Write your name in the top right corner of this page. Please turn off and put away phones and pagers, computers and calculators before the exam begins. Please have your student ID or other photo ID out for inspection. All notes, books or other paper should be placed in your bag and zipped up before starting.

There are 15 multiple choice/short answer questions worth 4 points each, followed by 5 longer questions worth 15 points each. Of the longer questions, the lowest score will be dropped.

If you have any questions during the exam, please remain seated and raise your hand.

**Part I: Short Answer**

15 questions worth 4 points each. Answers should not be longer than one or two sentences. For this section only, you do not need to justify your answers.

1. Let $f(n)$ be a positive function. Which best defines $\Theta(f(n))$?

   (a) The class of all functions that are at most a constant times $f(n)$.
   
   (b) The class of all functions that are at least a constant times $f(n)$.
   
   (c) The class of all functions that are at least a constant times $f(n)$ and at most another constant times $f(n)$.
   
   (d) The class of all functions whose ratio when divided by $f(n)$ approaches 1 as $n$ grows.

2. Which of the following contradicts the statement, *The worst-case running time of the algorithm is $\Omega(2\sqrt{n})$.*

   (a) The algorithm runs for $O(1)$ steps on some inputs.
   
   (b) The algorithm runs in polynomial time. *This upper bound is smaller than the given lower bound (impossible).*
   
   (c) The worst case running time is $O(2^n)$.
   
   (d) The worst case running time is $\Omega(n^3)$. 
3. A divide and conquer algorithm takes an array of size $n$ as input, and makes two recursive calls on arrays of size $n/2$. Then, after an additional $O(n)$ work, it produces an output. The running time should be:

(a) $\Theta(n)$
(b) $\Theta(n \log n)$
(c) $\Theta(n^{\log_2 3})$
(d) $\Theta(n^2)$

(this is the MergeSort recurrence.)

4. In defining the Stable Matchings problem, how did we proceed?

(a) We said a matching is stable if no man wants to propose to someone else's wife.
(b) We defined the output of the Gale-Shapley algorithm to be a stable matching.
(c) We said a matching is stable if and only if every man and woman gets paired with their first choice.
(d) We first defined what an instability is, and said a stable matching is one with no instabilities.

5. Let $G$ be a connected graph with $n$ vertices and $m$ edges. Which of the following best corresponds to the notion of “linear time,” when this graph is the input to an algorithm?

(a) The worst case running time is $O(n)$.
(b) The worst case running time is $O(m)$.
(c) The worst case running time is $O(n^2)$.
(d) The worst case running time is $O((n + m)^2)$.

\[ \text{Since } G \text{ is connected, } m \geq n - 1. \]
\[ \text{So } n + m \text{ is } O(m). \]
6. Let $G$ be a connected graph. Suppose $T$ is a BFS tree for $G$, and $e$ is an edge of $G$ that is not included in $T$. State a relationship between the endpoints of $e$.

The endpoints of $e$ are in the same or adjacent levels of $T$.

7. What is the definition of a **topological ordering** for a directed graph?

Sort the vertices so that all edges go from earlier vertices to later vertices.

8. For a balanced tree of size 5, stored in an array indexed from 0 to 4, what are the indices of the leaves?

```
  0
 / \/
1   2
 / \/
3   4
```

9. Is it better to be on the side proposing, or the side receiving proposals, in the Gale-Shapley algorithm? In one sentence, state why.

Proposing. Every person on the proposing side will end up paired with their favorite partner, out of all possible partners from any stable matching. (For the other side, it will be the worst such partners instead.)

10. Give an example of an order relation on Strings.

$s_1$ is a prefix of $s_2$.

OR suffix

OR substring

OR "is lexicographically before"
11. How big is \( \log(n!) \), where \(!\) denotes factorial. You may use big-O notation. You don’t need to justify your answer.

\( \Theta(n \log n). \) (Since \( (\frac{n}{2})^{\frac{n}{2}} \leq n! \leq n^n \) by comparing individual terms.)

12. Give a definition for a directed graph to be strongly connected.

\( G \) is strongly connected if for all vertices \( V, W \),
there is a directed path from \( V \) to \( W \) and
from \( W \) to \( V \).

13. What can usefully be said about the sequence of fiancés a woman is engaged to, in the “men propose” version of the Gale-Shapley algorithm?

A woman is always prefers any later fiancé over
any earlier one.

14. What is the Heap Order property?

The key stored at a parent node is always
\( \leq \) the key stored at its child. (For any edge in
the tree.)

15. Name one or more algorithmic applications for the Priority Queue data structure.

Dijkstra’s algo. for shortest paths in positive-weighted graph.
Prim’s algo. for MST.
HeapSort \( \leftrightarrow \) sort a list of values.
Huffman coding
(\( \text{etc.} \))
Part II: Problems

5 homework-style problems worth 15 points each. Show all your work, and justify any claims you make. Your lowest score in this section will be dropped.

16. Write pseudocode to determine in linear time whether a graph is 2-colorable or not. If you use any algorithms from the book or class, be sure to include pseudocode for them too.

Note: "2-colorable" = bipartite.

Outline: (1) Use BFS or DFS to build a spanning forest for G.
(2) Check to see if any edge of G joins 2 vertices whose levels have the same parity (both even or both odd). If so, G is not 2-colorable.
(3) Otherwise, G is 2-colorable, and the level, mod 2, can be used as the coloring function.

Note: steps (1) and (2) can be done separately, or combined into one.

