University of Utah
School of Computing

CS 4150 Midterm Exam Solutions Oct 20, 2016

The average grade on the midterm was 73.

If you believe that a mistake was made in grading your exam, write an explanation on the cover sheet and return the exam to me by Monday, October 31.

Master Theorem

If $T(n) = aT(n/b) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

1. Four algorithms are carefully timed on n-bit integers (modular exponentiation), dense graphs with n vertices (Dijkstra's algorithm), ordered arrays of length n (binary search), and sparse dags with n vertices (topological sort). A different computer is used to time each implementation.

The results of the timing experiments are summarized below. All times are expressed in the same (unspecified) units.

n	Algorithm W	Algorithm X	Algorithm Y	Algorithm Z
4000	1	100	2000	4
8000	8	200	3000	16
16000	64	400	4000	64
32000	512	800	5000	256
64000	4096	1600	6000	1024

Identify the algorithms by writing W, X, Y, and Z in the appropriate boxes below.

Algorithm	Identity
Modular exponentiation	W
Dijkstra's algorithm	Z
Binary search	Y
Topological sort	X

2. Give the value of each of the expressions below. In each case, the result must be a non-negative integer less than the modulus.

Expression	Value
$(5 + 8) \bmod 7$	6
$(5 * 8) \bmod 7$	5
$(1872387487398 * 3498738472987348) \bmod 2$	0
$3^{10293987756798768400} \bmod 80$	1

3. Consider the following method:

```
public static int f (int n) {
  int total = 0;
  for (int i = 0; i <= n; i += 2) {
    for (int j = i+2; j <= n; j += 2) {
      total++;
    }
  }
  return total;
}
```

What does `f(2*k)` return? Assume that $k > 0$. Give your answer in terms of k. *Give a simplified expression that does not contain a summation.*

Answer	k(k+1)/2

4. For each pair of functions below, write

- $\Theta(g)$ if f is $\Theta(g)$

Otherwise, write

- $O(g)$ if f is $O(g)$ or
- $\Omega(g)$ if f is $\Omega(g)$

f	g	Answer
$2n^2$	$5n^2$	$\Theta(g)$
2^n	3^n	$O(g)$
$(\log n)^{1000}$	$n^{.001}$	$O(g)$
n^2	$100n \log n$	$\Omega(g)$

5. Use the Master Theorem to derive a tight upper bound on the following two recurrences:

$$\begin{aligned} T_1(n) &= 3\,T_1(n/3) + O(n) \\ T_2(n) &= 4\,T_2(n/2) + O(n^3) \end{aligned}$$

Recurrence	Bound
T_1	$O(n \log n)$
T_2	$O(n^3)$

6. The two fundamental operations on a priority queue are `insert` (which inserts a new key into the priority queue) and `deleteMin` (which removes the smallest key from the priority queue).

Heapsort is a comparison-based sorting algorithm that takes advantage of these two operations. It creates an empty priority queue, into which it `inserts` each of the n elements to be sorted. It then does n consecutive `deleteMin` operations, storing the elements that come out into an array. Since the elements come out of the priority queue in ascending order, the resulting array is sorted.

Assuming that we use the *unordered array* representation of priority queues discussed in class, give tight upper (using O() notation) and lower (using $\Omega()$ notation) bounds on the running time of heapsort.

Upper bound	$O(n^2)$
Lower bound	$\Omega(n^2)$

7. Refer back to the previous question for the discussion of priority queues and heapsort.

Assuming that we use the *binary heap* representation of priority queues discussed in class, give tight upper (using O() notation) and lower (using $\Omega()$ notation) bounds on the running time of heapsort.

Upper bound	$O(n \log n)$
Lower bound	$O(n)$ **

** Heapsort runs in linear time when all the elements being sorted are identical because the heap operations are all constant time in this case. We also accepted $O(n \log n)$ since we never discussed this point.

8. Refer back to the two previous questions for the discussion of priority queues and heapsort.

Explain, in one of two sentences, why it is impossible to implement a comparison-based priority queue for which both `insert` and `deleteMin` are $O(1)$.

Constant time heap operations would yield an $O(n)$ heapsort. We proved, however, that this is impossible; comparison-based sorting is $\Omega(n \log n)$.

9. Use $O()$ and $\Omega()$ notation to give the tightest possible upper and lower bounds on the time required to determine whether every vertex in a directed graph has an edge to itself. Assume that the graph has V vertices and E edges.

Do this once assuming that the graph is represented as an adjacency matrix and again assuming that the graph is represented as an adjacency list.

