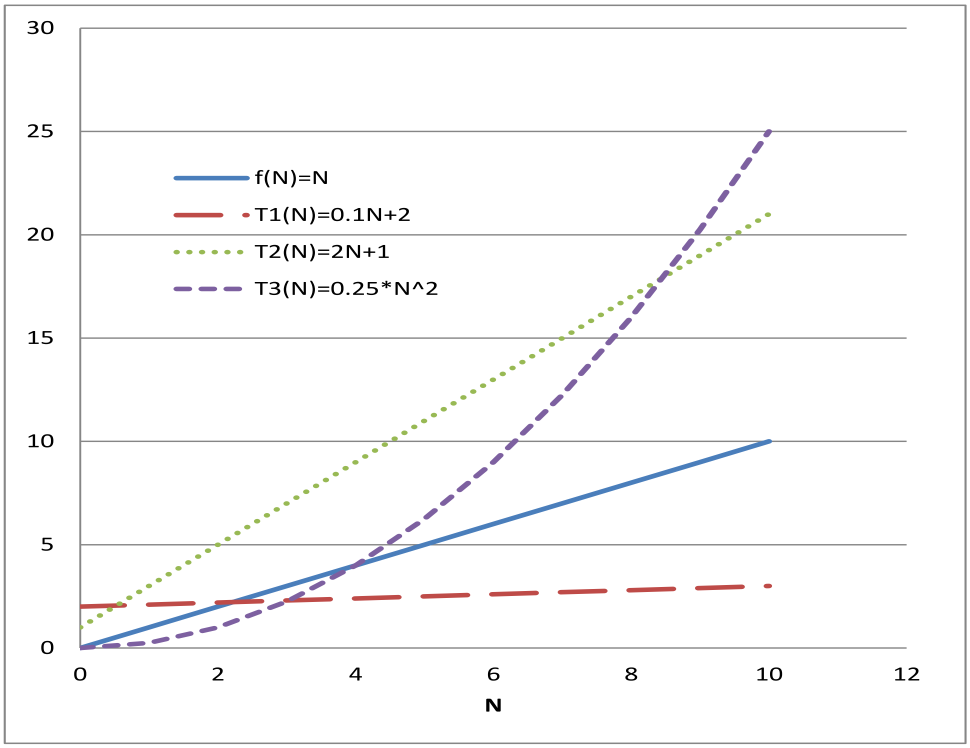
CS340 – Advanced Data Structures and Algorithm Design – Fall 2020
Handout 1 – September 09, 2020

Dr. Sandra Zilles, Department of Computer Science, University of Regina

Big-O Notation



Note. Multiplying a function by a constant factor c is like stretching or squishing its graph. Adding a constant to it is like moving its graph up or down vertically.

Intuition.

- $T_1(N) = O(f(N))$: even though for small N, the value of $T_1(N)$ is bigger than that of $f(N)$, from some n_0 onwards, all N satisfy $T_1(N) \leq f(N)$. In other words, *eventually*, $T_1(N)$ becomes upper-bounded by $f(N)$.
- Curiously though, despite the fact that *eventually*, $T_1(N)$ becomes upper-bounded by $f(N)$, we also have $f(N) = O(T_1(N))$: if we multiply $T_1(N)$ by an appropriate constant factor, e.g., $c = 10$, it becomes an upper bound on $f(N)$.
- The above two statements are simply because both $f(N)$ and $T_1(N)$ are linear functions in N. They have the same growth rate, i.e., the same asymptotic growth behaviour. In that sense, asymptotically, we can consider either one of them both an upper and a lower bound on the other, if multiplied by the respective appropriate constant.
- Likewise, $T_2(N) = O(f(N))$, $f(N) = O(T_2(N))$, $T_2(N) = O(T_1(N))$, $T_1(N) = O(T_2(N))$.
- The odd one out here is T_3. We can see that $T_3(N) \neq O(f(N))$: no matter how much we squish or stretch either of T_3 and f, eventually the squished/stretched version of T_3 will outgrow the squished/stretched version of f. This is because T_3 grows quadratically in N, while f grows only linearly. We have $f(N) = O(T_3(N))$, but $T_3(N) \neq O(f(N))$. Instead, we can say $T_3(N) = \Omega(f(N))$.