Sheet 1: Introduction - Exercices

Advanced Algorithms - Master DSC/MLDM

Recap

Classic series:

- $\sum_{i=1}^{n} i = 1 + \cdots + n = \frac{n(n+1)}{2}$
- $a + ar + ar^2 + \cdots + ar^{n-1} = \sum_{j=0}^{n-1} ar^j = a\frac{1-r^n}{1-r}$.

1 Exercise

Let f and g be two functions that take non negative values and suppose that f is $O(g)$, show that g is $\Omega(f)$.
Repeat the same process for proving (easy?):

- If $f \in O(g)$ and $g \in O(h)$ then $f \in O(h)$
- If $f \in \Omega(g)$ and $g \in \Omega(h)$ then $f \in \Omega(h)$
- If $f \in \theta(g)$ and $g \in \theta(h)$, then $f \in \theta(h)$
- If $f \in O(h)$ and $g \in O(h)$ then $f + g \in O(h)$
- If $g \in O(f)$ then $f + g \in \theta(f)$

2 Exercise

Assume you have functions f and g such that $f(n)$ is $O(g(n))$. For each of the following statements, decide whether you think is true or false and give a proof or a counter example

1. $\log_2 f(n)$ is $O(\log_2 g(n))$
2. $2^{f(n)}$ is $O(2^{g(n)})$
3. $f(n)^2$ is $O(g(n)^2)$

3 Exercise

Arrange the following list of functions in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list than it should be the case that $f(n)$ is $O(g(n))$.

1. $f_1(n) = 10^n$
2. $f_2(n) = n^{1/3}$
3. $f_3(n) = n^n$
4. $f_4(n) = \log_2 n$
5. $f_5(n) = 2^{\sqrt{\log_2 n}}$

4 Recurrences

Can you find the solution for each recurrence?

1. $T(n) = T(n-1) + n$, $n \geq 2$ and $T(1) = 1$.
2. $T(n) = T(n/2) + n$, $n \geq 2$, $T(1) = 0$ and n a power of 2.
3. $T(n) = 2T(n/2) + n^2$, $n \geq 2$, $T(1) = 0$ and n a power of 2.
4. $T(n) = 2T(\sqrt{n}) + \log_2 n$, with $n \geq 4$ and $T(2) = 1$.
5. $T(n) = T(\sqrt{n}) + \log_2 \log_2 n$, with $n \geq 4$ and $T(2) = 1$.

5 Exercise

Consider the algorithm below that aims at detecting if half of a set of electronic chips have the same behavior (*e.g.* they have the same function).

```
Input: S set of chips;
begin
    if |S| = 1 then Return one chip;
    if |S| = 2 then
        test if two chips are equivalent;
        Return either chip if they are equivalent;
    Let S_1 be the set of the first ⌊n/2⌋ chips;
    Let S_2 be the set of the remaining chips;
    Call the algorithm recursively for S_1;
    if a chip is returned then
        test this chip against all other chips;
    if no chip with a majority equivalence has yet been found then
        Call the algorithm recursively for S_2;
        if a chip is returned then
            test this chip against all other chips
    Return a chip from the majority equivalence class if one is found;
end
```

Algorithme 1: Algorithm for finding a majority equivalence class in a set of chips

We assume that testing if two chips are equivalent works in constant time $O(1)$. Can you prove that this algorithm works in $O(n \log n)$?

Consider now this procedure called ELIMINATE:

1. Group all the chips by pairs- if the number of chips is odd one remains unmatched.
2. For pairs with equivalent chips, keep one of the two. For the other pairs discard both chips.

We apply then this procedure recursively on the remaining chips until there is at most one single chip.
At the end, if we have one chip left, we compare it to all the others.
Can you justify that this procedure is linear?