2 Graph Theory: Introduction to Algorithms

2.1 Complexity

Complexity is a measure of the resources used, both time and space. Of these, time complexity is the crucial one for most algorithms. It can be measured in computational time or the number of computational steps. The latter is less machine dependent, so we usually measure time in computational steps.

Efficiency An algorithm whose time complexity is $10n^2$ where $n$ is determined by the size of the input is more efficient for large $n$ than on whose time complexity is $\frac{1}{10}2^n$. For large enough $n$ the $\frac{1}{10}2^n >> 10n^2$. We say the order of $10n^2$ is less than or equal to the order of $\frac{1}{10}2^n$ because $10n^2 < 200\frac{1}{10}2^n$ for all $n \geq 1$.

An aside: Prove $10n^2 < 200\frac{1}{10}2^n$ for all $n \geq 1$ by induction. We begin by rewriting our ‘statement’ to be proved as $n^2 \leq 2^n+1$

Proof: see blackboard

Definition If $f$ and $g$ are two functions defined on the positive integers the order of $f$ is less than or equal to the order of $g$ if there are two positive constants $C$ and $n_0$ such that $f(n) \leq Cg(n)$ for all $n \geq n_0$

We write $f(n) = O(g(n))$ (Big “Oh”)

Examples: We have $\log(n) = O(n)$ and $n = O(n \log(n))$. But note that $n \neq O(\log(n))$. So there is a hierarchy of increasing orders: $O(1), O(\log(n)), O(n), O(n \log(n)), O(n^2), O(n^3), O(2^n), O(n!)$

Lemma If $f(n) = O(g(n))$ then $f(n) = O(kg(n))$ for any $k > 0$.

Definition If $f(n) = O(n^k)$ we say $f(n)$ is of polynomial order. Algorithms whose time complexity is of polynomial order are generally said to be efficient or good.

Note: It can be seen that every polynomial is of lower order than every exponential of the form $a^n$ where $a > 1$.

Definition If $f(n) \geq a^n$ for $a > 1$ we say $f$ is of order at least exponential.

Note: Algorithms with exponential time complexity or worse are not generally called efficient (related to “hard” problems). A recent algorithm for a problem called ”approximate Euclidean Steiner trees” has time complexity $O(n(\log(n))^400)$. The author called this ”nearly linear”!!!

2.2 Searching and sorting

Problem (searching) Given a list of words find if a given word is in the list.
Sequential search algorithm: Test the words one by one in order given.

If there are $n$ words, this takes up to $n$ comparisons, so the time complexity is $O(n)$. This is called worst case complexity. (If we are lucky we might stop much earlier.)

Note: Sequential search does not depend on the fact that the words are alphabetically sorted.

Now, suppose the given list is in alphabetical order. For a faster algorithm we can make use of the alphabetical order. The **binary search algorithm** is a divide and conquer type algorithm: first decide if the word is in the first half or the last half of the list, then repeat the algorithm with that half of the list.

The following algorithm finds if the word “KEY” is in the list $w(1), \ldots, w(n)$ (alphabetical arrangement is assumed). Note that $\lceil x \rceil$ is the least integer not less than $x$ called the ceiling of $x$. Eg. $\lceil 6.7 \rceil = 7$ while $\lceil 6.0 \rceil = 6$.

**Algorithm ‘Binary Search ($w(1), \ldots, w(n)$)’**

1. If $n \geq 1$ do
   
   (a) If KEY = $w(\lceil \frac{n}{2} \rceil)$, output YES
   
   (b) If KEY comes before $\lceil \frac{n}{2} \rceil$ in alphabetical order, Binary Search ($w(1), \ldots, w(\lceil \frac{n}{2} \rceil - 1)$)
   
   (c) Otherwise Binary Search ($w(\lceil \frac{n}{2} \rceil + 1), \ldots, w(n)$)

2. Else output NO.

Notes: The worst-case time complexity of Binary search is $O(\log(n))$. We can prove this by showing that the algorithm calls itself recursively at most $(\log_2(n)) + 1$ times. (For each recursive call there are a constant number of steps performed.

Note that $\lfloor x \rfloor$ is the greatest integer not more than $x$ called the floor of $x$. Eg. $\lfloor 6.7 \rfloor = 6$ while $\lfloor 6.0 \rfloor = 6$.

Proof: see blackboard

Sorting Algorithms

Given a list of words, put them in alphabetic order. (Note: putting a list of numbers in increasing order is an equivalent problem.)

Selection(or sequential) sort algorithm: Select the first word alphabetically in the list. Put it at the front (or swap with the first word), then sort the rest of the list using Selection Sort.

Notes: Selection sort can be performed in time $O(n^2)$. To do this note that we can select the first word alphabetically by running through the list keeping track of the “least” word found so far. (least means in alphabetical order). Since we need to look at each word once this takes time $O(n)$. This means time for one word is $\leq Cn$. There are $n$ words so that the total time $\leq Cn^2$. We can use induction to get a finer estimate.

Notes: There are many sorting algorithms eg merge sort and heapsort (both $O(n \log(n))$, and quicksort ($O(n^2)$). These are worst-case complexities but on average quicksort takes time $O(n \log(n))$ and is popular. (try scrambling input to get ‘average’ case)