1. Given an algorithm with input size \( n \), and \( n > 1 \), the best-case efficiency of the algorithm...

   \[ \text{A. is achieved for the best-case input of size } n. \]
   \[ \text{B. can be achieved when } n \text{ very is large.} \]
   \[ \text{C. is achieved when } n \text{ is known in advance.} \]
   \[ \text{D. is always achieved when } n \text{ is small enough.} \]

2. An algorithm runs on your work computer. The algorithm must be run 100 times per day for one year. Each run of the algorithm takes 5-seconds. You know with some coding you could improve the algorithm's running time by 20%. What is the maximum amount of time you should put into the coding so that you do not spend more time coding than you will save by the speed improvements.

   \[ \text{A. 20 hours} \]
   \[ \text{B. 10 hours} \]
   \[ \text{C. 2 hours} \]
   \[ \text{D. 1 hour} \]

3. An adaptive sorting algorithm...

   \[ \text{A. does not have a worst-case scenario.} \]
   \[ \text{B. adapts to always perform at a best-case efficiency.} \]
   \[ \text{C. changes to perform more efficiently when } n \text{ is large.} \]
   \[ \text{D. takes advantage of the initial order of the input data.} \]
   \[ \text{E. relies on the data being in reverse order.} \]

   This was a tough question.

4. Which of the following assertions are true?

   \[ \text{A. } n(n-1)/2 \in O(n^3) \quad \text{n}^2 \text{ is at least as efficient as } n^3 \]
   \[ \text{B. } n(n-1)/2 \in \Omega(n) \quad \text{n}^2 \text{ is not more efficient than } n \]
   \[ \text{C. } n(n-1)/2 \in \Theta(n) \]
   \[ \text{D. } n(n+1)/2 \in \Theta(n^3) \]

5. Time efficiency of an algorithm is best measured

   \[ \text{A. by counting the number of loops.} \]
   \[ \text{B. by comparing the running-time to other algorithms.} \]
   \[ \text{C. by counting the number of times its basic operations are executed.} \]
   \[ \text{D. by using an accurate real-time clock or stopwatch.} \]

6. Name an algorithm we studied which operates with great efficiency because of employing a space and time trade-off?

   Hashing, hash table, perfect hash.
   Some credit for Mergesort, because it uses \( O(n) \) extra memory, but not really considered a tradeoff.
7. Given the algorithm below, for n > 0. What does the algorithm do?

```c
int Mysterious(int arr[], int n)
{
    int k = 0;
    int x = arr[k];
    for (int i=1; i<n; i++) {
        if (x > arr[i]) {
            k = i;
            x = arr[k];
        }
    }
    return k;
}
```

A. Returns the largest value in arr.
B. Returns the index of arr's smallest value.
C. Returns the index of the largest value in arr.
D. Returns the count of values larger than arr[0].
E. Changes all values in arr.

8. How many multiplication operations will occur in the Mystery algorithm below for input n = 16.

(5) t*t surpasses n on the 5th try.

```c
Mystery(n)
    t ← 1
    while (t*t) ≤ n
        t ← t + 1
    return t
```

9. How many recursive calls to the function below will be made after an initial call of foo(65, 34)?

(4)

```c
foo(m, n)
    if n is 0 then
        return m
    else
        return foo(n, m mod n)
```

10. Below is a heap. To add a node to a heap, place the new node at the end of the heap, then percolate up. Insert new nodes for values 7 and 8, in that order. Draw the final heap and circle it.

```
5
/ \
3 4
```

```
8
/ \
7 4
/ \
3 5
```
11. An exhaustive search tests all possible solutions to a problem. Using an exhaustive search, how many possible solutions must be tested for each of the following problems with input size \( n \)?

- Sorting an array: \( n! \)
- Finding the closest pair of points: \( n^2 \)
- Finding the convex hull: \( 2^n \)
- Finding all \( n \)-queens solutions: \( n^n \)

12. Quicksort using 3-way partitioning is especially efficient for data with many duplicate values. Given the following array, show the arrangement of the array after a 3-way partition is complete. Pivot value is arr[15].

\[
\text{arr[]} = \{ 1, 0, 2, 3, 2, 4, 2, 5, 2, 3, 2, 2, 1, 3, 4, 2 \}
\]

\[|____1 0 1_______|__2 2 2 2 2 2__|___3 4 5 3 3 4_____|
\]

13. In question 13 above, data size \( n \) is 16, so the initial call to Quicksort was the following:

\[
\text{Quicksort(arr, 0, 15)}
\]

where 0 and 15 indicate the start and end indices of the array.

After the 3-way partitioning, several recursive calls to Quicksort must be made. Show those calling statements. Hard-code the indices as above.

\[
\text{Quicksort(arr, 0, 2)}
\]
\[
\text{Quicksort(arr, 10, 15)}
\]

14. Below is a list of Huffman codes. Add a valid Huffman code of length 4.

\[
01
11
001
101
1000 ~ add a valid code of length 4 right there.
\]

15. Solve the following nested summation for \( n=5 \). Hint: it's like nested loops.

\[
\sum_{i=0}^{n-1} \sum_{j=i}^{n} 1
\]

(20)
16. Given the following recurrence relation

\[ f(0) = 1 \]
\[ f(n) = f(n-1) + n \]

When solving with backwards substitution, which of the following are valid?

A. \[ f(n-2) = f(n-3) + n-2 \]
B. \[ f(n) = f(n-3) + (n-2) + (n-1) + n \]
C. \[ f(n) = f(n-3) + 2 + 1 + n \]
D. \[ f(n-1) = f(n-1) + n-2 \]

17. Please solve the following matrix using Gaussian elimination.
Show solution matrix. Draw your own solution matrix if you need more room.

\[
\begin{array}{ccc|c}
3 & -1 & | & 10 \\
1 & 0 & | & 2 \\
-1 & 2 & | & -10 \\
0 & 1 & | & -4 \\
\end{array}
\]

solution here

18. Number the following algorithm growth rates 1 to 5, where 1 is the most efficient and 5 is the least efficient.

3 ___ linear
2 ___ logarithmic
5 ___ exponential
4 ___ quadratic
1 ___ constant

19. Your classroom has 2 rows of seats. A row of 7 seats and a row of 5 seats. There are 10 students and exactly 2 of them are twin brothers. If seating is assigned randomly, what is the probability that the twins sit next to each other in the same row?

12 choose 2 = 66
|E| = 10 all the ways for success
|S| = 66 all the possibilities
|E| / |S| = 10/66 = 0.151515

20. You are asked to solve the N-Queens problem using an exhaustive search for a board of size 7x7. Your algorithm can check 10 solutions per second. How long will it take to count all the solutions?

7^7 * .1 = 82354.2 seconds
82354.2 / 3600 = 22.87619 hours

22.87619 hours