CMSC 251: Algorithms

Spring 1998

http://www.cs.umd.edu/~mount/251/

Instructor: Dave Mount. Office: AVW 3209. Email: mount@cs.umd.edu. Office phone: (301) 405–2704. Office hours: Mon-Wed 2:00–3:00. I am also available immediately after class for questions. Feel free to send me email if you cannot make these times to set up another time.

Teaching Assistant: Kyongil Yoon. Office: AVW 1151. Email: kiyoon@cs.umd.edu. Office hours: Tue 2:00–3:00, Wed 1:00–2:00. Send him email if you cannot make these times, to set up another time.

Class Time: Tue, Thur 11:00–12:15. Room: CLB 0111.

Course Overview: This course presents an introduction to the techniques for designing efficient computer algorithms, proving their correctness, and analyzing their running times. General topics include asymptotics, solving summations and recurrences, algorithm design techniques (such as divide-and-conquer, dynamic programming, and greedy algorithms), analysis of data structures, sorting, search, and selection, and introduction to NP-completeness.

Text: T. Cormen, C. Leiserson, R. Rivest, *Introduction to Algorithms*, McGraw Hill and MIT Press, 1990.

Prerequisites: CMSC 112 or 114, CMSC 150. Each student is expected to know the basic concepts of programming (e.g. loops, pointers, recursion), discrete mathematics (proofs by induction), and calculus (logarithms, differentiation, integration).

Course Work: There will be about 7–10 homeworks. Because solutions will be discussed in class on the due date, homeworks are to be turned by the start of class on the due date. Late homeworks will be not be accepted. (In other words, hand in whatever you have finished.) The lowest homework grade will be dropped. In exceptional circumstances (illness, university business, religious observances) extensions may be granted. However, all extensions must be approved by the course instructor BEFORE the due date.

All class work is to be done independently. You may discuss homework problems and general solution strategies with classmates, but no written communication (by paper, on chalkboards, by email, etc.) is permitted. Instances of academic dishonesty will be dealt with harshly.

As a courtesy to the grader, homeworks are to be written neatly. Poorly written work will not be graded. When writing algorithms be sure not only that your solution is correct, but also that it is easy for the grader to understand why your solution is correct. Part of your grade will be based not only on correctness, but also on the simplicity, clarity, and elegance of your solutions. The burden of the proof of correctness of your algorithms is on you.

Grading: Final grades will be based on the homework assignments, two midterm exams (tentatively planned for March 3 and April 7) and a comprehensive final exam. The relative weights of these will be *roughly* 25% for the homework total, 20% for each midterm, and 35% for the final exam.

Syllabus: The topics and order listed below are tentative and subject to change.

Analysis of Algorithms: Basics of the analysis of algorithms, worst-case behavior, order notation, asymptotics, recurrences, summations.

Sorting and Selection: Comparison-based sorting algorithms (mergesort, quicksort, and heapsort), medians and order statistics, lower bounds on sorting, radix and counting sorts.

Other Algorithms: Algorithms on graphs, strings, and/or geometric objects.

NP-completeness: Complexity classes P and NP, examples of reductions.

Homework 1: Basics

Handed out Thurs, Jan 29. Due at the start of class Thurs, Feb 5.

Problem 1. Consider the 7 log identities at the bottom page 34 in CLR. Assuming that the first four identities are true (and using whatever basic facts about logarithms you like) derive the last 3 identities. For any $a, b, n > 0$:

(a) $\log_b(1/a) = -\log_b a$.

(b) $\log_b a = 1/\log_a b$.

(c) $a^{\log_b n} = n^{\log_b a}$.

Problem 2. Prove the following formula by induction on n. For all $n \geq 1$:

$$\sum_{i=0}^{n-1} i(i-1)(i-2) = \frac{n(n-1)(n-2)(n-3)}{4}.$$

Problem 3. You are given two multiple precision positive integers expressed as two arrays, $A[0..n]$ and $B[0..n]$. Each entry of each array contains a single decimal digit from 0 to 9, such that the least significant digit is stored in the 0-th entry of the arrays. For example, for $n = 5$, the numbers $A = 9,725$ and $B = 153,496$ would be stored as $A[0..5] = \langle 5, 2, 7, 9, 0, 0\rangle$ and $B[0..5] = \langle 6, 9, 4, 3, 5, 1\rangle$.

(a) Give pseudocode for a procedure which, given A, B, and n, computes the sum of A and B and stores the result in array $C[0..n+1]$. Briefly explain how your procedure works. Remember, that when writing pseudocode, keep the description simple and to the point. You do not need to give a complete program with full declarations. Let your explanation handle any missing details.

