CPS 100, Ramm **Hour Exam #2 (3/31/99)** **Spring, 1999**

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	13 pts.	
Problem 2	14 pts.	
Problem 3	12 pts.	
Problem 4	9 pts.	
Problem 5	6 pts.	
Problem 6	16 pts.	
TOTAL:	70 pts.	

Good Luck!

PROBLEM 1 : (*Various ...* (13 points))

Part A (3 points) When a priority queue is implemented using a min heap, the heap is stored as an array. If a node is found at index value k, where would the parent of that node be found? (Assume that the root is located at index 1, not index 0)

Part B (3 points) For a large min heap, why is an *Insert* operation typically much faster than a *RemoveMin* operation?

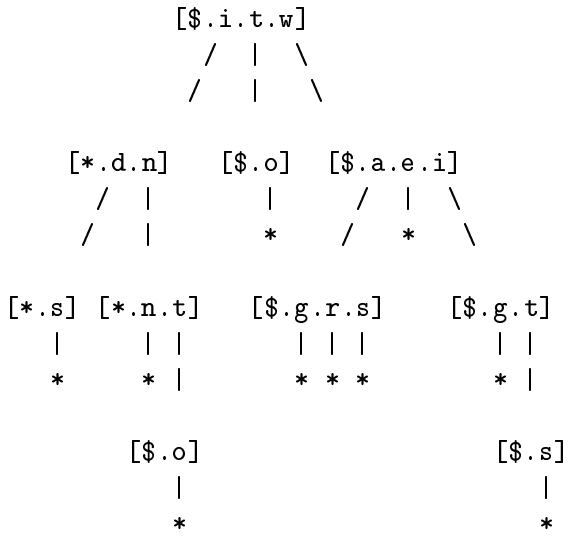
Part C (3 points) For a heap, why is the **worst** case for *Insert* or *RemoveMin O(log N)*?

Part D (4 points) What is the output from the following code fragment employing the shift operators?

```
int k = 57, m = 57;
m = k ^ m;
cout << m << endl;
k = k >> 2;
cout << k << endl;
```

PROBLEM 2 : (*trie and trie again ...* (14 points))

Part A (4 points) The figure below represents that compacted notation we had for representing alphabet tries. (Remember that each node represent in brackets below is really a 27 element vector of pointers.) List all of the words that the trie below contains in its present state.



Part B (2 point) The trie shown above currently does not contain the word **wit**. Describe the change needed to have it include that word.

Part C (2 point) The trie shown above currently contains the word **in**. Describe the changes needed to remove that word.

Part D (4 points) The justification for a trie of this type was that it provided extremely rapid access to determine the presence or absence of a word. Explain why it is so fast and give the Big O to represent the time cost of accessing (determining the presence or absence of) a particular word.

Part E (2 points) What is the biggest drawback of the implementation of alphabet tries as described?

PROBLEM 3 : (*Pass the hash* (12 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.**

PROBLEM 4 : (*Options and Choices* (9 points))

Assume you are contracted to implement a database for a client.

For each set of requirements listed below, choose the implementation which you would use and give a brief (one sentence) justification.

Implementations based on:

a: An unsorted vector.

b: A hash table

c: An balanced binary search tree

d: A sorted vector

e: An unsorted linked list

f: A sorted linked list

Requirements:

A: Frequent inserts; frequent queries; frequent deletes; occasional complete dumps of the system in sorted order.

B: Very frequent inserts; few or no deletes; few queries, most likely query is to confirm the most recent entry. (Essentially an archiving, back-up, or logging system).

C: Moderate number of inserts, very few deletes, very frequent queries

PROBLEM 5 : (*Hyacinth Bouquet* (6 points))

Part A (3 points) A radix sort could be described series of bucket sorts. What determines the number of bucket sorts or passes required?

Part B (3 points) A radix sort could be described series of bucket The radix sort is often described as order N. Is that at odds with the answer to question A? Explain.

PROBLEM 6 : (*A sorted affair ...* (16 points))
Part A (8 points)