	Matrix	List
Upper bound on worst case	$O(V)$	$O(E+V)$
Lower bound on best case	$\Omega(1)$	$\Omega(1)$

10. Consider this method g, which operates on an array of integers. The helper method `extract` is also defined.

```
// Does a curious computation on an array
int g (int[] A) {
    if (A.length == 0) {
        return 0;
    }
    else if (A.length == 1) {
        return A[0];
    }
    else {
        int n = A.length;
        return g(extract(A, n/5, 2*n/5)) + g(extract(A, 3*n/5, 4*n/5));
    }
}


// Returns an array containing A[lo] through A[hi-1].
int[] extract (int[] A, int lo, int hi) {
    int size = Math.max(0, hi-lo);
    int[] result = new int[size];
    for (int i = 0; i < size; i++) {
        result[i] = A[lo + i];
    }
    return result;
}
```

Find a recurrence relation of the form

$$T(n) = a\,T(n/b) + O(n^d)$$

that gives a tight upper bound on the amount of time required to compute $g(A)$ on an n-element array A. Below, give your values for a, b, and d. Also, use the Master Theorem to find an $O()$ bound for the recurrence.

	Answer
a	2
b	5
d	1
Bound	$O(n)$

11. Suppose that the return statement at the end of the method `g` from the previous problem is replaced with this:

```
return g(extract(A, 0, n/3)) + g(extract(A, n/3, 2*n/3)) + g(extract(A, 2*n/3, n));
```

Repeat the analysis from the previous problem.

	Answer
a	3
b	3
d	1
Bound	$O(n \log n)$

12. Consider the following four functions:

- $f_1(n) = \log(n!)$
- $f_2(n) = \log(2^n)$
- $f_3(n) = n^2$
- $f_4(n) = \sqrt{n}$

Put these functions in order of increasing asymptotic complexity by writing f_1, f_2, f_3, and f_4 in the appropriate boxes below.

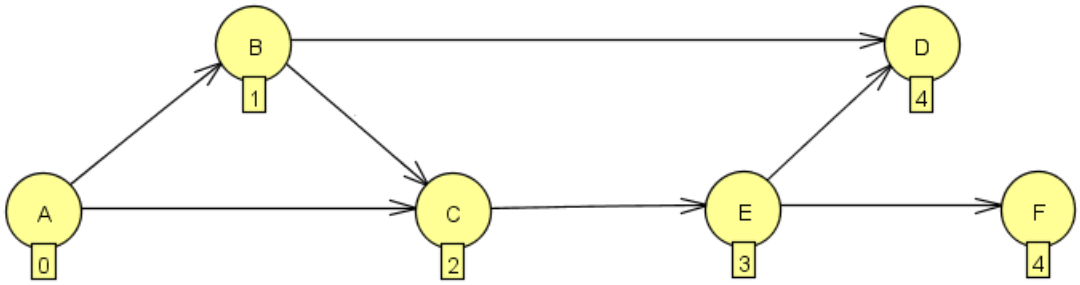
Lowest asymptotic complexity	f_4
Next highest asymptotic complexity	f_2
Next highest asymptotic complexity	f_1
Highest asymptotic complexity	f_3

13. Ben Bitdiddle has invented a novel divide and conquer algorithm for computing the square roots of n-bit integers. It is asymptotically faster than the standard iterative algorithm. Ben's timing measurements reveal that the two algorithms use the same amount of time when n is approximately 100000. He decides to blend the two algorithms, and is wondering at what value for n his blended algorithm should cut over from the new algorithm to the old one.

Should he expect the optimal cutover point to be approximately 100000, significantly larger than 100000, or significantly smaller than 100000?

Answer	significantly smaller

14. In this problem and the next you will be designing an algorithm for directed acyclic graphs that have a single source vertex, such as the one below:



What we need is an algorithm that will calculate, for each vertex, the length of the *longest* path from the source to the vertex. The graph above is annotated with its solution.

Let `longest[u]` be the length of the longest path from the source to vertex `u`. Naturally, if `A` is the source, `longest[A]` will be zero. Below, describe in one or two sentences the computation you must perform on each vertex `u` to calculate `longest[u]`. Describe the individual vertex computation, not the entire algorithm.

Set longest[u] to one more than the maximum of the longest[] values of the immediate predecessors of u.

15. Describe, in one sentence or less, the order in which the computation you described in the previous problem should be applied to the vertices of the dag.