(b) Give pseudocode for a procedure which, given A, B, and n, computes the product of A and B and stores the result in an array $D[0..2n+1]$. You may invoke the procedure from part (a) in solving this part. Briefly explain how your procedure works.

Homework 2: Asymptotics

Handed out Tues, Feb 10. Due at the start of class Tues, Feb 17.

Problem 1. Use the formal definitions (not the Limit Rule) to establish the following. In each case state specific values of the constants (e.g., c_1, c_2, n_0) you used to satisfy the conditions, and show how you arrived at these values. (There are many potentially correct choices.)

(a) $2n^3 - 3n^2 + 10 \in \Theta(n^3)$.

(b) $n^2 + 10n \lg^2 n \in O(n^2)$. (Hint: Find n_0 such that $\lg^2 n \leq n$, for all $n \geq n_0$.)

Problem 2. Repeat Problem 1, but this time use the Limit Rule.

Problem 3. Rank the following functions in increasing order of asymptotic growth rate. For functions that are asymptotically equivalent, group them together. You do not need to prove your ordering, but you may supply explanations for the purpose of getting partial credit.

$$\begin{aligned}
f_1(n) &= 500n^3, \\
f_2(n) &= 17n + (2/n^2), \\
f_3(n) &= 7n \lg\lg n, \\
f_4(n) &= 20n \lg^3 n + 5n^2, \\
f_5(n) &= 4\sqrt{n} + 3\lg(n^2), \\
f_6(n) &= 2^{\sqrt{n}}, \\
f_7(n) &= 2^{(3 \lg n)}, \\
f_8(n) &= 5\lg^2 n + 10 \lg n, \\
f_9(n) &= 20 \lg(n^2), \\
f_{10}(n) &= 4 \log_3 n
\end{aligned}$$

Homework 3: Divide-and-Conquer and Recurrences

Handed out Thurs, Feb 19. Due at the start of class Thurs, Feb 26.

Problem 1. Consider the following recurrence, defined for n a power of 3.

$$T(n) = \begin{cases} 2/7 & \text{if } n = 1, \\ 2T(n/3) + n^2 & \text{otherwise.} \end{cases}$$

Solve this recurrence exactly using the method of iteration. For full credit, simplify the final expression as much as possible.

Problem 2. Use the Master Theorem to derive an asymptotic bound on the recurrence in Problem 1.

Problem 3. Describe an $O(n \log n)$ divide-and-conquer algorithm for the 2-d maxima problem. The input to your algorithm should be an array of n points $P[1..n]$, where $P[i].x$ and $P[i].y$ are the x- and y-coordinates of the ith point. You may store the output however you like. Your algorithm should *not* explicitly invoke any sorting algorithms. Informally explain your algorithm's correctness, and derive its asymptotic running time. You may assume that there are no duplicate x- or y-coordinates, if it helps simplify your algorithm.

Homework 4: Sorting

Handed out Tues, Mar 10. Due at the start of class Tues, Mar 17.

Problem 1. Consider the following algorithm for selection sort. Recall that the algorithm operates in stages, from 1 to $n-1$. During the ith stage, the algorithm finds the minimum elements of the subarray $A[i..n]$ and swaps it into its final sorted location in $A[i]$.

```
SelectSort(int n, array A[1..n]) {
    for i = 1 to n-1 do {
        min = i
        for j = i+1 to n do
            if (A[j] < A[min]) min = j
        swap A[i] with A[min]
    }
}
```

(a) As a function of n, express *exactly* how many times the test in the "if" statement is executed. Express your answer as a simple expression with no embedded summations or recurrences.

(b) In class we said that selection sort is stable. This is *not* true. Give an example of an array with some duplicate elements, such that the relative position of a duplicate pair of items is reversed after the sort is finished. To keep things straight, label duplicate elements with subscripts, e.g. 2_a, 2_b, etc. (To make the grader's life easy, you will receive the maximum credit by giving the smallest counterexample.) Briefly explain your counterexample.

(c) It is natural to consider whether there is some simple change to the algorithm which will make it stable. Suppose we change the $<$ in the algorithm above with $\leq$. Show that the algorithm is still unstable even after this change.

Problem 2. Do Exercise 7.2-4 on page 144 in CLR.

Problem 3. You are given an array $A[1..n]$ of real numbers, some positive some negative. Design an $O(n \log n)$ algorithm which determines whether A contains two elements $A[i]$ and $A[j]$ such that $A[i] = -A[j]$. (If A contains the element 0, then the answer is always yes.) Briefly explain your algorithm and derive its running time.

