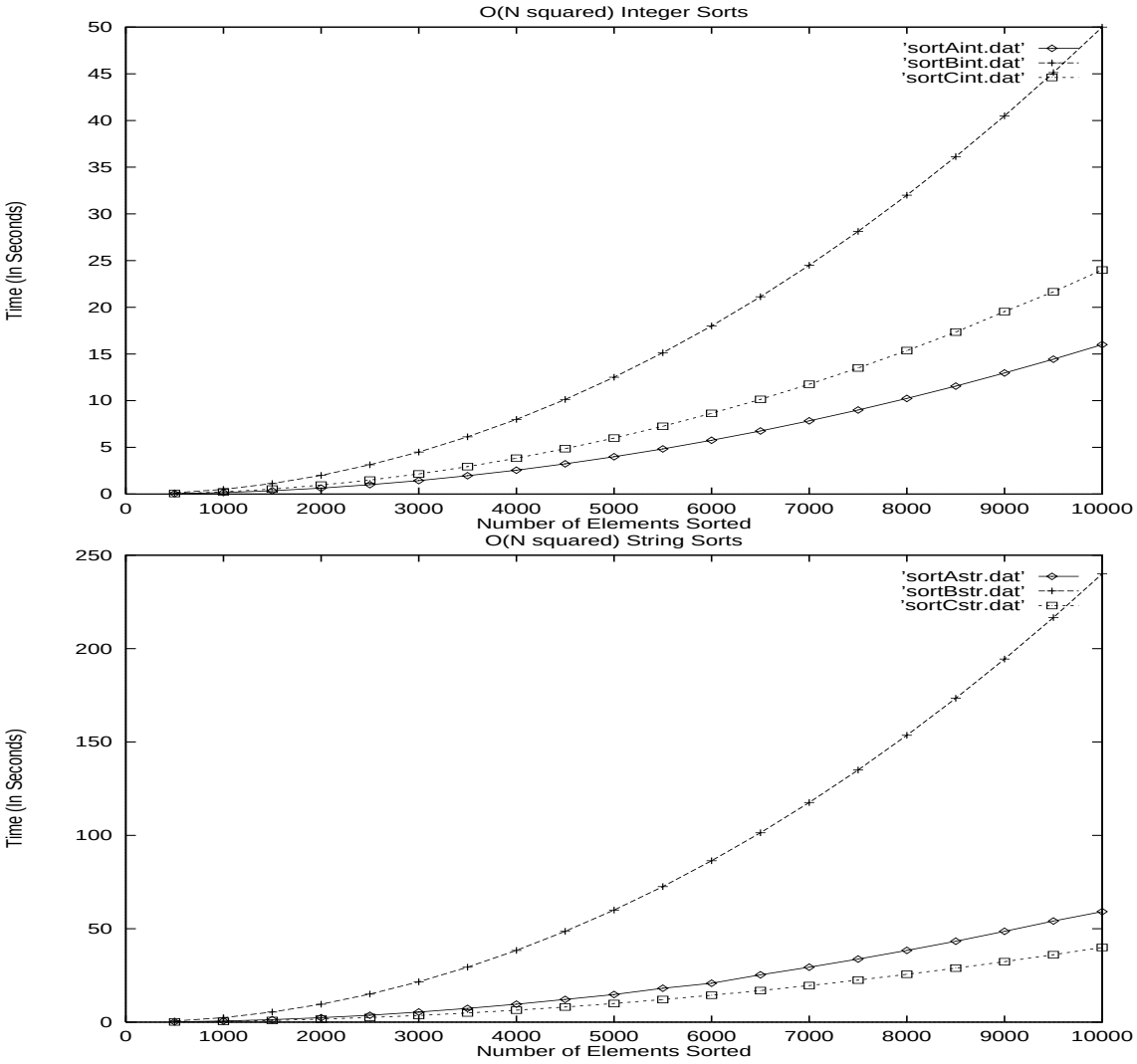


Figure 1: Graphs for $O(N^2)$ sorts.

A person runs sortall for $O(N^2)$ sorts, but forgets which sort is which. From the graphs in Figure 1, determine which sort Sorts A, B, and C are respectively. (The top graph shows data for sorting integers, the bottom for sorting strings.) Explain.

Part B (4 points)

Table 1: Timings for Sorts of integers

Elements	Sort A	Sort B	Sort C
500	0.04	0.13	0.06
1000	0.16	0.5	0.24
1500	0.36	1.13	0.54
2000	0.64	2.00	0.96
2500	1.00	3.13	1.50
3000	1.44	4.50	2.16
3500	1.96	6.13	2.94
4000	2.56	8.00	3.84
4500	3.24	10.13	4.86
5000	4.00	12.50	6.00
5500	4.84	15.13	7.26
6000	5.76	18.00	8.64
6500	6.76	21.13	10.14
7000	7.84	24.50	11.76
7500	9.00	28.13	13.50
8000	10.24	32.00	15.36
8500	11.56	36.13	17.34
9000	12.96	40.50	19.54
9500	14.44	45.13	21.66
10000	16.00	50.00	24.00

The timings of the 3 sorts for integers are given in Table 1. From this table, estimate how long each sort will take to sort 30000 integers. Justify.