4520/6520 Design and Analysis of Algorithms

Mockup Midterm

Instructor: Alina Nemira

10/06/2025

Name: _____

Honor Code Statement: I will not commit any act of academic dishonesty while completing this assignment. I am fully aware that any of my own personal actions while attempting this assignment that are interpreted as academic dishonesty, will be treated as such. I understand that if I am held accountable for an act of academic dishonesty that I will receive a grade of "0" (zero) for this assignment and the incident will be reported to the Dean of Students Office.

The mockup midterm contains 12 pages (including this cover page) and 11 problems. Total of points is 125. You will receive min{your score, 100} points.

Good luck and productive work!

4520/6520 Design and Analysis of Algorithms 10/06/2025

Distribution of grades

Question	Points	Score
1	8	
2	10	
3	10	
4	5	
5	20	
6	10	
7	14	
8	12	
9	10	
10	16	
11	10	
Total:	**125**	

1. (8 points) A smart guy Carl thinks it is possible to create a better version of the Merge sorting algorithm by breaking the array into three pieces (instead of two as it was described in class). Of course, this idea came to his mind, after he could come up with an algorithm to merge three sorted arrays in linear time. He implemented his algorithm and called the procedure Carl'sMerge(a[1..n], l, r). Supposing a[1..l], a[l + 1..r] and a[r + 1..n] are sorted, Carl'sMerge merges them into one sorted array in Θ(n) time. Based on this procedure, Carl has designed his recursive algorithm that follows.

Algorithm 1. Carl's version of MergeSort

1. **function** Carl'sSort($a[1..n]$)
2. **if** $n > 1$ **then**
3. $l \leftarrow \lfloor \frac{n}{3} \rfloor$
4. $r \leftarrow \lfloor \frac{2n}{3} \rfloor$
5. Carl'sSort($a[1..l]$)
6. Carl'sSort($a[l+1..r]$)
7. Carl'sSort($a[r+1..n]$)
8. Carl'sMerge($a[1..n], l, r$)

Estimate the running time of this algorithm. Is this algorithm faster than regular Merge-Sort? Explain your answer.

2. (10 points) Describe a divide-and-conquer algorithm that finds the maximum difference between elements of a given array of size n in $O(n)$ time.

For instance, on input $[4, 5, 10, -4, 2, 4, -7]$, your algorithm should return 17.

Note: for full marks, your answer must make use of the divide-and-conquer approach.

3. Fill in each gap with a suitable word from the list below.

(a) (4 points) _____ dynamic programming solutions make recursive calls according to the recurrence relation while _____ dynamic programming solutions strategically iterate over each state.

List of words:
top-down
bottom-up

(b) (6 points) _____ algorithm solves a problem by dividing it into smaller non-overlapping subproblems.

_____ algorithm chooses at each step the best at a time option that eventually will form a solution.

_____ algorithm solves a problem by dividing it into smaller subproblems that overlap; stores their solutions to avoid repeated calculations.

List of words:
Greedy
Divide and Conquer
Dynamic Programming

4. (5 points) Which of the following must be done when converting a top-down dynamic programming solution into a bottom-up dynamic programming solution? Select all that applies, no explanation needed here.

A. Change plus to minus in the recurrence relation.
B. Use recursion instead of iterations.
C. Create new base cases.
D. Use iterations instead of recursion.
E. Find the correct order in which to iterate over the states.

5. Decide if the statements below are TRUE or FALSE. Explain your answers. *No credit here without a proper explanation!*

(a) (4 points) With all equal-weighted intervals, a greedy algorithm based on earliest finish time will always select the maximum number of compatible intervals.

(b) (4 points) Any dynamic programming algorithm that solves n subproblems must run in $\Omega(n)$ time.

(c) (4 points) $n^2 \in O(2^n)$.

(d) (4 points) The order of items in the sequence interface is intrinsic, while the set interface has an extrinsic order of items.

(e) (4 points) A divide and conquer algorithm for finding a maximum element in an array problem that divides the array into two half-size subarrays and does a constant amount of extra work to merge the results would yield a linear time algorithm.

6. In some town, n houses are located on one side of the road, one by one (see Figure 1).

Figure 1: An example of a street illuminated by lamp posts.

You are given an array of m lamp posts with non-negative integer light values, where 0 means that the lamp post can only light up a section outside one house, 1 means lighting a given house plus two immediately neighboring houses from the lamp pole, and a light value of k means that houses a distance k away, in addition, will be illuminated. You can place a lamp post next to any house. A house may be lit by several lamp posts.

(a) (2 points) If we have an array of five lamp posts $[3, 5, 2, 6, 1]$, what is the minimum number of lamp posts needed to light the street with houses len $= 24$?

(b) (8 points) Prove the greedy algorithm that sorts the set of lamp posts in decreasing order (and picks lamp posts until all houses on the street are lit) is optimal.

7. The Tribonacci sequence T_n is defined as follows:

$T_0 = 0$, $T_1 = 1$, $T_2 = 1$, and $T_{n+3} = T_n + T_{n+1} + T_{n+2}$ for $n \geq 0$. Given n, return the value of T_n.

(a) (10 points) Draw a graph of computations for a dynamic programming solution of this problem.

(b) (2 points) What is a tight bound on the running time of the dynamic programming algorithm?

(c) (2 points) What would the run-time be without memoization (saving solutions to the subproblems in memory)?

8. (12 points) Three divide-and-conquer algorithms A, B and C are proposed to solve the same problem. Suppose they are all correct, what are their running time (in $\Theta(\cdot)$ notation) and which one is more efficient? Justify your answer.

Algorithm A: divides the problem of size n into 10 subproblems of size $n/10$ and combines the solutions in linear time.

Algorithm B: divides the problem of size n into 4 subproblems of size $n/2$ and combines the solutions in constant time.

Algorithm C: divides the problem of size n into 9 subproblems of size $n/3$ and combines the solutions in quadratic time.

9. (10 points) An integer array contains n distinct integers. It is known that exactly $k = \lceil \sqrt{n} \rceil$ elements in the array are even. The odd integers in the array appear in sorted order. Describe an $O(n)$-time algorithm to sort the array.

10. Each question may have one or more correct answers. No need for explanations here.

(a) (4 points) How many comparisons does selection sort make on an input array that is already sorted?
A. Constant
B. Logarithmic
C. Linear
D. Quadratic
E. Exponential

(b) (4 points) How many comparisons does insertion sort make on an input array that is already sorted?
A. Constant
B. Logarithmic
C. Linear
D. Quadratic
E. Exponential

(c) (4 points) What data structure is appropriate if you need insertion, deletion, and look up to be performed in $O(1)$ time (on average)?
A. Static sorted array
B. Dynamic unsorted array
C. Dynamic sorted array
D. Linked list
E. Hash set

(d) (4 points) Which of the following operations has a time complexity of $\Theta(1)$?
A. Adding an item to the back of a queue
B. Removing an item from the front of a queue
C. Finding an arbitrary element in a queue
D. Iterating through a queue
E. Peeking at the front of a queue

11. (10 points) You are given an integer array of size n. Two elements in the array are called coupled if they differ by one. Describe an algorithm that in $O(n)$ counts all *distinct* coupled pairs in the array.