0.1 Chapter 1: basics

0.1.1 What is an algorithm?

By definition, an algorithm is just a procedure to turn some input into some output through some calculations.

0.1.2 Selection Sort

The Champion Problem: The champion problem inputs an array A of n integers.
The program outputs an index such that A[k] is the minimum value.
We can solve this problem by looping through the array and selecting the least element as we find. This takes $4n + c$ comparisons.

0.1.3 sorting problem

Given a matrix of n integers, output the elements of the array in non-decrementing order.

0.1.4 Selection Sort

The idea behind selection sort is to use the `findMin()` procedure described above to sort an array. With an array of n elements, we find the min of n elements and place it in A[0]. Then, we find the min of n-1 elements and place it in A[1],... For each element a_i in the array, we loop over the remaining n-i elements, which means that we have to perfom $n(4n + c + 3)$ calculations.
Why does no. of calculations matter? Means that for bigger n, the number of calculations grows significantly.

0.2 Insertion Sort

The main idea is that we maintain a sorted array of length i, and then we take an element from the unsorted part of the array and insert it into its corresponding place in the sorted part. However, when the array is sorted backwards, we will perform many, many more calculations.

0.3 Asymptotic notation: O-Notation

$f(n) = O(g(n))$ means that:

$$\in (h(n) : \exists n_0, c \text{ such that} \forall n \geq n_0, ch(n) \geq f(n))$$

Basically, there exists a constant n and c such that $\forall n \geq n_0$, $c \times h(n) \geq f(n)$
Essentially, $h(n)$ describes some sort of upper bound for the number of calculations fro a given input n. Then, they also specify how the number of calculations grow as a function fo the input size. The def for $\Omega(n)$ and $\Theta(n)$ is the same, exept $\Omega(n)$ is just a lower bound and Θ(n) is a lower bound.

0.4 Merge Sort

Merge sort recursively breaks down the array into two halves, sorts the lower halves, and then merges the two sorted arrays into one. The complexity is $T(n) = T(n/2) + O(n)$,= $O(n \log(n))$, because we split the array down a total of $\log_2(n)$ times,a= and each time we operate n times over the array.