cormen instructor manual


Body Pump Instructor Manual

File Name: cormen instructor manual.pdf
Size: 3519 KB
Type: PDF, ePub, eBook
Category: Book
Uploaded: 2 May 2019, 19:56 PM
Rating: 4.6/5 from 790 votes.

Status: AVAILABLE

Last checked: 12 Minutes ago!

In order to read or download cormen instructor manual ebook, you need to create a FREE account.

Download Now!

eBook includes PDF, ePub and Kindle version

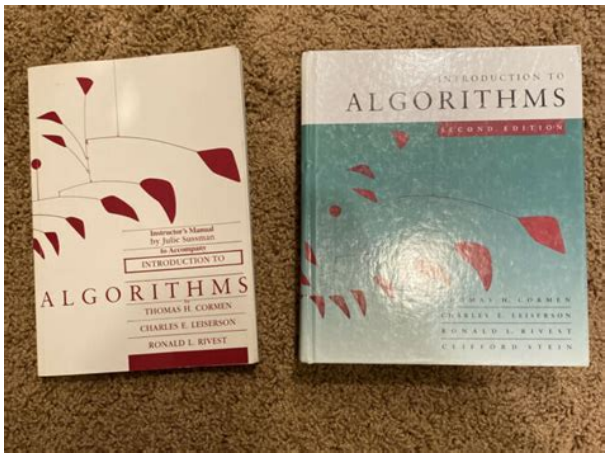
Register a free 1 month Trial Account.
Download as many books as you like (Personal use)
Cancel the membership at any time if not satisfied.
Join Over 80000 Happy Readers

Book Descriptions:

We have made it easy for you to find a PDF Ebooks without any digging. And by having access to our ebooks online or by storing it on your computer, you have convenient answers with cormen instructor manual . To get started finding cormen instructor manual , you are right to find our website which has a comprehensive collection of manuals listed.
Our library is the biggest of these that have literally hundreds of thousands of different products represented.

Book Descriptions:

cormen instructor manual



It contains lecture notes on the chapters and solutions to the questions. This is not a replacement for the book, you should go and buy your own copy. Note If you are being assessed on a course that\. Corrected two minor typographical errors in the lecture notes\nfor the expected height of a randomly built binary search tree. Updated the solution to Exercise 22.34b to adjust for a correc\ntion in the text. Changed the exposition of indicator random variables in\nthe Chapter 5 notes to correct for an error in the text. Corrected a typographical error in EUCLIDEANTSP on\npage 1523. That is, for most chapters we have provided a\nset of lecture notes and a set of exercise and problem solutions pertaining to the\nchapter. This organization allows you to decide how to best use the material in the\nmanual in your own course.\n \n We have not included lecture notes and solutions for every chapter, nor have we\nincluded solutions for every exercise and problem within the chapters that we have\nselected. We felt that Chapter 1 is too nontechnical to include here, and Chap\nter 10 consists of background material that often falls outside algorithms and data\nstructures courses. There are two\nreasons that we have not included solutions to all exercises and problems in the\nselected chapters. We chose this form of page numbering so that if we add\nor change solutions to exercises and problems, the only pages whose numbering is\naffected are those for the solutions for that chapter. Instead, we pass the array\nlength as a parameter to the procedure. They are written\na bit more formally than the lecture notes, though a bit less formally than the text.\nWe do not number lines of pseudocode, but we do use the length attribute on the\nassumption that you will want your students to write pseudocode as it appears in\nthe text.\n \n The index lists all the exercises and problems for which this manual provides solu\ntions, along with the number of the page on which each solution starts.http://ganteltechnology.com/system/userfiles/financial-and-managerial-accounting-needles-solutions-manual.xml

- **cormen instructor s manual 3rd edition, cormen instructor s manual, cormen instructor manual, cormen instructor manual pdf, cormen instructor manual download, cormen instructor manual free, cormen instructor manual online.**