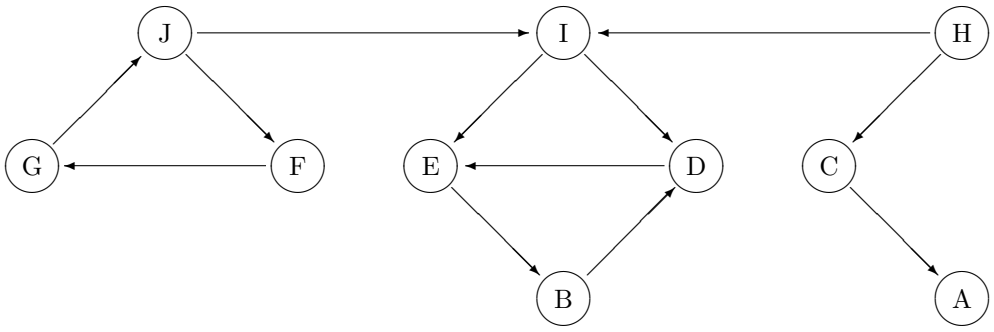
Apply the computation in topological order.

16. In this problem we will consider four sorting algorithms: selection sort, insertion sort, quicksort with the tail-recursion optimization discussed in class, and mergesort with the merge optimization discussed in class.

Answer the following two questions:

Which algorithm's decision tree is the most balanced?	selection sort
Which algorithm's decision tree has the smallest height?	mergesort

The directed graph X diagrammed below is the subject of the next six questions.



17. Suppose that you do a breadth-first search of X beginning at vertex J. Answer the following two questions.

How many vertices are incorporated into the resulting breadth-first search tree?	7
How many edges appear in the longest root (J is the root) to leaf path in the resulting breadth-first search tree?	3

18. Suppose that you do a depth-first search of X. Suppose also that whenever there is an arbitrary choice of vertices to visit (in either `dfs` or `explore`), you always pick the one that comes first in the alphabet. Give the pre and post numbers requested below. (The lowest pre number should be 1, not 0.)

Pre number of A	1
Post number of B	8
Pre number of C	9
Post number of D	7

19. This problem concerns the depth-first search described in the previous problem. Give the additional pre and post numbers requested below.

Pre number of G	12
Post number of H	20
Pre number of I	14
Post number of J	16

20. Find the strongly connected components of X. In the table below, list the SCCs of X's meta-graph that are meta-sources. List an SCC by giving the vertices of X that make it up. For example, if you think H, C, and A make up a meta-source, you would write HCA in one of the boxes of the table.

You may, or may not, need to use all three boxes.

Meta-Sources
FGJ
H

21. Continue with the previous problem by listing below the SCCs of X's meta-graph that are meta-sinks.

You may, or may not, need to use all three boxes.

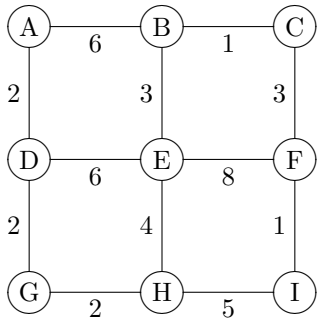
Meta-Sinks
BDE
A

22. Continue with the previous problem by listing below the SCCs of X's meta-graph that are neither meta-sources nor meta-sinks.

You may, or may not, need to use all three boxes.

Other SCCs
C
I

The next three questions concern this undirected, weighted graph Y.



23. Suppose Dijkstra's algorithm is used on Y to find a shortest paths tree rooted at vertex B. The first vertex to be removed from the priority queue via a `deleteMin` operation will be B. What will be the third and fifth vertices removed? (If there is a tie, choose the vertex that comes earlier in the alphabet.)

Vertex	Answer
Third vertex	E
Fifth vertex	I

24. Continuing with the previous question, what will be the seventh and ninth vertices removed from the priority queue?

Vertex	Answer
Seventh vertex	H
Ninth vertex	G

25. Suppose that the Bellman-Ford algorithm is used on Y to find a shortest paths tree that is rooted at B. Assume that the algorithm uses the early termination optimization discussed in class. Answer each question below with an integer. Keep in mind that the number of iterations required by Bellman-Ford is determined in part by the order in which the edges are considered.

Question	Answer
Including the final iteration during which nothing changes, what is the fewest number of iterations that Bellman-Ford could possibly take when finding the shortest paths tree for Y rooted at B?	2
Including the final iteration during which nothing changes, what is the largest number of iterations that Bellman-Ford could possibly take when finding the shortest paths tree for Y rooted at B?	4