Challenge Problem: (Challenge problems are graded for extra credit points. These points are only considered after the final cutoffs have been assigned.) Consider a 2-dimensional array $A[1..m, 1..n]$. Suppose we sort each of the m rows of A, and then sort each of the n columns of A. Prove that the rows remain sorted. (Hint: First show that after both sortings has finished, for each $A[i, j]$ ($j \geq 2$), there are at least i elements in column $j-1$ that are less than or equal to $A[i, j]$. Then explain why this implies that $A[i, j-1] \leq A[i, j]$.)

Homework 5: More Sorting

Handed out Thurs, Mar 19. Due at the start of class Thurs, Apr 2.

Problem 1. For each of the following algorithms, derive its asymptotic Θ running time given that the input array is given in reverse sorted order (that is, they keys appear in decreasing order in the initial array). You may assume all the elements are distinct. In each case briefly explain your answer, but a formal proof is not required.

(a) MergeSort.

(b) HeapSort.

(c) QuickSort. (Rather than picking the pivot at random, assume that the pivot is always chosen to be the first element in the subarray, that is, $A[p]$.)

Problem 2. Problem 7.5-6 on page 151 of CLR. You may explain your algorithm at a high-level. Briefly explain how your algorithm works and derive its running time.

Problem 3. You are given an array of n keys, each with one of the values *red*, *white*, and *blue*. Give an $O(n)$ algorithm for rearranging the keys so that all the *reds* come before all the *whites*, and all the *whites* come before all the *blues*. The only operations permitted are examination of a key to find out what color it is, and swap of two keys (specified by their indices). Explain your algorithm and derive its running time.

Homework 6: Graphs

Handed out Tues, April 14. Due at the start of class Tues, April 21.

Problem 1. As a function of n, what is the minimum number of edges in a *connected* undirected graph with n vertices? Explain.

Problem 2. Problem 5-1 on page 97 or CLR on graph coloring. Do only parts a. and c.

Problem 3. You are to organize a tournament involving n competitors. Each competitor must play exactly once against each of the $n-1$ possible opponents. Each competitor is to play at most one match per day.

(a) Show that if n is a power of 2, then it is possible to design a tournament that takes exactly $n-1$ days (which is optimal). Do this by giving an algorithm which inputs n, and outputs the list of player pairings for each of the $n-1$ days. (Hint: Use divide-and-conquer.)

(b) Consider a graph in which the n players are the n vertices of the graph. Express the problem of whether there exists a k-day tournament as an edge-coloring problem in graphs.

Challenge Problem: You have a friend who is an unreliable programmer. He has just written a program, which is supposed to produce a long linked list. The head of the list is given by a pointer `head`. Each entry p in the list has two fields, a key field, `p.key` (which you may assume is different for every entry in the list) and a pointer to the next element in the list, `p.next`. The list is supposed to end with a null pointer. However, because he is unreliable there might be no null pointer at the end, and instead, the last entry points somewhere back to some prior element in the linked list. (You have no idea which one.) You are two write a procedure to determine whether the linked list ends in null or whether it loops back on itself. You do not know the number of elements that are in the list. Also, since the list may be very long, you are not allowed to use any additional array storage. Your algorithm should run in $O(n)$ time, where n is the (unknown) number of elements in the list. (Hint: The solution is not technically complex. It just involves some clever programming.)

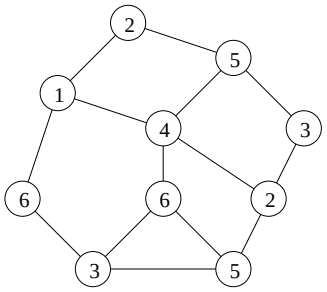
Homework 7: Graphs and Shortest Paths

Handed out Thur, April 23. Due at the start of class Thur, April 30.

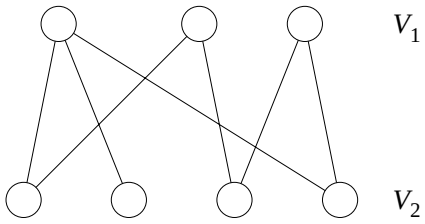
Problem 1. There are some interesting and unexpected applications of RadixSort and/or CountingSort that arise in graph algorithms. This problem illustrates one of them. Suppose that you are given a graph with n vertices, in which each of the vertices is assigned a label in the range from 1 to n. Assume that the graph is represented by an adjacency list $Adj[1..n]$, and the vertex labels are given in an array $label[1..n]$.

Recall that the *distance* between two vertices in an unweighted graph is the number of edges on the shortest path between them. Give an $O(n+e)$ algorithm which determines whether there are any two vertices that have the same label, and are within distance 2 from each other. Your algorithm may use $O(n+e)$ additional working storage. Explain your algorithm and derive its running time. For example, in the figure below (left) the answer is "yes", and in particular there are two vertices with label 6 that are within distance 2 of each other.