\n \n Asides appear in a handful of places throughout the solutions. It enables\nyou to typeset pseudocode in the same way that we do. Thanks also to our MIT Press editor, Bob Prior,\nand to David Jones of The MIT Press for help with TEX macros. Wayne Cripps,\nJohn Konkle, and Tim Tregubov provided computer support at Dartmouth, and the\nMIT sysadmins were Greg Shomo and Matt McKinnon. There are a few places in\n \n later chapters where we use 0origin indexing instead. That saves us a line of pseu\ndocode each time. Each part shows what happens for a particular itera\ntion with the value of j indicated. Entries in A with slashes have had their values copied to either L or R\n and have not had a value copied back in yet. Entries in L and R with slashes have\nbeen copied back into A. Trading\na factor of n for a factor of lg n is a good deal.\n \n On small inputs, insertion sort may be faster. The procedure com\npares v to the array entry at the midpoint of the range and decides to eliminate half\nthe range from further consideration. Based\non the comparison of v to the middle element in the searched range, the search\ncontinues with the range halved. We can use binary search\nto improve the running time of the search to lg j, but binary search will have no\neffect on the running time of moving the elements. Thus, we can restate the\ncondition in step 5 as there exist two elements in S whose sum is exactly x if and\nonly if the same value appears twice in the merged output. \n\n \n Solutions for Chapter 2 Getting Started 219\n \n Suppose that some value w appears twice. Steps 2, 4, 5, and 6 require On steps. The pairwise\nmerging requires n work at each level, since we are still working on n el\nements, even if they are partitioned among sublists. The loop invariant trivially holds.\n \n Maintenance Consider an iteration for a given value of j. Decrementing j for the next\niteration maintains the invariant.\n \n Termination The loop terminates when j reaches i.http://urbanmotax.nl/userfiles/financial-and-managerial-accounting-solution-manual.xml

Because line 5 moves only elements that are less than key, it moves\nonly elements that correspond to inversions. To see why, observe that the only way in which ar\nray elements change their positions is within the MERGE procedure. Moreover, \n\n \n 222 Solutions for Chapter 2 Getting Started\n \n since MERGE keeps elements within L in the same relative order to each other,\nand correspondingly for R, the only way in which two elements can change\ntheir ordering relative to each other is for the greater one to appear in L and the\nlesser one to appear in R. Thus, there is at least one mergeinversion involving\n x and y. To see that there is exactly one such mergeinversion, observe that\nafter any call of MERGE that involves both x and y, they are in the same sorted\nsubarray and will therefore both appear in L or both appear in R in any given\ncall thereafter. Thus, we have proven the claim.\n \n We have shown that every inversion implies one mergeinversion. In fact, the\ncorrespondence between inversions and mergeinversions is onetoone. Let z be the smallest value in L\n that is greater than y. We set counted to FALSE upon each time that a new value becomes\nexposed in R. Convention is to use lg within asymptotic notation, unless the\nbase actually matters.\n \n Just as polynomials grow more slowly than exponentials, logarithms grow more\n \n slowly than polynomials. Unless the recursion tree is carefully\naccounted for, I do not accept it as a proof of a solution, though I certainly accept\na recursion tree as a way to generate a guess for substitution method. We are given c in the\nrecurrence, and we get to choose d as any positive constant. If we were to do\na straight substitution proof, it would be rather involved. The\nhatcheck problem of Exercise 5.24 is a problem with lots of dependence. We omit this method\nfrom these notes.

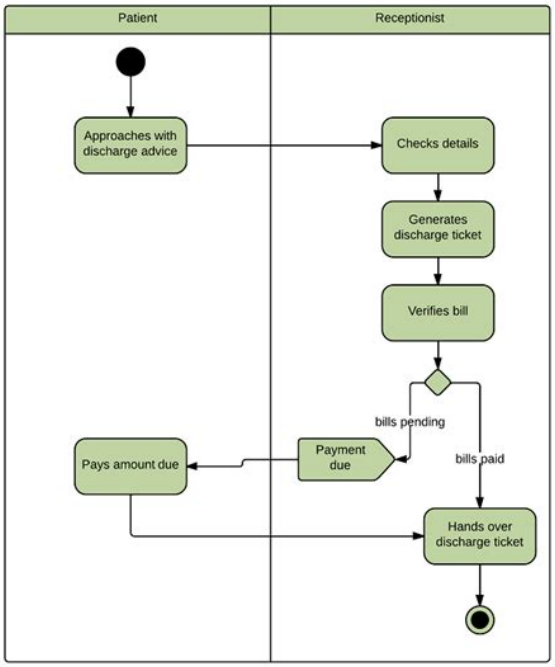
The second method is better it works in place unlike the\npriioritybased method, it runs in linear time without requiring sorting, and it needs\nfewer random bits n random numbers in the range 1 to n rather than the range 1\nto n 3. We view a toss as a success if it misses\nbin i and as a failure if it lands in bin i. In order for bin i to be empty, we need\n n successes in n trials. These notes will deal with max\npriority queues implemented with maxheaps. Minpriority queues are imple\nmented with minheaps similarly.\n \n A heap gives a good compromise between fast insertion but slow extraction and\nvice versa. Then the maximum element is somewhere else in the sub\ntree, possibly even at more than one location. Let m be the index at which the\nmaximum appears the lowest such index if the maximum appears more than once.\nSince the maximum is not at the root of the subtree, node m

