READ THIS NOW!

- Print your name in the space provided below.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- Most of the questions involve making a conclusion and supporting it. Be sure you clearly state your conclusions. Being vague or hedging your bets will not be rewarded.
- There are 10 short-answer questions, priced as marked. The maximum score is 120.
- When you have finished, sign the pledge at the bottom of this page and turn in the exam and your fact sheet.
- Aside from the allowed two-page fact sheet, this is a closed-book, closed-notes examination.
- No laptops, calculators, cell phones or other electronic devices of any kind may be used during this examination. If you are seen violating this restriction, your exam form will be collected immediately and you will not be permitted to complete the test.
- You may not discuss this examination with any student who has not taken it.
- Failure to adhere to any of these restrictions is an Honor Code violation.

Name _____
printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

_____
signed

1. [10 points] Determine the Θ-complexity function F(N) of the body of the following function. Assume that calls to operator `new` take time N, that calls to the function `RanGen()` take time 10, and that calls to the function `Sort()` take N log N time. You may assume any other operations take constant time. Your answer for F(N) should be in simplest form. Show supporting work!

```
public static void Mystery(int[] A, int N) {  // Line 1

   int[] B = new int[N];                       // Line 2

   for (int i = 1; i < N; i = 2*i) {           // Line 3

      for (int j = 0; j < N; j++) {            // Line 4
         B[i] = RandGen( );                    // Line 5
      }

      Sort(B, N);                              // Line 6

      A[i] = B[0];                             // Line 7
   }
}
```

$$T(N) = N+1+1+\sum_{pass=1}^{\log N}\left(1+2+1+\sum_{j=1}^{N}(1+1+2+10)+1+N\log N+1\right)+1$$

$$= N+3+\sum_{pass=1}^{\log N}\left(6+\sum_{j=1}^{N}(14)+N\log N\right)$$

$$= N+3+\sum_{pass=1}^{\log N}\left(6+14N+N\log N\right)$$

$$= 3+N+6\log N+14N\log N+N\log^2 N$$

Obviously this is Θ(Nlog^{2}N).

2. [16 points] A contiguous sequence of filled slots in a hash table is called a *run*. Assume that the hash function distributes the set of actual key values uniformly across the hash table (i.e., if a huge number of keys were hashed then each table slot would be expected to be hit the same number of times).

   a) If a hash table uses linear probing to resolve collisions, then the longer a run is the more likely it is that the run will become even longer when the next insertion occurs. Explain why.

The goal that the hashing scheme provides a uniform distribution of keys across the table implies that each slot in an empty table is equally likely to be filled on the next insertion; i.e., the probability is 1/N. For a filled slot, the probability will be 0.

If we have a run of K filled slots, the probability of the immediately succeeding slot being filled on the next insertion will be (K+1)/N. So, the slots most likely to be filled will be those immediately succeeding runs, and filling one of them will make the run even longer, increasing the probability of the next empty slot even more.

   b) Exactly how does the creation of long runs described above affect the performance of searches in the hash table?

The creation of a long run can cause a record to be stored in a slot far from its home slot, after a lengthy sequence of linear probe steps. Those same steps must be repeated if the record is searched for, so this results in an increased cost for some searches, and can prevent overall $\Theta(1)$ average search cost.

3. [8 points] Explain clearly why the use of quadratic probing eliminates the creation of long runs described in the previous question.

With quadratic probing, adjacent home slots do not probe the same set of alternate slots (aside from the first step). So, there is no "unioning" effect, as there would be with linear probing.

4. [12 points] A *range query* involves searching a collection of data values for all the values within a certain interval; e.g. find all customer names between "Alcorn" and "Lombardi".

   Circle the following data structures that organize records in a way that guarantees it is possible to perform an efficient range query. (Given that there are N records in the collection, and R records in the range, and that R is much smaller than N, *efficient* should be interpreted as being $\Theta(R + \log N)$ or better.)