Hint: RadixSort is helpful to solve this. For each vertex u and each neighbor v of u, construct a vector consisting of $(u, v, label(v))$. Show that if you sort all the resulting vectors properly by RadixSort then the answer can be determined just by considering *consecutive pairs* in the final sorted list. Which order should you apply CountingSort to the various components for this to work?



Problem 1



Problem 3

Problem 2. Consider the recurrence that arose in the analysis of the recursive algorithm for all-pairs shortest paths:

$$T(m,n) = \begin{cases} 1 & \text{if } m = 1, \\ nT(m-1,n) + 1 & \text{otherwise.} \end{cases}$$

Solve this recurrence (exactly) as a function of n and m. (Hint: Use expansion.)

Problem 3. An undirected graph is *bipartite* if it is possible to partition the vertices into two subsets, V_1 and V_2, such that all the edges go between V_1 and V_2. Describe an $O(n+e)$ algorithm to determine whether a given connected undirected graph is bipartite. Explain your algorithm and derive its running time. (Hint: Use BFS.)

Homework 8: Dynamic Programming and NP-Completeness

Handed out Tue, May 5 (Cinco de Mayo!). Due at the start of class Tue, May 12.

Problem 1. Consider the following code extract from the chain matrix multiplication algorithm.

```
for L = 2 to n do {
    for i = 1 to n-L+1 do {
        j = i + L - 1
        for k = i to j-1 do {
            output "Yabbadabbadoo"
        }
    }
}
```

Show that the number of times that the output statement is executed is

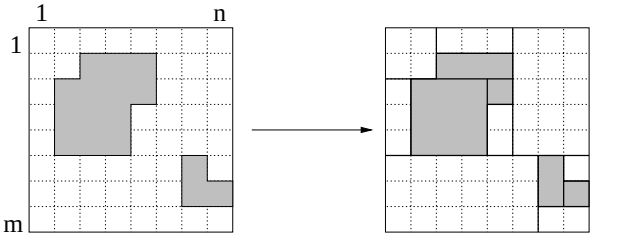
$$T(n) = \frac{n^3 - n}{6}.$$

Hint: The following formula for the *quadratic series* may be of help:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}.$$

This one is messy, so work carefully and check each step.

Problem 2. An important problem in image processing is called *segmentation*. You are given an image, which for simplicity we will think of as being a matrix of "pixels" $A[1..m, 1..n]$ where $A[i, j] = 1$ means that this pixel is white and $A[i, j] = 0$ means that this pixel is black. The segmentation problem is to partition the image into a set of rectangles, such that each rectangle is either all black or all white. An example is shown below.



Input image Segmentation (cost = 12)

To simplify the problem, we will assume that the partition is constructed in a very particular way, called a *guillotine decomposition*. Given any rectangle, you can partition it into two smaller rectangles by cutting completely through the rectangle by either a horizontal or a vertical cutting line. (In particular, once you start cutting, you cannot stop part of the way

through the rectangle or change directions.) The resulting rectangles may be decomposed further by repeating this operation. The *cost* of a guillotine decomposition is the total number of cuts that are made.

Devise a dynamic programming algorithm to determine the minimum cost guillotine segmentation for an image. Here is a suggested formulation. Let $C[i_0, i_1, j_0, j_1]$ denote the minimum cost of decomposing the rectangular subarray $A[i_0..i_1, j_0..j_1]$.

Rather than giving pseudocode, just give the recursive rule for computing $C[i_0, i_1, j_0, j_1]$. Which entry of the C array is the final answer?

Problem 3. The Hamiltonian path problem is known to be NP-complete. (Given an undirected graph, does it contain a path that visits every vertex exactly once?)

(a) Your friend tells you that he has found a clever $\Theta(n^3)$ time algorithm which inputs a graph and two vertices u and v, and determines whether the graph has a Hamiltonian path that starts at u and ends at v.

(b) Your friend tells you that he has found stunningly brilliant $\Theta(n^{42})$ time algorithm which, given a graph, determines whether there is a simple path that visits at least half of the vertices of the graph.

In each case, give a clear mathematical argument for why your friend is either full of bull or else he's a genius. (More likely the former.) (Hint for (b): Make two copies of one graph.)

Practice Questions for the First Midterm

The first midterm will be on Tuesday, Mar 3. The exam will be closed-book and closed-notes, but you will be allowed one cheat sheet (front and back).

Tips: These are practice problems. They do not necessarily reflect the actual length, difficulty, or coverage for the exam. Look back over the homework problems, since there will be more problems of this form. Also review any proofs or constructions given in class. They may reappear with slightly different form.