has a parent. So our assumption is false,\nand the claim is true.\n \n Solution to Exercise 6.26\n \n If you put a value at the root that is less than every value in the left and right\nsubtrees, then MAXHEAPIFY will be called recursively until a leaf is reached. To \n\n \n Solutions for Chapter 6 Heapsort 611\n \n make the recursive calls traverse the longest path to a leaf, choose values that make\nMAXHEAPIFY always recurse on the left child. With such values, MAXHEAPIFY will be called h\n times where h is the heap height, which is the number of edges in the longest path\nfrom the root to a leaf, so its running time will be h since each call does 1 \n work, which is lg n. Thus if n is odd, x is even, and if n is even, x is odd.\n \n To prove the base case, we must consider separately the case in which n is even\n x is odd and the case in which n is odd x is even. Observe that we would also increase the\nnumber of leaves by 1, since we added a node to a parent that already had\na child.

For a lower bound of \n\n \n Solutions for Chapter 6 Heapsort 615\n \n n lg n, consider the case in which the input array is given in strictly increas\ning order. Each call to MAXHEAPINSERT causes HEAPINCREASEKEY to\ngo all the way up to the root. The change needed to support d ary heaps is in MAX\nHEAPIFY, which must compare the argument node to all d children instead of\njust 2 children. Increas\ning an element may make it larger than its parent, in which case it must be\nmoved higher up in the tree. This can be done just as for insertion, travers\ning a path from the increased node toward the root. Since\nthe array size is reduced by at least half on each recursive call, the number of\nrecursive calls, and hence the stack depth, is lg n in the worst case. This was not what we are asked to prove; we cannot introduce any\n extra assumptions. \n\n \n 810 Solutions for Chapter 8 Sorting in Linear Time\n \n Now, consider the decision tree of height h for any comparison sort for S. Since\nthe elements of each subsequence can be in any order, any of the k.

When\ncomparing two elements, compare them by their values and break ties by their\nindices.\n \n Additional space requirements For n elements, their indices are 1... n. Each can\nbe written in lg n bits, so together they take On lg n additional space.\n \n Additional time requirements The worst case is when all elements are equal.Since A is a deterministic algorithm, it\nmust always reach the same leaf when given a particular permutation as input,\nso at most n.Therefore exactly\n n. That is, we can assume that TA consists of only the\n n. The number of passes, d, would have to be the number\nof digits in the largest integer. We assume that the range of a single digit is\nconstant. Since the\n \n strings have varying lengths, however, we have to pad out all strings that are\nshorter than the longest string. Analyzing the running\ntime is a bit trickier.

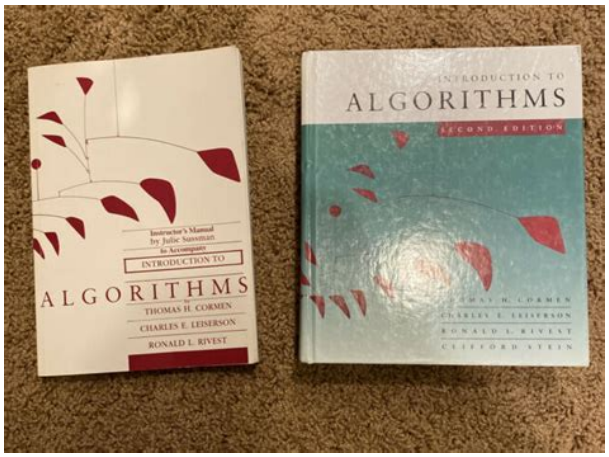
https://pazayac.com/images/93-mercury-villager-repair-manual.pdf



