Problems on Algorithms and Data Structures

Søren Dahlgaard
Revised by Rasmus Pagh

February 19, 2024

1 Sorting and recurrence analysis

Motivation This assignment is about two subjects: solving recurrences and sorting. Solving recurrences is a very important tool for the analysis of algorithms, and it is important that you get familiarized with the methods in Chapter 4 of CLRS. Likewise, sorting is a very fundamental algorithmic problem and is used as a basic step in many algorithms. It is important to be able to recognize which sorting algorithms to use and why.

To solve the assignment, it may be useful to look both at the slides (available on Absalon) and the course book.

1.1 Recurrence analysis

The CPS Brewing Company (CPSBC) has a wide selection of beers, and every time they introduce a new one they always sell it for cheap. The next year they up the price a bit, and again the following year, and so on. In this devious pricing scheme the price of a beer in 2022 may depend on the price of that same beer in the previous years. Your goal is to figure out the *asymptotic behaviour* of the price of different beers. We denote the price of a beer the year it is introduced and the following years by $p(1), p(2), \ldots$, respectively. The price for any beer the first two years are $p(1) = 1$ and $p(2) = 2$. The CPC (Chief Price Consultant) of CPSBC then decides on a recursive formula (called a *pricing scheme*) – for instance the price of CPSBC's world famous USB-Porter is given by $p(n) = 2p(n/2) + n$. We know from CLRS that this asymptotically behaves as $p(n) = O(n \log n)$.

Your task is to give an asymptotic upper bound for $p(n)$ for each of the following pricing schemes. You should make your bound as tight as possible.

Task 1: For the following pricing schemes, use the master method.

1. $p(n) = 8p(n/2) + n^2$.
2. $p(n) = 8p(n/4) + n^3$.
3. $p(n) = 10p(n/9) + n \log_2 n$.

Task 2: For the following pricing schemes, use the substitution method. You may ignore the induction start and only show the induction step. When showing the induction step, you may assume that $n/2$ and $n/3$ are integers in the first part and that $\sqrt{n}$ is an integer in the second part. Be careful to avoid the pitfalls of the substitution method (see CLRS section 4.3).

Additionally, you have to draw a recursion tree down to at least four levels for both of the recurrences. Your guess for the substitution method needs to be derived from the recursion tree.

1. $p(n) = p(n/2) + p(n/3) + n$.
2. $p(n) = \sqrt{n} \cdot p(\sqrt{n}) + \sqrt{n}$ *Hint: This one might be tricky. Take a close look at the section on subtracting lower order terms in CLRS.*

1.2 Sorting

The CPS Brewing Company is also very interested in sorting their n beers. The CEO of the company has heard that quicksort performs extremely well in practice, but he does not like that the worst-case running time is $\Theta(n^2)$. He has heard that there is an $O(n \log n)$ alternative called introspective sort (or introsort) [1], which combines quicksort, heapsort and insertion sort. To his knowledge this algorithm is used in libraries such as the GNU standard C++ library[1] and the Microsoft .NET Framework Class Library[2].

Introsort Introsort works as follows: Given an array A and a subrange $A[i, j]$, the algorithm maintains a counter of the recursion depth and performs quicksort recursively on this range until the counter becomes too big (typically something like $2 \log n$). If the recursion depth has become too big, the subrange $A[i, j]$ is simply sorted using heapsort. If the recursion depth has not become too big, we stop if $j - i < c$, where c is some constant (typically 16 or 32). In this case we simply sort the small array using insertion sort.

Your tasks As the CEO of the brewing company does not know anything about algorithms he has asked you to understand the algorithm and analyze it:

Task 3: Write pseudo-code for introsort. You may use functions from the book such as `HeapSort`, `InsertionSort`, and `Randomized-Partition`. You may find inspiration in the pseudocode for quicksort from CLRS.

Task 4: Show that the running time of introsort is worst-case $O(n \log n)$. You may use results from the course book without proving them.

Task 5: Discuss why we use heap sort rather than another $O(n \log n)$ sorting algorithm such as merge sort. (Hint: What are the properties of the different sorting functions?)

Task 6: In the description above, we use insertion sort to sort the small arrays of size $j - i < c$. An alternative is to simply return the recursive call without sorting this small array and instead call insertion sort with the entire (nearly sorted) array as input.

Why is it a good idea to run insertion sort on the nearly sorted data, when we know from CLRS that its worst-case running time is $\Theta(n^2)$?

