Exercises - Divide and Conquer (1) - Algoritmiek

Tutorial February 6, 2024

1. **Bounds:** Consider a sorted array A of (not per se distinct) integers and two integers $\ell \leq r$.
   (a) Show how to decide whether an integer k is contained in A that satisfies $\ell \leq k \leq r$ in $O(\log n)$ time.
   (b) Show how to count how many integers k are contained in A that satisfy $\ell \leq k \leq r$ in $O(\log n)$ time.

2. **Master theorem:**
   (a) Consider the recurrence $T(n) = 4T(n/2) + n^3$; $T(1) = 1$. What solution does the Master theorem give?
      A. Apply Case 1 to get $\Theta(n^2)$
      B. Apply Case 2 to get $\Theta(n^2 \log n)$
      C. Apply Case 3 to get $\Theta(n^3)$
      D. The Master Theorem does not apply.
   (b) Consider the recurrence $T(n) = 0.5T(n/2) + n \log n$; $T(1) = 1$. What solution does the Master theorem give?
      A. Apply Case 1 to get $\Theta(1/n)$
      B. Apply Case 2 to get $\Theta(n \log^2 n)$
      C. Apply Case 3 to get $\Theta(n \log n)$
      D. The Master theorem does not apply.
   (c) Consider the following two recurrences: $T(n) = 2T(n/2) + O(n \log n)$ and $T(n) = 2T(n/2) + O(n\sqrt{n})$, both with $T(n) = 1$. Use the Master theorem to solve them. Are the solutions the same or different? Discuss why.
   (d) Consider the following recurrence: $T(n) = 2T(n/2) + O(2^n)$; $T(1) = 1$. Does the Master Theorem apply? Discuss why/why not.
   (e) Consider Problems 4-1 and 4-4 of the book to exercise more with the Master theorem.

3. **Substitution:** Consider the following recurrences. Solve them (or prove a very good upper bound) using the substitution method. In all cases, $T(1) = 1$.
   (a) $T(n) = T(n/2) + 1$ (what is the subtle difference compared to what we saw in class?)
   (b) $T(n) = 3 \cdot T(n-1)$
   (c) $T(n) = T(n-1) + n$

4. **Minimum finding:** Finding the minimum in an array can be seen as a divide & conquer algorithm.
   (a) Give the recursive algorithm to find the minimum. Then prove using induction why the algorithm is correct.
   (b) Give the recurrence for the running time of the algorithm.
   (c) Solve the recurrence. Use at least two of the methods covered in the lecture.

5. ***O*-notation (repeat from Datastructuren):** This exercise is based on Problem 3-4 in Cormen et al. Let $f(n)$ and $g(n)$ be two non-negative functions. Answer the questions below. If your answer is yes, give a proof; if your answer is no, give a counterexample or counter-proof.

   - Is it true that $5n^2 = O(n^2)$?
   - Is it true that $5n^2 = \Theta(n^2)$?
   - Is it true that $5n^2 = o(n^2)$?
   - Does $f(n) = O(g(n))$ imply that $g(n) = O(f(n))$?
   - Does $f(n) = o(g(n))$ imply that $g(n) = \omega(f(n))$?
   - Does $f(n) = O(g(n))$ imply that $2^{f(n)} = O(2^{g(n)})$?
   - Is it true that $f(n) + o(f(n)) = \Theta(f(n))$?

6. ***O*-notation:**

   (a) Consider the following functions and order them according to their asymptotic growth: 2^n, $\log\log n$, $\sqrt{n}$, $\log n$, n^3, $2^{\sqrt{n}}$, $\log^5 n$.

   (b) Create a table with ten columns and many (16) rows. Label the first column by 'function', the second by 'constant', and the further seven by the above functions in order of increasing asymptotic growth. The final column should be labeled 'too big'.

   (c) Consider the following functions and put them the first column in any order: $(1.01)^n$, $n/\log n$, $\sqrt{n}\log n$, $\log(n^3)$, $\sqrt{\log n}$, $\log(n\log n)$, $2^{\log n}$, 2^{n^2}, $\log 128$, $1/n$, $\log(n)\cdot\log(n)$, $2^{\sqrt{n}} + n^2$, $\log^7 n$, $\sqrt[3]{n}$, $2\cdot 2^n$.

   (d) Consider a cell of the table. Let $f(n)$ be the function in the row and $g(n)$ be the function in the column. Put a mark in the cell if $f(n) = O(g(n))$. Use the column 'too big' if you think f is not big-Oh of any function. For example, in the row marked $n/\log n$ and column marked 2^n, there should be a mark. Fill out the entire table.

   (e) Why does it not make sense to add a column 'too small'?

   (f) Do the same for $\Omega()$ instead of $O()$. What about 'too big' versus 'too small' in this case?

Optional, more difficult exercise.

6. **Memory Usage of MergeSort:** Suppose you implement MergeSort as follows:

```
MergeSort(A, a, b) {
if b-a==1 then return new int[] {A[a]};
R1 = MergeSort(A,a,(a+b)/2);
R2 = MergeSort(A,(a+b)/2,b);
R = new int[b-a];
merge R1 and R2 into the array R
return R; }
```

   (a) How much memory do you allocate in total if you run this implementation on an array of n integers?

   (b) Your input is large and therefore, you try to write an implementation of MergeSort that uses less memory. Give an implementation that allocates $n + O(\log n)$ working memory in total (or $2n + O(\log n)$ if you count the input), but still has running time $O(n\log n)$.

   (c) Your input is very large and therefore, you try to write an implementation of MergeSort that uses much less memory. Give an implementation that allocates $(n/2) + O(\log n)$ working memory in total (or $(3/2)n + O(\log n)$ if you count the input), but still has running time $O(n\log n)$.
   Hint: You will need, amongst others, the answer to part (b).