I have intentionally selected problems here that are different from ones that we have seen in class or in the homeworks. Some of these problems are tricky. You should try working these problems without looking at the solutions. If you get stuck on any problem, try to look at as little of the solution as you can before trying to finish the problem.

Problem 1. Do problems 2.1-1 (without using the limit rule) and 2.1-2 (using the limit rule) on page 31 of CLR. For the second part, you may want to check out page 101 of CLR.

Problem 2. Do problem 3.2-4 on page 52 of CLR.

Problem 3. Use the Master Theorem (as given in class) to derive an asymptotic bound on each of the following recurrences. If the Master Theorem does not apply, then indicate this.

(a) $T(n) = 2T(n/2) + n^3$.

(b) $T(n) = T(9n/10) + n$.

(c) $T(n) = T(\sqrt{n}) + 1$.

(d) $T(n) = 7T(n/3) + n$.

(e) $T(n) = 2T(n/2) + n \lg n$.

Problem 4. Use iteration to solve (exactly) all of the recurrences of the previous problem, which you could not solve by the Master Theorem. In all cases you may make whatever simplifying assumptions you want about n. You may also assume that $T(1) = 1$. (In part (c) it is convenient to assume that n is of the form $2^{(2^k)}$ for some k, and that $T(2) = 1$.)

Problem 5. Do problem 4-2 on page 73 of CLR. Hint: Use the sieve technique.

Spring 1998 Dave Mount

CMSC 251: First Midterm Exam

This exam is closed-book and closed-notes. You may use a one sheet of notes, front and back. Write all answers in the exam booklet. If you have a question, either raise your hand or come to the front of class. Total point value is 100 points. Good luck!

Problem 1. (20 points) Use the simplified Master Theorem (as given in class) to derive an *asymptotic bound* (Θ) for each of the following recurrences. If the simplified Master Theorem does not apply, then indicate this. (You may make any reasonable simplifying assumptions about n, e.g., n is a power of 2). You may assume that $T(1) = 1$ in each case.)

(a) $T(n) = 4T(n/2) + n^2$.
(b) $T(n) = 9T(n/3) + n$.
(c) $T(n) = T(n/2) + 1$.
(d) $T(2^k) = 4T(2^{k-1}) + 2^k$.

Problem 2. (20 points) Consider the following two code sections. For each section, express the number of times that the output statement is executed as a function of n, for $n \geq 1$. Give both an *exact expression* (e.g. $3n^4 + 2n$) and an *asymptotic bound* (e.g. $\Theta(n^4)$). For section (b) you may assume that n is a power of 2.

```
for i = 1 to 2*n do {
    for j = 1 to i do {
        output("hello")
    }
}
```

Problem 2(a)

```
i = n   (You may assume n is a power of 2)
while (i >= 1) do {
    for j = 1 to 2*i do {
        output("goodbye")
    }
    i = i / 2
}
```

Problem 2(b)

Problem 3. (15 points) Use constructive induction to prove that there is a constant c such that for all $n \geq 0$,

$$\sum_{i=0}^{n} i(i-1) = \frac{(n+1)n(n-1)}{c}.$$

What is the value of c revealed by your proof? (Note: Determining c without a proof is not worth many points.)

Problem 4. (15 points) Use the method of iteration to derive an *exact solution* (not asymptotic) to the following recurrence for all $n \geq 0$. (For full credit, your solution should be a simple formula, and should not contain any embedded summations or recurrences.)

$$T(n) = \begin{cases} 1 & \text{if } n = 0, \\ T(n-1) + 3^n & \text{otherwise.} \end{cases}$$

Problem 5. (30 points) We say that a function $f(n)$ is *positive* if $f(n) > 0$ for all $n > 0$. For each of the following statements, indicate whether it is *true* or *false* for all positive functions $f(n)$. If true, then give a proof. If false, then give a specific example of a function for which it is false. You may use either the formal definitions of O, Θ, and Ω, or the limit rules.

(a) $(f(n) + n) \in \Omega(f(n))$.
(b) $f(n) \in \Theta(f(2n))$.
(c) $f(n) \in O(f(n)^2)$.

Practice Questions for the Second Midterm

The second midterm will be on Tuesday, Apr 7. The exam will be closed-book and closed-notes, but you will be allowed two cheat sheets (front and back).

Tips: These are practice problems. They do not necessarily reflect the actual length, difficulty, or coverage for the exam. Look back over the homework problems. Also review any proofs or constructions given in class.

Problem 1. Problems 7.1-1 and 7.1-2 on page 142 of CLR.

Problem 2. Problem 7.1-4 on page 142 of CLR.

Problem 3. Problem 7.3-2 on page 147 of CLR.

Problem 4. Problem 7-2 (parts a and b) on page 152 of CLR for $d = 3$.

