INF102 18H
Algoritmar, datastrukturar og programmering
Mandatory assignment 0

Student – Andrey Belinskiy (zur008)

Big-O Quiz

Function	f(n)	~	O()
A	$2n + 1$	$\sim 2n$	$O(n)$
B	$2n$	$\sim 2n$	$O(n)$
C	$\frac{n^2}{2} - \frac{n}{2}$	$\sim \frac{n^2}{2}$	$O(n^2)$
D	$\frac{n^2}{2} - \frac{n}{2} + n + 2$	$\sim \frac{n^2}{2}$	$O(n^2)$
E	1	~ 1	$O(1)$
F	$\frac{n}{2}$	$\sim \frac{n}{2}$	$O(n)$
G	$2n + 2$	$\sim 2n$	$O(n)$
H	$log(n) + 2$	$\sim log(n)$	$O(log(n))$
I	$2n - 1$	$\sim 2n$	$O(n)$
J	$nlog(n) + n$	$\sim nlog(n)$	$O(nlog(n))$
K	$\sqrt{n}$	$\sim \sqrt{n}$	$O(\sqrt{n})$
L	$log(n) + 1$	$\sim log(n)$	$O(log(n))$
M	??? +1	$\sim n^n + n$	$O(n^n)$
N	??? +1	$\sim log(n)$	$O(log(n))$

Function	f(n)	~	O()
O	$\frac{3^{n+1}-1}{2}$	$\sim\frac{3^{n+1}}{2}$	$O(3^n)$
P	$nlog(n)+2n$	$\sim 2n$	$O(n)$
Q	$log(n)+1$	$\sim log(n)$	$O(log(n))$
R	$n+2\sqrt{n}+1$	$\sim n$	$O(n)$
S	$n^3+\frac{n^3}{\sqrt{n}}$	$\sim n^3$	$O(n^3)$
T	$2n^2\sqrt{n}+n^2+\frac{n^2}{2}-n$	$\sim 2n^2\sqrt{n}$	$O(n^2\sqrt{n})$
U	$log(n)+log(\ldots log(n))+1$	$\sim log(n)$	$O(log(n))$
V	$2n$	$\sim 2n$	$O(n)$
W	$2n+log(n)+1$	$\sim 2n$	$O(n)$
X	??? $+1$	$\sim 2log(n)$	$O(log(n))$
Y	??? $+1$	$\sim\sqrt{n}$	$O(\sqrt{n})$ (assumptions: If n%2 == 0, then O(1); If n%3 == 0, then O(2))
Z	---	---	---

*Assuming log base is 2.

Union Find

c) In the solution for problem b) (Fakebool UnionFind), is it possible to use weighted quick union? If yes, explain how you would do it, if not explain why it is not possible.

In this situation we can't use the weighted union find to solve the problem described in task b). Let's assume that the tree with lowest element as the root has less weight than some other tree. After executing the weighted union operation, the root of the heavier tree will become the root of the lighter tree. Therefore, the root of the lighter tree will not be correct for the task b), because it won't be the lowest one for the elements of the lighter tree.

d) Describe a scheme for making m calls to union and/or find such that the number of array accesses becomes maximum. Analyze how many array accesses will be made with your scheme as a function of n and m.

In this situation the amount of array accesses becomes maximum if every consecutive element of the tree is connected to the previous one and if we call to union the last and second to last elements.

For example, if we have an array of elements [0, 1, 2, 3, 4], such that the root at the top of the tree is element 4, and each element from 0 to 3 is connected to the next one, i.e. the root of element 0 is 1, the root of element 1 is 2, and so on. Then, if we make a call to union the elements at the bottom of the tree, *union(0, 1);* the algorithm will have to go all the way up to the top to find the root of elements 0 and 1. In this case, the first call *find(0);* will take $n - 1$ steps to find the root of element 0, and the second call *find(1);* will take $n - 2$ steps. Finally, we have a constant operation of changing the root of element 0, i.e. $O(1)$.

Therefore, if we omit the actual time it takes to access the array to retrieve the element (operation *id[p]*), as well as the time it takes to assign the values to the variables *p* and *q* (*p = find(p); q = find(q);*), we can describe the scenario with maximum amount of accesses to the array as a function of *m* and *n*:

$$f(m) = (n-1) + (n-2) + 1$$

$$f(m) = 2n - 2$$

The Tilde approximation in this case will be $\sim 2n$, and the Big-O will be $O(n)$.