[1]`http://gcc.gnu.org/onlinedocs/libstdc++/libstdc++-html-USERS-4.4/a01027.html`
[2]`http://msdn.microsoft.com/en-us/library/6tf1f0bc(v=vs.110).aspx`

2 Fibonacci heaps

1. Consider an implementation of Fibonacci heaps where CONSOLIDATE is not called in FIB-HEAP-EXTRACT-MIN. Show that there exists a sequence of n operations that take time $\Omega(n^2)$ in total. What does it imply about the amortized bounds without CONSOLIDATE?

2. The book's implementation of CONSOLIDATE requires a bound of $D(n) = O(\lg n)$ children for each root node. Assume that this bound is instead $D(n) = n$. Show that after CONSOLIDATE, there are at most $\sqrt{2n}$ root nodes in the root list.

3 Dynamic programming

Motivation The goal of this assignment is to understand the aspects of dynamic programming: Optimal substructure and overlapping subproblems. Dynamic programming is a very important tool, and being able to understand the recursive nature of such problems can greatly reduce running times.

3.1 Problem statement

Three students at University of Copenhagen have just graduated and want to celebrate. To do this they wish to drink a lot of beer, but first they have to buy it. The local store features n different beers from the CPS Brewing Company. The ith beer has a price of p_i DKK and the students have C total DKK to spend. They now wish to know in how many ways they can spend exactly C DKK buying beers and have asked you to help calculating this.

It is assumed that C and the beer prices p_i are integers. There is only one of each beer, i.e., the ith beer is a single bottle having price p_i DKK.

Example Consider the following example with $n = 5$ beers and $C = 5$ DKK. The different beer prices are

$$p_1 = 2 \quad p_2 = 3 \quad p_3 = 2 \quad p_4 = 1 \quad p_5 = 4$$

Out of the 2^n possible subsets of beer only the following have a total price of 5:

$$(p_1, p_2), \quad (p_1, p_3, p_4), \quad (p_2, p_3), \quad (p_4, p_5),$$

so the total number of ways the students can spend all their money is 4.

3.2 Your tasks

Task 1: Write a recursive formula for the number of ways the students can spend all their C DKK. *Hint: You can look at a formula, where $N(C, i)$ denotes the number of ways to spend exactly C DKK on beers with prices $p_1, \ldots, p_i$.*

Task 2: Prove that your recursive formula is correct and that it consists of overlapping subproblems.

Task 3: Turn your recursive formula into an $O(nC)$ dynamic programming algorithm. Provide pseudocode for the algorithm. You can use either memoization or bottom-up DP.

Task 4: Prove the correctness and running time of your algorithm.

What is the memory usage of your algorithm?

4 Greedy Algorithms

Motivation The goal of this assignment is to familiarize the student with the proof methods of greedy algorithms: the greedy-choice property and optimal substructure. Furthermore you should see how we can attack some optimization problems by turning them into easier decision problems, which can be used to solve the original problem.

4.1 Problem statement

It is summer in Sunny Beach. A famous Danish chain of bars (not to be named) owns n bars along the shore, which are connected by the shore highway. Each bar has a limited supply b_i of beers. Since no one knows which bar the customers will prefer, we would like each bar to have the same amount of beers. In order to achieve this, that company has hired two Danish students to haul beers between the bars in their truck. However, being Danish, the students each drink a beer per kilometer they drive[3].

Your task will be to help the chain calculate the largest amount of beers $\hat{b}$ they can have at *all* bars. Specifically, you must describe an algorithm with the following input and output specifications:

Input: The position p_i and beer supply b_i of each bar. Here b_i is the supply for the bar at position p_i. You may assume that the positions are in sorted order – i.e. $p_1 < p_2 < \ldots < p_n$.

Output: The largest amount $\hat{b}$, such that each bar can have a beer supply of at least $\hat{b}$ after the two students have transferred beer between the bars.

Example Consider the example in Figure 4.1. Here we have $b = (20, 40, 80, 10, 20)$ and $p = (0, 5, 13, 33, 36)$ (in kilometers). In order to send one beer from bar 3 to bar 4 we need to put 41 beers in the truck, as the students will drink 40 before reaching their destination (to send two beers we need to put 42 in the truck). The optimal $\hat{b}$ for the example is 21 and can be achieved as follows:

1. Bar 2 sends 11 beers towards bar 1. One beer arrives while ten are "lost" on the way.
2. Bar 3 sends 59 beers towards bar 4. 19 arrive while 40 are lost on the way.
3. Bar 4 now has 29 beers and send eight towards bar 5. Two of these arrive and six are lost on the way.
4. The final distribution of beer is: $(21, 29, 21, 21, 22)$.