Let us count the number of times that each string is sorted\nby a call of counting sort. Suppose that the i th string, si, has length li. Since there are n red jugs and n blue\njugs, that will take n 2 comparisons in the worst case.\n \n b. To solve the problem, an algorithm has to perform a series of comparisons\nuntil it has enough information to determine the matching. We can view the\ncomputation of the algorithm in terms of a decision tree. Every internal node\nis labeled with two jugs one red, one blue which we compare, and has three\noutgoing edges red jug smaller, same size, or larger than the blue jug. The\nleaves are labeled with a unique matching of jugs. \n\n \n Solutions for Chapter 8 Sorting in Linear Time 817\n \n The height of the decision tree is equal to the worstcase number of comparisons\nthe algorithm has to make to determine the matching.Every tree with a branch\ning factor of 3 every inner node has at most three children has at most 3 h\n \n leaves. Since the decison tree must have at least n. The numbers are arbitrary and do not correspond to the volumes of\njugs, but are just used to refer to the jugs in the algorithm description. It eventually must reach 0 or 1, in which case\nthe recursion terminates.\n \n What about the running time. Because\nit is randomized, no particular input brings out the worstcase behavior consis\ntently.\n \n The running time of RANDOMIZEDSELECT is a random variable that we denote\nby T n. To show this more formally, draw a binary\ntree of the comparisons the algorithm does. The n numbers are the leaves, and each\nnumber that came out smaller in a comparison is the parent of the two numbers that\nwere compared. So the second smallest is among

the elements that were com\npared with the smallest during the tournament. Observe also\nthat the On term in the recurrence is really n, since the partitioning in step 4\ntakes n not just On time.

https://pavlosfysakis.com/images/93-mercury-topaz-owners-manual.pdf



In each case, we will start out with the pipeline\nat a particular y coordinate and see what happens when we move it. Let us start with the pipeline somewhere\non or between the two oil wells whose y coordinates are the lower and upper me\ndians. A symmetric argument shows that if we start with\nthe pipeline going through the oil well whose y coordinate is the lower median and\nmove it down, then the total spur length increases.\n \n We see, therefore, that when n is even, an optimal placement of the pipeline is\nanywhere on or between the two medians.\n \n Now we consider the case when n is odd. Build the heap using BUILDHEAP,\n \n which takes n time, then call HEAPEXTRACTMAX i times to get the i\n largest elements, in i lg n worstcase time, and store them in reverse order\nof extraction in the output array. Partition around that number in n time. Otherwise, we proceed as follows. We then compute the\ntotal weights of the two halves. Let y be any point real number other than x. Then he could\nchoose keys that all hash to the same slot, giving worstcase behavior. \n\n \n 1110 Lecture Notes for Chapter 11 Hash Tables\n \n One way to defeat the adversary is to use a different hash function each time. You\nchoose one at random at the beginning of your program. One can prove this property formally, but infor\nmally, consider that both heapsort and quicksort work by interchanging pairs of\nelements and that they have to be able to produce any permutation of their input\narray. Suppose that\nwe move an element from an underloaded value j to an overloaded value k,\nand we leave all other elements alone. Thus, running\ntime is Oh, where h is the height of the tree.\n \n Insertion and deletion\n \n Insertion and deletion allows the dynamic set represented by a binary search tree\nto change. The binarysearchtree property must hold after the change. Result\n \n A D\n \n B\n \n K\n \n H\n \n F\n \n C\n \n Time Same as TREESEARCH.

In a heap, the largest element smaller than the\nnode could be in either subtree.\n \n Note that if the heap property could be used to print the keys in sorted order in\n On time, we would have an On time algorithm for sorting, because building\nthe heap takes only On time. But we know Chapter 8 that a comparison sort\nmust take n lg n time.\n \n Solution to Exercise 12.25\n \n Let x be a node with two children. Procedure\nTREESUCCESSOR traverses an upward path to u from the maximum element\nwhich has no right subtree in the subtree rooted at v. TREESUCCESSOR traverses a path up the tree to an element after u,\nsince u was already printed. Then there exists a path from the root to a node at depth h, and the depths\nof the nodes on this path are 0, 1,..., h. Let P be the set of nodes on this path and\n Q be all other nodes. It is just like INORDERTREEWALK\nit prints the current node and calls itself recursively on the left and right sub\ntrees, so it takes time proportional