Problem 5. Recall the recurrence that arose in the deterministic selection algorithm. $T(1) = 1$ and

$$T(n) = T(n/5) + T(3n/4) + n.$$

In class we used constructive induction to show that $T(n) \leq cn$ for some constant c. (In Lecture 9 on Feb 24.) Consider the following generalization:

$$T(n) = T(\alpha n) + T(\beta n) + n.$$

where α and β are arbitrary positive constants, and $\alpha + \beta < 1$. Show that $T(n) \leq cn$ for some positive constant c (depending on α and β).

Problem 6. Problem 9.1-4 on page 175 of CLR.

Problem 7. Problem 9.2-3 on page 177 of CLR.

Problem 8. Problem 9.3-4 on page 180 of CLR.

Spring 1998 Dave Mount

CMSC 251: Second Midterm Exam

This exam is closed-book and closed-notes. You may use a two sheets of notes, front and back. Write all answers in the exam booklet. If you have a question, either raise your hand or come to the front of class. There are 5 problems and a total point value is 100 points. Good luck!

Problem 1. (15 points) Consider the following input array to HeapSort. Show the heap that results after calling BuildHeap on this array. You may express your answer either as a tree or as an array. (You do *not* need to show intermediate results, but they may be given for partial credit.)

	1	2	3	4	5	6	7
A	3	2	5	7	9	1	6

Figure 1: Problem 1.

Problem 2. (30 points, roughly 8 points each) Short answer questions. NO explanations are required, but a short explanation (2 or 3 sentences) may be given for partial credit.

(a) Which of the following sorting algorithms (as presented in class) is stable? Mergesort, Quicksort, Heapsort, CountingSort.

(b) Recall that a sorting algorithm is *in-place* if it requires no additional array storage (excluding the recursion stack). Which of the following sorting algorithms (as presented in class) is in-place? Mergesort, Quicksort, Heapsort, CountingSort.

(c) You have an array containing n keys, which are in heap order. Suppose that the value of exactly one key is changed (either increased or decreased). As a function of n, how long does it take to restore the heap order. (Do NOT give an algorithm, just the asymptotic running time.)

(d) You are given an array of n characters, where each character is stored as a binary integer that is k bits long. As a function of n and k, how fast can you sort these characters using CountingSort? (Do NOT give an algorithm, just the asymptotic running time.)

Problem 3. (20 points) Consider the recurrence below. $T(1) = 1$, and for $n > 1$

$$T(n) = \frac{1}{n}\left(\sum_{i=1}^{n-1} T(i)\right) + 3n.$$

Give a proof by constructive induction that $T(n) \leq cn$ for all $n \geq 1$, and for some positive constant c. What constraints are there on the value of c?

Problem 4. (15 points) This problem involves proving a lower bound for a problem. Suppose that you are given a *sorted* array $A[1..n]$ of n real numbers, and a real key value X. The

nearest-neighbor problem is to find the element of A that is closest in value to X. For example, for the input below:

$$A[1..8] = \langle 1.4, 2, 3.5, 5, 9, 12.7, 15.5, 19.41 \rangle \qquad X = 8.2,$$

then the nearest neighbor to $X = 8.2$ would be 9. Prove that *any* comparison-based algorithm for the nearest-neighbor problem has a worst-case running time of $\Omega(\log n)$. (Do NOT give the algorithm, just the lower bound proof.)

Problem 5. (20 points) You are given a array of n vectors, where each vector consists of three components, and each component is an integer in the range from 1 to 20.

(a) (10 points) Two vectors are *equal* if they are equal in all components. Thus, $(3, 14, 2)$ is equal to $(3, 14, 2)$ but is *not* equal to $(14, 2, 3)$. Give an $O(n)$ time algorithm which, given such a array of n vectors, either finds two vectors that are equal, or else reports that there are no two such vectors. (Hint: Use RadixSort.)

(b) (10 points) Two vectors are said to be *close* if they are equal in any two components, and differ by at most 1 in the third component. For example $(3, 14, 2)$ is close to $(3, 15, 2)$ and it is close to $(3, 14, 1)$, but it is not close to $(4, 14, 3)$. Give an $O(n)$ time algorithm which, given such a array of n vectors, either finds two vectors that are close, or else reports that there are no two such vectors.

In each case, give a high-level description of your algorithm, and derive its running time.

Practice Questions for the Final Exam

The final exam will be on Monday, May 18, 10:30–12:30. The exam will be closed-book and closed-notes, but you will be allowed three cheat sheets (front and back). These practice problems do not necessarily reflect the actual length, difficulty, or coverage for the exam.