[3]The author of this assignment does not condone drunk driving in any way!

Note that we can employ as many trucks as possible starting at arbitrary places. You can check for yourself that it is not possible to get 22 beers at each bar.

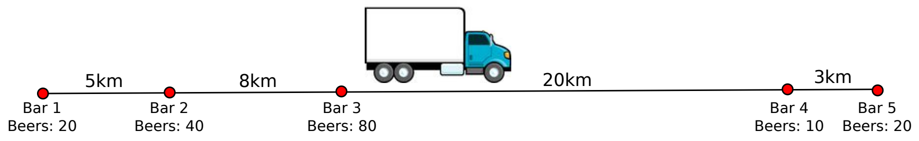


Figure 4.1: An example of a bar layout and b_i values.

4.2 Your tasks

Task 1: Consider first the related decision problem: Given an integer $\bar{b}$. Determine whether it is possible to get at least $\bar{b}$ beers in every bar.
Give a greedy algorithm that solves this problem in $O(n)$ time, where n is the number of bars.
In order for the greedy choice property and optimal substructure to make sense for a decision problem, you can define an optimal solution to be a solution with at least $\bar{b}$ beers in every bar if such a solution exists; otherwise, any solution is an optimal solution. *Hint: Consider the bars in increasing order (left to right in Figure 4.1); for bar $i < n$, either it needs beers from bar $i + 1$ or it may have a surplus of beers for bar $i + 1$.*

Task 2: Give a formal proof of correctness of your algorithm from task 1. If you are unsure how to do this, you can follow the structure from the book by showing that the problem exhibits the greedy-choice and optimal substructure properties.

Task 3: Using the procedure from task 1 give an $O(n \log B)$ algorithm to find the maximum value $\hat{b}$. Here B is the maximum amount of beers in any bar (i.e. $\max(b_1, \ldots, b_n)$). You must argue for the correctness and running time of your algorithm.

5 Binary Search Trees

Motivation The goal of this assignment is to familiarize the student with binary search trees. We will see how we can modify the standard binary search trees from CLRS in order to tackle other problems.

5.1 Problem statement

The CPS Brewing Company would like to know which of their beers are doing well and which are not. To do this, they wish to investigate the revenue generated by each different beer. As an example they might want to know what the median profit is among all the beers or among some group of beers (e.g. all lager beers).

The CPSBC wants you to develop a data structure that supports the following operations:

- Insert a key.
- Delete a key.
- Query the kth smallest key in the data structure.

An external consultant tells you that this can be accomplished with binary search trees, but you might have to modify the ones from CLRS.

5.2 Your tasks

Consider the node types used in CLRS chapters 12 and 13. We will add an extra field $x.size$, which denotes the size of the subtree rooted in x. The size of a subtree is the number of nodes in that subtree. In order to maintain this field a few things need to be changed.

Task 1: Using the size field described above, give pseudocode for a procedure `Get-kth-Key`(x, k), which returns the kth smallest key in the binary search tree (BST) rooted in x.

If k is not positive or bigger than the number of elements in the tree, your function should return **Nil**.

Task 2: Prove the correctness of your algorithm in task 1. *Hint: You may do this using induction over the tree size.*

Task 3: Modify the pseudocode of `Left-Rotate` (CLRS section 13.2) such that it correctly updates the size field. You do not have to include the entire pseudocode – just state the necessary changes in a readable way.

Task 4: Modify the pseudocode of `RB-Insert` and `RB-Insert-Fixup` (CLRS section 13.3) such that the size field is updated. You can assume that the rotate functions update the size field correctly (this is addressed in Task 3 for `Left-Rotate`). You do not have to include the entire pseudocode – just state the necessary changes in a readable way.

Task 5: What is the worst-case running time of `RB-Insert` now? What are the worst-case running times of the three operations discussed in section 5.1?

Task 6: Consider a sequence of n insertions/deletions and m queries (that is calls to `Get-kth-Key`) intertwined arbitrarily. What is the worst-case running time of handling such a sequence with your algorithm?