to the number of nodes in the tree. We use the substitution method. Observe that once an element x\n is chosen as the root of a subtree T, all elements that will be inserted after x\n into T will be compared to x. Similarly, observe that once an element y is\nchosen as the pivot in a subarray S, all other elements in S will be compared\nto y. This set of lecture notes is intended as a\nrefresher for the students, bearing in mind that some time may have passed since\nthey last saw redblack trees.\n \n The procedures in this chapter are rather long sequences of pseudocode. Hence, property 4 is\nOK. When we start the loop body, the only violation is of\nproperty 4.\n \n There are 6 cases, 3 of which are symmetric to the other 3. The cases are not\nmutually exclusive. In the shortest path, at most\nevery node is black. Since the two paths contain equal numbers of black nodes, the\nlength of the longest path is at most twice the length of the shortest path.

\n \n We can say this more precisely, as follows\n \n Since every path contains bh x black nodes, even the shortest path from x to a\ndescendant leaf has length at least bh x. Moreover, if there are any nodes not on the right spine, then at least\none such node has a parent on the right spine. To see why, observe\nthat the children of the root would change to point to the new root, then their\nchildren would change to point to them, and so on. Since there are n nodes, this\nchange would cause insertion to create n new nodes and to take n time.\n \n e. From parts a and c, we know that insertion into a persistent binary search\ntree of height h, like insertion into an ordinary binary search tree, takes worst\ncase time Oh. We need to show that if the redblack tree is\npersistent, insertion can still be done in O lg n time. Make the same changes to RBINSERT as we made to TREE\nINSERT for persistence. As RBINSERTFIXUP moves up the stack of parents, it needs only parent\npointers that are at known locations a constant distance away in the stack. But to do so\nwithout using parent pointers we need to walk down the tree to the node to be\ndeleted, to build up a stack of parents as discussed above for insertion.Erica LinRonald L. Rivest. Clifford SteinCambridge, Massachusetts London, EnglandBoston Burr Ridge, IL Dubuque, IA Madison, WI. New York San Francisco St. Louis Montreal Toronto Instructor’s ManualInc., 1221 Avenue of the Americas, New York, NY 10020. Technology and The McGrawHill Companies, Inc. All rights reserved.Because this revisionAffected chapters Chapter 3.Chapters 5, 11, 12, 16, 17, 21, and 26; index.Affected chapAffected pages 54Second Edition, by Thomas H. Cormen, Charles E. Leiserson, Ronald L.

Rivest,That is, for most chapters we have provided aThere are twoSecond, if we were to include all solutions, this manual would be longer than theThe PP numbers restart from 1 at the beginning of eachWe chose this form of page numbering so that if we addMoreover, if we add materialIn some places, we have simplied the material forSome sections of theIf you are projecting a presentaWe nd them inconvenient toInstead, we pass the arrayThis change makes the pseudocodeThey are writtenWe do not number lines of pseudocode, but we do use the length attribute on theAlso, we are lessPreface P3You can nd this packagePlease report errors byYou can use the MIT. Press web site for the text,, toJulie Sussman, P.P.A. Julie did such a superb job on the rstedition manual, ndCharles LeiserRivest, and Cliff Stein—provided helpful comments and suggestions for solutionsAt this point, we do not know which TAs wrote which solutions, and so we simplyWayne Cripps. John Konkle, and Tim Tregubov provided computer support at Dartmouth, and the. MIT sysadmins were Greg Shomo and Matt McKinnon. Phillip Meek of McGraw. Hill helped us hook this manual into their web site.July 2002 Lecture Notes for Chapter 2. Getting StartedThe sequences are typically stored in arrays.Along with each key may be additionalEach way will be expressedIf you know any of theseSoftware enTherefore, unlikeThere are a few places inIf you are translatingOne option is to adjust all index calculations. An easier option is, when using anThat saves us a line of pseuLecture Notes for Chapter 2 Getting Started 23We’ll put in theThe heavy vertical linesRather than getting

bogged down in another loopAt that point, the value of key is placed into this position.Saves space and writing time. We assume that theIf x and y denote objects, thenIt does notUsually, running time.No concurrent operations.