Hash table	BST	Max-heap	B+ tree
Stack	Sorted array	Queue	Sorted linked list

5. [12 points] Describe an efficient algorithm, including choice of data structure, for solving the following problem and state the Θ-complexity of your answer:

   You are given a collection of N integer values (possibly including duplicates) and an integer K. Find two integers, x and y from the given set such that x + y = K, or show that no such integers exist.

Step 1: **Bin-sort (if the range of the data values is small), or use a trivial mod-N hash into a suitably sized array of linked lists (append or prepend each value on collisions).**

Step 2: **Linearly process the data values; for each value V, search the structure from Step 1 to see whether the value K – V is present. Stop if a match is found.**

Step 1 should be Θ(N) as long as the hash table is operating efficiently or there are not too many collisions in the array of lists.

Step 2 is clearly Θ(N) as long as the hash table or array of lists is working well.

So, this has the potential to be Θ(N), provided that things work out well.

Step 1: **Sort the N values into a structure that supports log N worst-case search cost. An array will do, as will an AVL tree.**

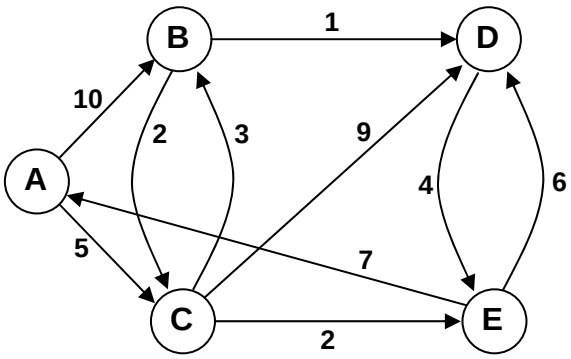
Step 2: **For each element ei in the structure, calculate its "dual" K – ei, and then perform a log N search of the structure to see if the "dual" is present. If it is, we are done.**

Step 1 is Θ(N log N) in the worst case if we choose an appropriate algorithm.

Step 2 requires N iterations, each costing Θ(log N) in the worst case, for a worst-case cost of Θ(log N).

So, the worst-case cost of this is Θ(N log N).

6. [12 points] Apply Dijkstra's SSAD algorithm to find the shortest path from the vertex **A** to every other vertex in the graph below. For each vertex, list the shortest path (as a sequence of node labels) and state the length of the shortest path. Show all work.



Work:

```
A    B    C    D    E
----------------------
0   10    5   inf  inf    {A}
     8        14    7     {A, C}
     8        13          {A, C, E}
              9           {A, C, E, B}
                          {A, C, E, B, D}
```

Destination Vertex	Path from A	Total length of path
A		
B	A -> C -> B	8
C	A -> C	5
D	A -> C -> B -> D	9
E	A -> C -> E	7

7. [10 points] Consider the following functions:

$$f(n) = n^{1.01} + n\log n \qquad g(n) = n^{1.01} \qquad h(n) = n\log n$$

Which of the following is true? f is $\Theta(g)$. f is $\Theta(h)$.

Prove your conclusion is correct by using theorems covered in class. Note: the theorem that says the Θ-category is determined by the dominant term doesn't help since none of the theorems implies which is the dominant term in this case.

$$\begin{aligned}
\lim_{n\to\infty}\frac{f(n)}{g(n)} &= \lim_{n\to\infty}\frac{n^{1.01}+n\log n}{n^{1.01}} = \lim_{n\to\infty}\left(\frac{n^{1.01}}{n^{1.01}}+\frac{n\log n}{n^{1.01}}\right)\\
&= \lim_{n\to\infty}\left(1+\frac{\log n}{n^{0.01}}\right)\\
&= 1+\lim_{n\to\infty}\frac{1/n\ln 2}{0.01n^{-0.99}}\\
&= 1+\lim_{n\to\infty}\frac{1}{0.01\ln 2(n^{0.01})}\\
&= 1+0\\
&= 1
\end{aligned}$$

So, the correct relationship is that f is $\Theta(g)$.

8. [12 points] Consider representing a directed graph containing 10,000 vertices and 200,000 edges. Assume vertices are labeled with consecutive integers from 0 to 9999. The graph can be represented using either an adjacency matrix structure or an adjacency list structure.

