CS 61B, Fall 1994
Final
Professor Yelick

Problem 1. *(15 points)* For each of the following, fill in the blank with the appropriate phrase that corresponds to the description. All correct answers have between one and three words. Each is worth 1 point.

1. Write your name and login at the top of every page in this exam.
2. A small piece of code that "fakes" a procedure and is used to test higher level procedures.
3. This type of C++ function is used to specify an interface to be used by all derived classes. The **iterator** class from Budd contains these. (The correct answer is 3 words.)
4. This representation of an undirected graph is best if we are interested in finding whether or not a vertex is isolated, .e. is not connected to any otehr vertices.
5. What function, besides quicksort, uses a **partition** function?
6. You are asked to sort 10,000 integers that are evenly distributed between 0 and 1000. Which sorting algorithm is best?
7. This kind of matrix is typically represented using same data structures as those used in an edge list representation of a graph.
8. This representation of a queue is best for large queues in a system in which **new** and **delete** are very expensive.
9. What is minimum number of keys for a non-root node in a B-tree of order k, where k is even?
10. Give another name for an undirected graph in which there is a single path from every vertex to every other vertex.
11. A storage allocator may not be able to find a large enough block for allocation, even though there is plenty of unused storage. What causes this?
12. Scheme does not have a **new** and **delete** operations (or anything like them) because it does this.
13. This method of reclaiming memory cannot reclaim cycles.
14. This strategy has proven superior when searching a free list for a sufficiently large block of memory.
15. This method of memory allocation uses multiple free lists.

Problem 2. *(20 points)*

a. *(6 points)* The appendix contains the **tree** code from lab 9, modified to use integer values at the nodes rather than strings. Draw a picture of the data structure for **t1** and **t2** after the following code is executed. Designate which objects are on the stack and which are on the free store.

```
Tree* t1 = new Tree(4);
Tree* t2 = new Tree(2, t1);
t2 = new Tree(3, t1);
t2 = new Tree(4, t2);
```

b. *(4 points)* What would the above data structure look like after executing: **delete t2**?

c. *(10 points)* In lab 9, we used trees to represent amoeba family trees. We will call the number of elements in a given level of such a tree the *generation size* of that level. The generation of the root is 1; if it has two amoeba children, the generation size of their level is 2; if both of them have 3 children, the generation size of their level is 6. Write a function **maxGeneration** that, given a pointer to a **tree**, returns the size of the largest generation in the tree. You may use any auxiliary data structures or functions from the books. Give a brief description of how your algorithm works here. Write your solution on the following page.

Problem 3 *(15 points)*

Assume the government has a database on all of us, keyed by a 9 digit number (e.g. social security number). They've been collecting data since birth, and have roughly 10,000 bytes per person. Your job is to implement a system for Berkeley that stores a subset of this database for its 20,000 undergraduate students. Your computer has 1 megabyte (2^10 or approximately 1,000,000 bytes) of memory and one gigabyte (2^20 or approximately 1,000,000,000 bytes) of disk space. Assume that disk accesses are very slow and that it takes 8 bytes to store a pointer to something on the disk.

a. *(5 points)* How would you store this data to make the **lookup-by-SS** operation as fast as possible? (The lookup-by-SS operation takes a single 9 digit social security number and returns the data for that key.) Draw a picture of your data structure and designate whether each piece is in memory or on the disk.

b. *(5 points)* Assume you now want to add a new operation, **lookup-range-SS** which takes 2 social security numbers and erturns all the records in the range between them. Assume that **lookup-range-SS** will be more common that **lookup-by-SS**. Descrive how this operation would work using your data structure from part a or describe a new data structure.

c. *(5 points)* Assume that there is a now second key, the student id, which is a 5 digit number. You now want to support two lookup operations: **lookup-by-SS** and **lookup-by-SID**, which takes a single SID. Both of them should be as fast as possible. (The lookup-range-SS operation from prat b is not support.) Describe how lookup-by-SID would work using your data structure from part a or describe a new data structure.

Problem 4. *(10 points)* Manipulating heaps.

a. *(6 points)* Draw a picture of heap (stored in an array that results by starting with an empty heap and adding the folowing order: 50, 25, 80, 94, 1, 85, 97. Assume smaller elements have higher priority (i.e., the smallest is at the top).

b. *(4 points)* Draw a picture of heap (stored in an array) that results when the smallest element is deleted from your answer to the previous question.

Problem 5. *(10 points)* In class we noted that the insertion algorithm for B-Trees from the handout may cause a node to split when this is not strictly necessary, i.e., when there is space in child node that should receive the key.

a. *(5 points)* Give an example of a B-tree of order 4 and an element to be inserted, such that this unnecessary splitting would occur. You need not show the B-tree after this insertion, but mark the node that wuld split using the algorithm in the book. Do not show the data nodes in your picture, just show the keys.

b. *(5 points)* This algorithm splits unnecessarily, but is sometimes faster than an algorithm that splits only when the leaf node is full. Explain (in 1 or 2 sentences) why this is true.

Problem 6. *(5 points)* Show all the intermediate steps in a radix sort. It should produce an array in alphabetical order.

```
gag
got
to
get
tag
```

```
sag
rat
go
rag
```

Problem 7. *(10 points)*

a. *(5 points)* Given the following implementation of radix sort, state a loop invariant that could be used to prove it works.

```
RADIX_SORT(A)
  maxKeyLength = length of longest key in A;
  for (i = maxKeyLength-1; i >= 0; i--)
    stably sort A using digit i of each key
```

b. ***(5 points)* Show that if your loop invariant holds for i=k, it also holds for i=k-1.