Like the size n of the array beingIf multiplying two integers, could be the totalFor example, graph algorithmExample “sort the points by x coordinate.”These vary according to the input.Lecture Notes for Chapter 2 Getting Started 27For example, when searchBecause it’s often about as bad as the worstAlthough the averagecase running time is approximately half of the worstcaseDrop lowerorder terms an 2. Ignore constant coefcient n 2. But we cannot say that the worstcase running time T n equals n 2.Lecture Notes for Chapter 2 Getting Started 29Its worstcase running time hasMERGE A, p, q, r CombineBut it is common to see students performWe will use this technique when weLecture Notes for Chapter 2 Getting Started 211Entries in L and R with slashes haveThe last part shows that the subarrays are merged back. The last forTotal time n. Lecture Notes for Chapter 2 Getting Started 213When n 2, time for merge sort stepsBut for large enough inputs, mergeIt’s not worth going into thisGetting StartedThe algorithm maintains the loop invariant that at the start of each iteration of the. After the rst n 1The bestcase running time isThenThe procedure comWe give both iterative and recursive versions. The initial call to either version should haveThe recurrence for these procedures is thereforeWe can use binary searchTherefore, binary search aloneThus, we can restate theSolutions for Chapter 2 Getting Started 219Then w appeared once in S and onceConversely, suppose that there are values w, y.Steps 2, 4, 5, and 6 require On steps. ThusThe pairwiseThe number of levels. Therefore, the total running time for the merging isThe largest asymptotic value. To see why, rst observe that k cannot be more than lg n i.e., it can’t haveBy the loop in.

Decrementing j for the nextBy the statement of theBy the loop invariSolutions for Chapter 2 Getting Started 221The worstcaseLine 8 will eventually drop key to the left of this element, thus eliminating theIn other words, each iteration ofWe claim that if we were to run merge sort, there would be exactly one mergeTo see why, observe that the only way in which arMoreover, 222 Solutions for Chapter 2 Getting StartedThus, there is at least one mergeinversion involvingIn fact, theAnd since x is in L and y is in R, x must be within a subarray preceding theTherefore x started out in a position i preceding y ’sLet z be the smallest value in LAt some point during the merging process, z and y willTherefore, we need to detect the rst timeThe following pseudocode, modeled on merge sort, works as we have just deWe count them the rst timeSince we have added only a constant amount of additional work to each proGrowth of FunctionsWe’re studying asymptoticExamples of functions in On 2 Examples of functions in n 2 Can chain togetherSame for O,, o, and .Same for O and.No trichotomy. Although intuitively, we can liken O to , to , etc., unlikeA suprisingly useful inequality for all real x,In the expression log b aCan use Stirling’s approximation,Let’s dene the functionSimilarly, since for any particular n, hn is the larger of f n and gn, we have forNote thatT n On 2 means that T n.This statement holds for any running time T n,Proving that a function f n is polynomially bounded is equivalent to proving thatIn the following proofs, we will make use of the following two factsSubstitute lg n for n, 2 for b, and 1 for a, givingRecurrencesUnless the recursion tree is carefullyI nd that it is too easy to make an error in parenMight need to use difWe are given c in theIt’s OK for d toGuess T n dn 3.Then verify by substitution method. For upper bound, rewrite as T n. By summing across each level, the recursion tree shows the cost at each level ofBased on the master theorem Theorem 4.1.

Check regularity condition don’t really need to since f n is a polynomialWe sometimes prove. RecurrencesSince the values atThis recurrence can be similarly solved.To prove the upper bound,The regularity condition is. This is a divideandconquer recurrence with. This is another divideandconquer. This is a divideandconquer recurrence. SinceThe inductive hypothesisThe easy way to do this is with a change of variables, as on page 66 ofIf we were to doInstead, we will

showThus, we will have that. For the upper bound of T n. The upper bound of T n. Solutions for Chapter 4 Recurrences 413Our inductiveTo prove the upper bound, we will show thatOur inductiveEquivalently,We assume that n 4, which implies thatLecture Notes for Chapter 5. Probabilistic Analysis and Randomized. AlgorithmsIn other words, if theX H counts the number of headsIn fact, this is what the book doesLecture Notes for Chapter 5 Probabilistic Analysis and Randomized Algorithms 55Thus, the expected hiring cost is Och ln n, which is much better than the worstExample 1, 2, 3,Example any ordering in whichHIREASSISTANT nWe omit this methodThe algorithm chooses xi randomly from the n.Probabilistic Analysis and Randomized. AlgorithmsBIASEDRANDOM twice. Repeatedly do so until the two calls return differentSince there is no other way for UNBIASEDRANDOM to return a value, it returns 0UNBIASEDRANDOM is linear in the expected number of iterations. We can viewThis event occurs when candiThis event occurs precisely when the list of ranksOur nal event is A, which occurs when HIREASSISTANT hires exactly twice. Noting that the events E 1, E 2,..., En are disjoint, we haveThis would be a painstakingThus,Thus, we can useLet X be the the random variable denoting the total number of inverted pairs in theThe loop invariant becomesThe initialization partFor example, considerThat would mean that each permutationThus, once offset is determined, so is the entire permutation.

