1 Chapter 1: basics

1.0.1 What is an algorithm?

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

1.0.2 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.

1.0.3 sorting problem

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

1.0.4 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.

1.1 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.

1.2 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.

1.3 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.

2 2: Lower Bounds

2.1 Membership Query

assuming we have a sorted array as input, how do we find an element inside it? **Binary Search**. Binary search runs in $O(\log(n))$

2.2 *"little"* notations

For o(n), $\theta(n)$, and $\omega(n)$, the definition is similar to their "big" counterparts.

$$f(n) = o(n) \text{ iff } \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$$f(n) = \omega(n) \text{iff } \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

Sometimes it is easiest to prove them with L'Hopital's rule. What are the implications of these? If f(n) = O(n), then f(n)=o(n), and same for $\Theta(n)$, however the reverse is not necessarily true.
asymptotically optimal: when we have O(f(n)) and Ω(n), then the upper bound can't be reduced, and the lower bound can't be raised.
super/subconstant: $\omega(1)$ and o(1), respectively.

2.3 Other sorting Paradigms

Inside the comparison-based model, any algorithm tatkes at least $\Omega(n \log(n))$ time

3 Data Structures

Since not every data structure is the perfect fit for every case, we have to discriminate based on insertion/ extraction, etc...

3.1 Young Tableau

A Young Tableau is a table where the elements in every row are sorted in ascending order, as well as every row. The cost for insertions are O(n+m), since we need to search in every row and column to see where the element is. We then change the index by performing swaps on the element until it is in the correct position.

3.2 Heaps

A heap is an implementation of a (full)binary tree in array form. An array is a **max heap** if every i in $[1..n]$ is less than or equal to its, parent, i.e.

$$A[i] \leq A[\text{parent}(i)]$$

where parent(i) = $\lfloor \frac{i}{2} \rfloor$
A *min* heap would be organized with the opposite relation. Since heaps are sorted arrays, why would we need any sorted arrays? The time to build a min or max heap is O(n), whereas sorting an array takes $\Omega(n \log(n))$.
Then how would we create a max heap from unordered elements? We use divide and conquer, recurse on the two halves, and then call *Max-Heapify* on the tree rooted at i.
In heapsort, we then srot the array by taking the maximum value in the array at every step and place it in our sorted array.

4 Geometric Algorithms

4.1 Bichromatic Planar Matching

In this algo, we divide the set of points into red and blue. Then, we try and find a feasible matching for the elements so that the line segments connecting them don't overlap. We can recursively split the problem until we have two sets of lines and then just resolve the individual conflicts.

4.2 Intersecting Lines

How do we check if two lines intersect? We check the angles between their two endpoints. we would have to ckeck if there is a point on the line AB that intersects CD.

4.3 Ham Sandwich Cut

Given a set of red and blue points, can we draw a straight line so that the number of blue and red points on

either side of the line is equal? We can recursively split the set of points needed to check by half, then solve th BMP problem for each one.

4.4 Closest Pair of Points

Given points on a plane, output the euclidean distance between the two closest points. $O(n^2)$ time if we check naively. $O(n \log(n))$ if we use D&C. Either the CPair can be on the left or right sides, or it could cross the middle of the graph. The answer would be the minimum of those three options. Narrow down the distance to within $2 * \delta$ away from the center line.