6 Shortest Path and Minimum Spanning Tree

Motivation The goal of this assignment is to familiarize you with the properties of distances in graphs used for shortest paths and MST algorithms. It is also an illustration of how one can use the MST of a graph as a basic starting point for solving a problem. Finally, the assignment should also familiarize you with usage of disjoint sets.

6.1 Problem statement

The CPS Brewing Company controls a complicated network of breweries connected by roads. Having learnt quite a lot of algorithms, the company now only uses shortest paths to transfer ingredients between breweries. However, the company has only kept track of these shortest distances, and not the actual roads between breweries.

The company is planning to expand, and would therefore like you to calculate how the network actually looks. The CEO of CPSBC has told you that the network satisfies the following properties:

1. There are exactly n breweries and $n-1$ roads.
2. All breweries are connected with each other – i.e. the network consists of just *one* connected component. The network is thus a tree.
3. All roads have positive length (strictly greater than 0). All roads can carry traffic in both directions.
4. You are given the shortest path distance between each pair of breweries.

Given this information the CEO has asked you to figure out how the network could possibly look.

More formally: Let the actual brewery network be a tree T. Given just the shortest path distances of T, you have to reconstruct the original network T.

Input: An $n \times n$ distance matrix H with $H_{ij} = \delta_T(i,j)$, where T is the actual network of breweries and $\delta_T(i,j)$ is the shortest path distance between breweries i and j in T.

Output: The $n-1$ edges of T.

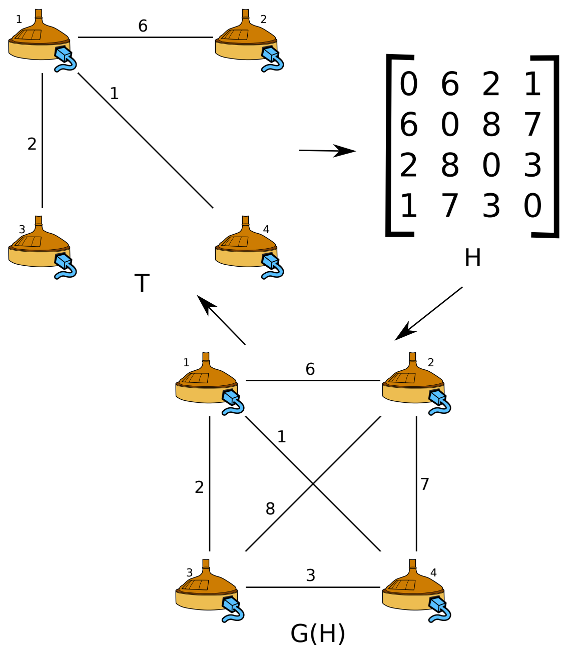
Example We will use a few definitions to help with the notation:

- T is the actual brewery network.
- H is the $n \times n$ shortest path distance matrix.
- $G(H)$ is the complete graph on n nodes, where edge (i,j) has weight H_{ij} – i.e. the shortest path distance in T.

As an example you may look at Figure 6.1. It shows the transformation from T to H and H to $G(H)$ and $G(H)$ back to T.

6.2 Your tasks

Task 1: Prove that an entry H_{ij} of minimal value must be an edge of T – i.e. if H_{ij} has minimum value among all entries of H, then (i,j) is an edge of T and has $w(i,j) = H_{ij}$; the minimum is only taken over non-diagonal entries of H. Do not mention minimum spanning trees in your proof. *Hint: Try to prove this by contradiction. Recall that all edges of G have positive weight (strictly greater than 0).*



$$\begin{bmatrix} 0 & 6 & 2 & 1 \\ 6 & 0 & 8 & 7 \\ 2 & 8 & 0 & 3 \\ 1 & 7 & 3 & 0 \end{bmatrix}$$

Figure 6.1: Example problem instance.

Task 2: Consider the complete graph $G(H)$ as described in the example above. Prove that a minimum spanning tree T' of $G(H)$ is equal to T. You must prove the statement for the general case and not just the example. *Hint: Consider an edge e added in a step of Prim's or Kruskal's algorithm. Is it possible that e was not a part of T? You should be able to argue the same way as in task 1.*

Task 3: What is the running time of the algorithm resulting from running a MST algorithm on the input and returning the list of edges as a function of n? (Note that $|E(G(H))| = \Theta(n^2)$).

References

[1] Musser, David R. "Introspective sorting and selection algorithms." *Softw., Pract. Exper.* 27.8 (1997): 983-993.