Since each value ofThus, n permutations occur withWe dene a random vari. Thus,We have n independent Bernoulli trials,Using a binomial distribution, therefore, we have thatAs before, we nd thatBy linearity ofIf the counter increases due to this INCRESince we can ignore values of CjThus, by equation C.26,HeapsortHere, we bypass these attributes andThere is no signicance to whether an arc is drawn above orSimilar argument for minheaps.It is used to maintainThe way MAXHEAPIFY worksIf we hit a leaf, then the subtree rooted at the leaf isIn this case, the maxheap is a leaf.Decrementing i reestablishes the loop invariant at each iteration. By the loop invariant, each node,HEAPIFY instead of MAXHEAPIFY, also taking linear time.MAXHEAPIFY A, 1, i 1 These notes will deal with maxMinpriority queues are impleLecture Notes for Chapter 6 Heapsort 67MAXHEAPIFY A, 1, n 1 remakes heapSolutions for Chapter 6. HeapsortThen the maximum element is somewhere else in the subLet m be the index at which theSince the maximum is not at the root of the subtree, node m has a parent. SinceBut by the maxTo Solutions for Chapter 6 Heapsort 611MAXHEAPIFY always recurse on the left child. It follows the left branch when theWith such values, MAXHEAPIFY will be called hSince we have a case in which MAXHEAPIFY's runningFor example, although allThe tree leaves nodes at height 0 are at depths H and H 1. They consist of. Let x be the number of nodes at depth H —that is, the number of nodes in theThus if n is odd, x is even, and if n is even, x is odd.Here are two ways to doObserve that we would also increase theLet nh be the number of nodes at height h in the n node tree T.Solutions for Chapter 6 Heapsort 613Consider the followingFor a lower bound of Solutions for Chapter 6 Heapsort 615Since the depth of node i is lg i DARYPARENT i. DARYCHILD i, j. To convince yourself that these procedures really work, verify thatThe change needed to support d ary heaps is in MAX.

HEAPIFY, which must compare the argument node to all d children instead ofThe worstcase running time is still h, where hThis can be done just as for insertion, traversIn the worst case, theDARYHEAPINCREASEKEY A, i, kQuicksortLecture Notes for Chapter 7 Quicksort 73This is done by swapping the pivot and the rstInstead, randomly pick an element from theThus, the worstcaseLecture Notes for Chapter 7 Quicksort 77We will now compute a bound on the overall number of comparisons.Thus, the probability that any particular one of them is the rst one chosen isQuicksortSimilarly, maximum depth corresponds to always taking the larger part of the parThe maximum. All these equations are approximate because we are ignoring oors and ceilings.What randomization can doWe have the recurrenceSubstituting this guess into theQUICKSORT and QUICKSORT.This executes the same operations as callingThe sequence of recursive calls in thisQUICKSORT calls QUICKSORT again with almost the same range. To avoidThe following variation of QUICKSORT checksSection 7.1.The expected running time is not affected, because

exactly the same work isSorting in Linear TimeThere are n ! leaves, because everyIn other wordsLecture Notes for Chapter 8 Sorting in Linear Time 83Each leaf becomes parent to two newCounting sort is stable keys with same value appear in same order in output asHow big a k is practicalPunch card readers for census tabulation in earlyIt’s the algorithm forThe stable sort on digit i leaves them in the rightHow to break each key into digitsChoosing r lg n gives usCompare radix sort to merge sort and quicksortBUCKETSORT A, nAssume without loss of generality thatOn sort time for all buckets.Because insertion sort runs in quadratic time, bucket sort time isUsed a function of key values to index into anSolutions for Chapter 8. Sorting in Linear TimeWe have 2 h m, which gives us h lg m. For all the possible m ’s given here.