Asymptotic Complexity

Suppose \( f \) and \( g \) are functions. We say \( g \) is in \( O(f) \) if there exist \( c > 0, n_0 \geq 0 \) such that for all \( n > n_0 \), \( g(n) \leq cf(n) \).

\( O(f) \) is a set: \( O(f) \) is the set of functions \( \{g, h, \ldots\} \) such that for some \( c_g \) and \( n_{0g} \), \( c_h \) and \( n_{0h} \),..., for all \( n > n_{0g} \), \( g(n) \leq c_g f(n) \), for all \( n > n_{0h} \), \( h(n) \leq c_h f(n) \), etc.

Example: \( \{n^2, 4n^2 + 13n + 32, \frac{1}{2}n^2 + 3000n \log n\} \subseteq O(n^2) \)

Another way to think about it: \( g \) is in \( O(f) \) if \( \lim_{n \to \infty} \frac{g(n)}{f(n)} = k \), for some constant \( k \). (Note: \( k \) can be 0.)

Don’t expect to get credit if you use this argument, though. In proofs, always use the definition of big-O.

Selection Sort

Here is the code for selection sort from lecture. Let’s look at how it works:

```c
void swap(int[] A, int i, int j) {
    int tmp = A[i];
    A[i] = A[j];
    A[j] = tmp;
    return;
}

int min_index(int[] A, int lower, int upper) {
    int m = lower;
    int min = A[lower];
    for (int i = lower + 1; i < upper; i++) {
        if (A[i] < min) {
            m = i;
            min = A[i];
        }
    }
    return m;
}

void sort(int[] A, int n) {
    for (int i = 0; i < n; i++) {
        int m = min_index(A, i, n);
        swap(A, i, m);
    }
    return;
}
```
Here are a couple of functions. Can we figure out a tight big-O bound just by looking at the code?

```cpp
int binsearch(int x, int[] A, int n)
//@requires 0 <= n && n <= \length(A);
//@requires is_sorted(A,n);
/*@ensures (-1 == \result && !is_in(x, A, n))
|| ((0 <= \result && \result < n) && A[\result] == x); @*/
{
    int lower = 0;
    int upper = n;
    while (lower < upper)
    { //@loop_invariant 0 <= lower && lower <= upper && upper <= n;
      //@loop_invariant (lower == 0 || A[lower-1] < x);
      //@loop_invariant (upper == n || A[upper] > x);
      int mid = lower + (upper-lower)/2;
      //@assert lower <= mid && mid < upper;
      if (A[mid] == x) return mid;
      else if (A[mid] < x) lower = mid+1;
      else /*@assert(A[mid] > x);@*/ upper = mid;
    }
    return -1;
}

void time_waster(int n) {
    for (int i = 0; i < n; i++) {
        int[] arr = alloc_array(int, i);
        for (int j = 0; j < i; j++)
            arr[j] = j;
    }
    return;
}
```