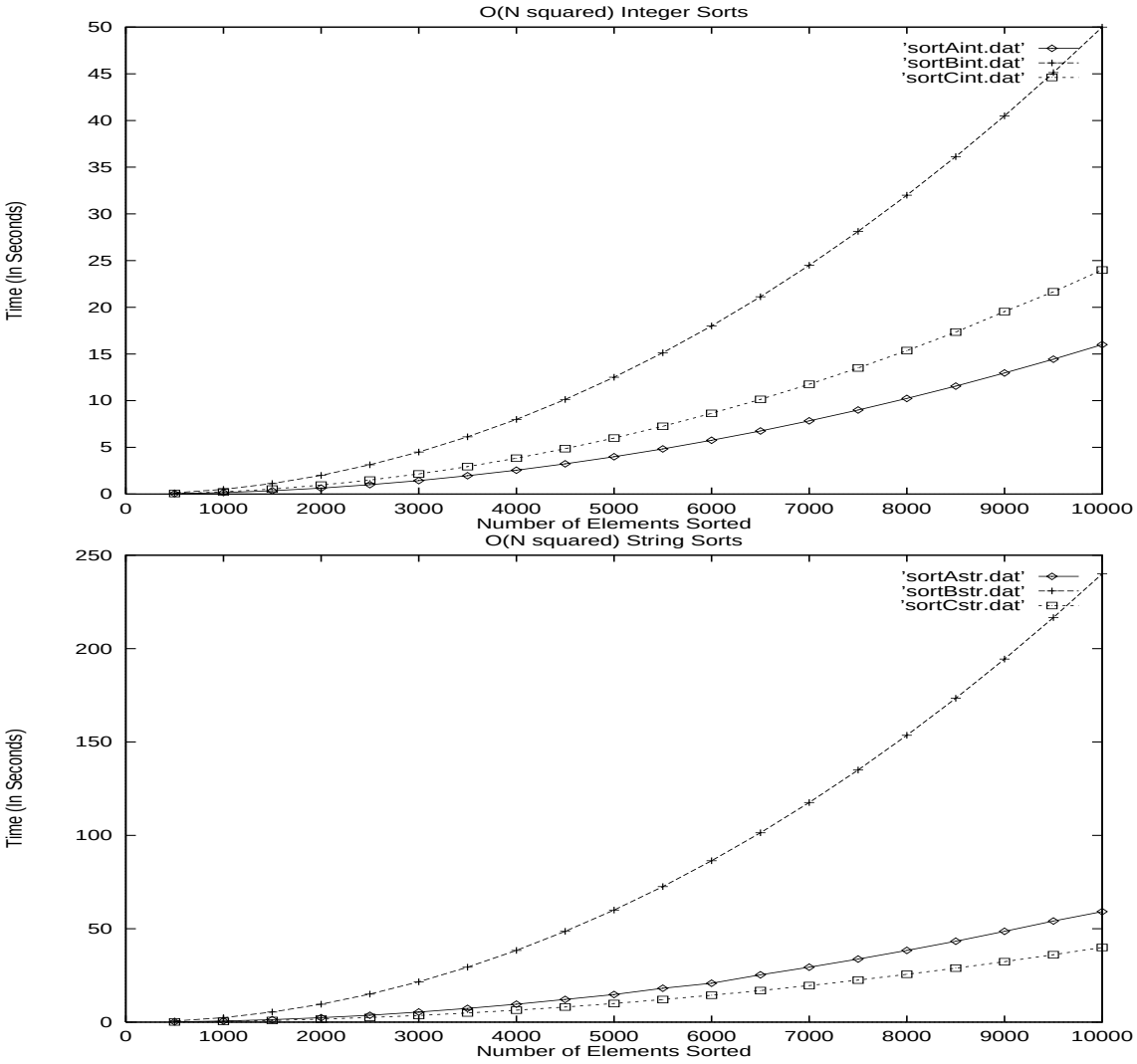


Figure 1: Graphs for $O(N^2)$ sorts.

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. 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 20000 integers. Justify.

Part C (4 points)

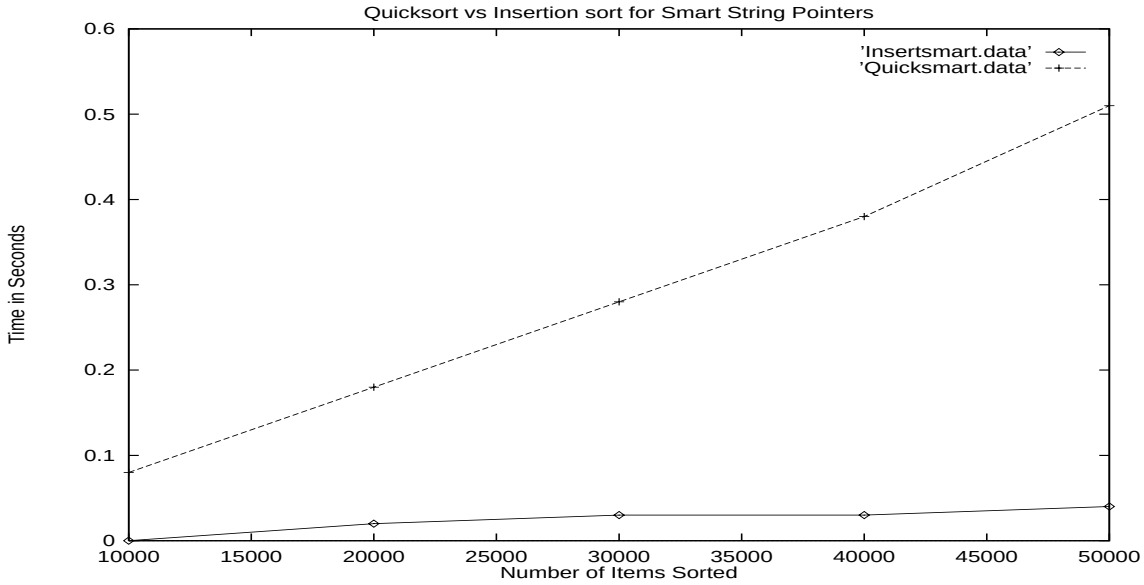


Figure 2: Comparison of Quicksort and Insertion sort for smart string pointers

Figure 2 shows a comparison of sortall for smart string pointers for Quicksort and Insertion sort. From the graph, we see that Insertion sort outperformed Quicksort. Explay why. Also, explain if there are any problems with the testing procedure. (The code used to run sortall is attached.)