Problem 1. Given two strings, $X = x_1x_2 \ldots x_m$ and $Y = y_1y_2 \ldots y_n$, the *shortest common supersequence* (SCS) is a minimum length string Z such that both X and Y are subsequences of Z. For example, if $X = \langle ABCBABA \rangle$ and $Y = \langle BCAABAB \rangle$, then $Z = \langle ABCAABABA \rangle$ is an SCS of both X and Y.

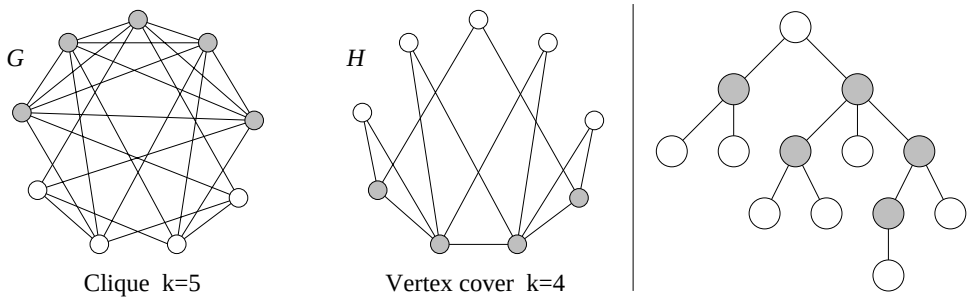
Give a dynamic programming algorithm which given X and Y computes the length of the SCS of X and Y. For full credit, your algorithm should run in $O(mn)$ time. You do not need to determine the actual SCS, just its length. (Hint: Construct a table $c[0..m, 0..n]$ where $c[i, j]$ denotes the length of the SCS for $x_1x_2 \ldots x_i$ and $y_1y_2 \ldots y_j$.)

Problem 2. In class we showed that it is possible to polynomially reduce Hamiltonian cycle to Hamiltonian path, but the reduction involved $O(e)$ calls to the `HamPath()` procedure.

(a) Show that it is possible to do the reduction with at most $n - 1$ calls to to this procedure.

(b) Show that it is possible to do the reduction with exactly one call to this procedure.

Problem 3. Consider the following two problems. The *Clique* problem is: Given a graph G and an integer k, does G contain a subset of V' of k vertices such that all pairs of vertices in V' are adjacent to one another. The *Vertex Cover* (VC) problem is: given a graph G and an integer k, does G contain a subset V' of k vertices such that every edge in G is incident to a vertex in V'. Assuming that Clique is known to be NP-complete, show that VC is also likely to be hard to solve, by showing that Clique $\leq_P$ VC. In other words, given a subroutine for VC, you can use it to solve the Clique problem. (Hint: Try taking the complement of the graph.)



Problem 3 Problem 4

Problem 4. Recall that a *free tree* $G = (V, E)$ is an undirected, connected, acyclic graph. A *vertex cover* is a subset V' of vertices such that every edge in G is incident to at least one vertex in V'. Write an $O(n + e)$ algorithm that computes the size (number of vertices) of the smallest vertex cover for a free tree G. You need only compute the *size* of the smallest vertex cover, not the cover itself. (Hint: This can be done by either dynamic programming or depth-first search of the tree.) An example is shown above right of a free tree with a vertex cover of size 5.

Spring 1998 Dave Mount

CMSC 251: Final Exam

This exam is closed-book and closed-notes. You may use three sheets of notes, front and back. Write all answers in the exam booklet. Unless otherwise stated, you may assume any results that we have given in class. If you have a question, either raise your hand or come to the front of class. There are 5 problems and the total point value is 100 points. Good luck!

Problem 1. (10 points) Consider the two strings $X = AAB$ and $Y = BAB$. Show the table generated by the Longest Common Subsequence algorithm when run on these two strings. In your table, you only need to indicate the subsequence lengths (the $c[i, j]$ matrix), NOT the matrix used to reconstruct the sequence (the $b[i, j]$ matrix).

Problem 2. (35 points, roughly 5 points each) Short answer questions. Except where noted, no explanations are required, but a short explanation (2 or 3 sentences) may be given for partial credit.

(a) List the following functions in *increasing* order of asymptotic growth rate. If two functions are of the same order, then indicate this.

$$n^2 \log^2 n, \qquad n^2 \log^3 \sqrt{n}, \qquad 4^{(\lg n + \lg\lg n)}$$

(b) Give an asymptotic (Θ) solution to the following recurrence: $T(n) = 4T(n/2) + n^2$.

(c) Give an exact solution to the following summation as a function of $n \geq 0$. Note that the sum runs from n to $2n$.

$$T(n) = \sum_{i=n}^{2n} 2^{(1+\lg i)}$$

(Hint: You only need the standard formulas for the arithmetic and/or the geometric series.)

