AD – Assignment 1

Sorting and recurrence analysis

Søren Dahlgaard

Hand-in date: May 15th

Should be done in groups of 2-3 students

1 Motivation

This week's 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.

2 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 2018 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's 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. Be

careful to avoid the pitfalls of the substitution method (see CLRS, page 86).

Additionally, you have to draw a recursion tree down to at least four levels for one of the recurrences.

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.

3 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 $O(n^2)$. 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].

3.1 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.

3.2 Assignment

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: Argue that the running time of introsort is worst-case $O(n \log n)$.

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

[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`

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 $O(n^2)$?

References

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