Pseudocode:

Mark all nodes "unfound".
while (exists unfound node) {
  let v be an unfound node.
  explore (v, 0)
  3. output true. //assuming we reach this line, G is 2-colorable.
method: explore (node v, color c) //essentially DFS.
  mark v as found, color it c.
  for all neighbors w of v:
    if (w is unfound)
      explore (w, 1-c) //the opposite color.
    else if (color(v) == color(w))
      output FALSE; terminate.
  //end method.
}
17. For parts (a) and (b), say whether \( f \) is \( O(g) \), \( \Omega(g) \), \( \Theta(g) \), or none of the above. Justify your answers. "log" denotes base-2 log by default.

(a) \( f(n) = 2^n \), \( g(n) = 4\sqrt{n} \).

\[
\begin{align*}
  f &= \Delta^2(g), \quad g(n) = 2^{2\sqrt{n}}. \quad \text{Since} \\
  f(n) &= 2^n = (2^{2\sqrt{n}})^{n/2} = g(n)^{n/2} \\
  \text{is vastly bigger than } g(n), \text{ for large } n. \\
  (\text{crossover point is where } \sqrt{n/2} = 1, \text{ i.e. } n = 4).
\end{align*}
\]

(b) \( f(n) = \log(n) \), \( g(n) = \log(n^2) \).

\[
\begin{align*}
  f &= \Theta(g), \quad \text{since} \quad g(n) = \log(n^2) = 2\log(n) = 2f(n).
\end{align*}
\]

(c) State the formal mathematical definition of \( f(x) = O(g(x)) \).

There exist \( C > 0 \) and \( n_0 > 0 \) such that, for every \( n \geq n_0 \), we have \( f(n) \leq C \cdot g(n) \).

(d) From the definition, prove formally that \( 10n^2 + 100n = O(n^2) \).

Let \( C = 11 \) and \( n_0 = 100 \). Suppose \( n \geq n_0 \).

Then \( 10n^2 + 100n \leq 10n^2 + n^2 \) (since \( n \geq 100 \))

\[
\begin{align*}
  &= 11n^2 \\
  &= C \cdot n^2.
\end{align*}
\]
18. Give a fast algorithm for this problem and analyze its running time using big-O notation.
INPUT: An integer array, A, of length $n$.
OUTPUT: Indices $i$ and $j$ such that $i \leq j$ and the sum $A[i] + A[i+1] + \ldots + A[j]$ is the maximum possible.

**Naïve solution:** try all pairs $i \leq j$. For each compute the sum.
Store the max. Runtime = $\Theta(n^3)$.

**Better:** Don’t compute each sum from scratch.
Saves factor of $n$: $\Theta(n^2)$.

**Fastest:** Greedy. Keep track of:
- $\text{Best}[i]$ = best solution using $A[0\ldots i]$ and
- $\text{BestIncluding}[i]$ = best solution from $A[0\ldots i]$ that actually includes $A[i]$

Start: $\text{Best}[-1] = 0$, $\text{BestIncluding}[-1] = 0$.

For ($i = 0$ to $n-1$) {
    $\text{BestIncluding}[i] = \max \{ A[i], A[i] + \text{BestIncluding}[i-1] \}$
    $\text{Best}[i] = \max \{ \text{Best}[i-1], \text{BestIncluding}[i] \}$
}

Output: $\text{Best}[n-1]$.

Running time: $\Theta(n^2)$.
19. Consider the following variant of (unweighted) Interval Scheduling:

There are $N$ requests, and request $i$ has its start time given to you in array element $\text{start}[i]$ and end time in array element $\text{end}[i]$. As usual, the goal is to schedule as many of the requests as possible. The twist is that this time, if you schedule $k$ of the requests, then there must be a "buffer" of at least $k$ time units between any two of the scheduled requests. Give an almost-linear time algorithm to solve this problem, and analyze its running time.

**Hint:** Use binary search to figure out how many requests to schedule.

```java
low = 0
high = n

while (low + high) // round up
    k = (low + high) / 2 // binary search to find the right k.
    if schedule(start, end, k) {
        low = k;
    } else high = k;
    //end while

method: schedule(start, end, k) {
    // can k jobs be scheduled? O(n) time steps.
    boolean sorted = start and end together using values at end. // better do this only once in preprocessing.
    lastend = end[0]; // first job automatically scheduled.
    for (i = 1 to n-1) {
        count = 1;
        if (start[i] >= lastend + k) { // then add job i,
            count++;
            lastend = end[i];
        }
        --end for;
        if (count >= k) return true;
    } else return false;
} //end method schedule.
```

**Analysis:** $O(n \log^2 n)$ time as written. $O(n \log n)$ if the arrays only are sorted once for all.
20. Draw the graph with vertex set \{1, 2, 3, 4, 5, 6, 7\} and edges \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{2, 6\}, \{3, 4\}, \{3, 7\}, \{4, 5\}, \{6, 7\}. Find and draw an example of a BFS and a DFS tree for this graph (clearly indicate the root for each). Prove that breadth-first and depth-first search on this graph cannot result in the same tree, even if different starting vertices are used.

All possible edges are present among vertices 1, 2, 3, 4.

Thus, for a BFS tree, these nodes must all be in one or two consecutive levels.

Also, for a DFS tree, each of these nodes must be an ancestor or descendant of each other one; thus, they must be on 4 different levels.

Since the underlined bits can’t both happen, any tree can be at most one of DFS or BFS.