(d) Recall that a graph is *planar* if it can be drawn in the plane so that no two edges cross over one another. Suppose you want to store a very large planar graph. Which representation would be better: an adjacency list or an adjacency matrix? Explain briefly.

(e) A digraph is *strongly connected* if there exists a path between every pair of vertices. Consider a strongly connected digraph G with n vertices and e edges. As a function of n what is the minimum number of edges that G can have?

(f) In the Floyd-Warshall algorithm, we assumed that there may be negative cost edges, but no negative cost cycles. What is the problem with negative cost cycles?

(g) Suppose that you have two decision problems, A and B. You know that problem A is NP-complete, and B is solvable in $O(n^2 \log^4 n)$ time, and you know that $B \leq_P A$. What does this imply? (Select one.)

(i) P = NP and all NP problems are solvable in $O(n^3)$ time.
(ii) P = NP but some NP problems may require more than $O(n^3)$ time to solve.
(iii) P $\neq$ NP.
(iv) We still do not know whether P = NP.

Problem 3. (15 points) Suppose that you are given k sorted linked lists, each of length m. These are given as an array $L[1..k]$, where $L[i]$ points to the head of the ith linked list. You want to merge them together into one sorted linked list of total length km. Assume that you have access to a procedure `Merge(A, B, C)`, which is given two sorted linked lists A and B and returns a pointer to a linked list C which contains the merged sorted list. The procedure takes time proportional to the sum of the

lengths of A and B. Using this procedure, give an algorithm that merges all k lists into one sorted list in time $O(mk \log k)$. You may assume that k is a power of 2. (You may NOT use a heap or any other tree data structure to help you solve this problem.) Explain your algorithm's correctness and derive its running time.

Problem 4. (20 points) Recall the 3-coloring problem (3COL): Given a graph G, is it possible to label the vertices of G with 3 different values $\{1, 2, 3\}$, such that every pair of adjacent vertices have different labels. 3COL is known to be NP-complete. (You should not assume that any other problem is known to be NP-complete.) In each case, give a clear argument as to why your friend is probably on drugs. (Either the problem probably cannot be solved in polynomial time, or else there is a much simpler solution.)

(a) Your friend claims that he has a $\Theta(n^4)$ algorithm that will determine whether any graph with n vertices is 3-colorable, provided that n is a power of 2.

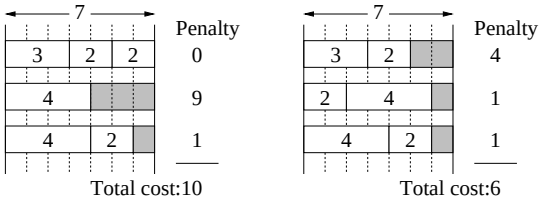
(b) Your friend claims that he has a $\Theta(n^6)$ algorithm that will determine whether any graph is 4-colorable.

(c) Your friend claims that he has a $\Theta(n^8)$ algorithm that will determine whether any bipartite graph is 3-colorable.

Problem 5. (20 points) An important problem for typesetting programs like WordPerfect is packing words of text into lines. This problem will consider a highly simplified version of this. You are given a sequence of n words, where the ith word consists of $w[i]$ characters. You are also given the length L of a line in characters. The objective is to partition the sequence of words into subsequences, each of which will constitute a single line of the final text. To simplify things, we will ignore spaces between words.

Obviously, you cannot change the order of the words. The goal is to fill each line as nearly full as possible, but not exceeding the line length L. You may assume that no word is longer than L. If you leave b blank characters on a line, then you are charged a penalty of b^2. The *total cost* of the final partition is the sum of the penalties over all the lines.

For example, consider a line length of $L = 7$, and 6 words whose widths are $w[1..6] = \langle 3, 2, 2, 4, 4, 2 \rangle$. The figure below shows two possible ways of breaking these words into lines, one of cost 10 and the other of cost 6. The shaded blocks indicate blanks. Each line is charged a penalty by squaring the number of blanks (e.g., 3 blanks carries a penalty of 9). The partition on the right is the best possible.



We want to design a dynamic programming algorithm which given integer L and the array of word widths $w[1..n]$, computes the cost of the minimum cost partition. Here is a suggested formulation. For $1 \leq i \leq j \leq n$, let $C[i, j]$ denote the cost of the minimum cost partition for the subsequence of words $w[i..j]$.

(a) For the basis case, express the value of $C[i, i]$ for each i, as a function of $w[i]$ and L.

(b) What entry of the C matrix is the final answer, that is, the cost of the optimum partition?

(c) Give a recursive rule, which expresses $C[i, j]$ as a function of smaller subsequences of words. (Hint: There are two decisions: (1) should you split this subsequence? and (2) if you do split, then where should you make the split?)