   Define the distance from a vertex A to a vertex B to be the minimum number of edges that must be traversed to go from A to B; if it is impossible to go from A to B we say the distance from A to B is infinite.

   Your most important goal in choosing a representation for the graph described above is to minimize the number of operations required to find all vertices that are a distance of two from an arbitrary given vertex *v*. Which representation would provide better performance, in the average case? Support your conclusion by analyzing the cost of each alternative. Remember, space cost is not an issue.

Given a vertex A, how do we find all vertices a distance of 2 from A? We must find all neighbors of A, and then find all of their neighbors, and filter out any of those that were actually a distance of 1 from A.

So, if an adjacency matrix is used, we must traverse the row for A once, and then traverse the row for each neighbor of A. From the given information, there are, on average, 20 edges for each vertex and we have rows of length 10000. So, this would take about 10000 + 20*10000 or 210,000 operations.

If an adjacency list structure is used, then we must traverse the neighbor list for A, and then traverse the neighbor lists for each of A's neighbors. On average, a neighbor list would be 20 elements long, so the cost of this would be about 20 + 20*20 or 420 operations.

9. A completely filled B+ tree of order 10 has 4 levels; the nodes are, of course, stored in a file on disk. Recall that in a B+ tree, full data records are stored only in the leaves.

   A buffer pool with 12 slots is used to cache nodes of the B+ tree as they are fetched from disk. A large number of relatively random searches are performed on the B+ tree. Assume that, on average, each node is equally likely to contain the value being searched for.

   a) [8 points] Describe the treatment of the root node by the buffer pool if the replacement policy is LRU.

Every search begins with the root and progresses to the leaf level, examining exactly 4 nodes. This means that once the root node is initially loaded into the pool, it will be hit before it drops below the 4th position in the pool.

So, the root node will remain in the pool continuously, after the initial load.

   b) [8 points] Assuming that LRU replacement is used, on a random search, how likely is it that the buffer pool will have to go to disk to retrieve a node from level 1 of the tree (i.e., a child of the root)? Discuss carefully.

Under the given assumptions, each level 1 node will be examined on 10% of the searches, each level 2 node on 1% of the searches, and each level 3 node on 0.1% of the searches. Note that the level 1 nodes can ALL be in the pool, along with the root, and still leave one slot free for a node from level 2 or level 3.

Given a sequence of 1000 searches, we'd expect the root to take 1000 hits, each level 1 node to take 100 hits, each level 2 node to take 10 hits, and each level 3 node to take 1 hit.

Furthermore, every search must reach the bottom level in the tree, and so each search will involve accessing four distinct nodes, one per level.

When a level 1 node takes a hit on a search, it moves to the top of the list. We expect the same node will be hit on the tenth search after this one. The intervening 9 searches will each involve three nodes (aside from the root), and so we expect that the level 1 node will be pushed out of the LRU list before it takes another hit.

10. [12 points] Illustrate the application of the radix sort algorithm to sort the following list of integers into ascending order:

47, 73, 18, 23, 32, 55, 37, 13

Pass 1	Pass 2			Gathering phase:
0:		0:		
1:		1:	13, 18	13, 18, 23, 32, 37, 55, 73
2:	32	2:	23	
3:	73, 23, 13	3:	32, 37	
4:		4:		
5:	55	5:	55	
6:		6:		
7:	47, 37	7:	73	
8:	18	8:		
9:		9:		

First is the sheer joy of making things. As the child delights in his mud pie, so the adult enjoys building things, especially things of his own design.

Second is the pleasure of making things that are useful to other people. Deep within, we want others to use our work and to find it helpful.

Third is the fascination of fashioning complex puzzle-like objects of interlocking moving parts and watching them work in subtle cycles, playing out the consequences of principles built in from the beginning.

Fourth is the joy of always learning. In one way or another, the problem is ever new, and its solver learns something.

Fred Brooks on "Why is Programming Fun?"

Have a great Summer!