CMPS 101 Homework 1

Dominik Schmidt

Ocotber 3, 2018

Question 1

A common algorithm for sorting is Bubble-Sort. Consider an input array A, with n elements. You repeat the following process n times: for every i from n - 1, if A[i] and A[i + 1] are out of order, you swap them.

Algorithm 1 Pseudo Code for slightly optimized Bubble Sort

for all i in A **do**
  for $j \leftarrow 0, n - i - 1$ **do**
    if $A[j] > A[j+1]$ **then**
      swap $A[j]$ and $A[j+1]$
    end if
  end for
end for

Time Complexity analysis for Bubble Sort:
Consider the worst case scenario where A is sorted in descending Order. In this case the algorithm will swap $n - i$ elements for all n iterations. Hence, the upper bound is $O(n^2)$. Next consider the best case scenario where the array A is completely sorted. In this case primitive unoptimized Bubble Sort will still compare elements $n - i$ times for all n iterations, so the algorithm will run at least $c \cdot (n-1) \cdot (n)$ comparison operations. Hence, the lower bound is $\Omega(n^2)$. In any case, Bubble sort executes $c \cdot (n-1) \cdot (n)$ operations.

$$\lim_{n \to \infty} \frac{c \cdot n \cdot (n-i)}{n^2} = \frac{n^2}{n^2} = 1$$

Since $1 \in \mathbb{N}$ and $1 \neq 0$, and $\Omega(f(n)) = O(f(n))$, Bubble Sort runs in $\Theta(n^2)$ because $f(n) = \Theta(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \in \mathbb{N} \neq 0$, $\Omega(f(n)) = O(f(n))$.

Proof of correctness:

> $\forall\ 0 \leq k \leq n-1$: After the outer loop runs k times, the elements from indices $A[n-k-1]$ up to $A[n-1]$ are sorted and the k largest

elements in the array.

Proof by Induction on k.

Base Case: $k = 0$ is vacuosly true since $A[1, 0]$ contains no real elements.
Induction: Assume the invariant holds for all elements up to k. Prove for $k+1$. By the Inductive Hypothesis the k largest elements of the array A are sorted up from the index $A[n-k-1]$, the inner loop at iteration $j = k$ inserts the largest element in $A[0]$ to $A[k-1]$ at index $A[k]$. Thus the invariant holds for $k+1$.

Question 2

Use induction to prove the following statement: The number of subsets of $\{1, 2, ..., n\}$ having an odd number of elements is 2^{n-1}.

Proof. To prove this statement we will use Induction.

Basis Step:

Let $f(n)$ be the hypothesis that the number of subsets of $\{1, 2, \ldots, n\}$ having an odd number of elements is 2^{n-1}, then for $n = 1$ $f(n) = 2^{1-1} = 2^0 = 1$, subsets of $\{1\}$ are $\emptyset$ and $\{1\}$, out of which exactly one has an odd number of elements.

Inductive Hypothesis: $f(n+1) \rightarrow$ the number of subsets of $\{1, 2, \ldots, n, n+1\}$ having an odd number of elements is $2^{(n-1)+1}$

Inductive Step: By the inductive hypothesis the number of subsets having an odd number of elements of the set
$\{1, 2, \ldots, n, n+1\} = 2^{n-1} + x$, where x is the number subsets with an odd number of elements that are unique to $\{1, 2, \ldots, n, n+1\}$. By the definition of the cardinality of the powerset, the number of subsets of $\{1, 2, \ldots, n\}$ is 2^n. There exist exactly 2 subsets of $\{1, 2, \ldots, n, n+1\}$ for every $\{1, 2, \ldots, n\}$, half of which have an odd number of elements and half of which have an even number of elements. Let A denote the power set of $\{1, 2, \ldots, n\}$ and let B denote the powerset of $\{1, 2, \ldots, n, n+1\}$, then $|B - A| = 2^n$. Therefore $x = \frac{1}{2} \cdot 2^n = 2^{n-1}$.
$f(n+1) \rightarrow 2^{n-1} + 2^{n-1} = 2 \cdot 2^{n-1} = 2^{(n-1)+1}$

$\square$

Question 3

Let $f(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_k n^k$ be a degree-k polynomial, where every $a_i > 0$. Show that $f(n) \in \Theta(n^k)$. Furthermore, show that $f(n) \notin O(n^{k'})$, for all $k' < k$.

$$\lim_{k\to\infty} \frac{f(n)}{n^k} = \lim_{k\to\infty} \frac{a_0 + a_1 n + a_2 n^2 + \ldots + a_k n^k}{n^k} = 1$$

Since $1 \in \mathbb{N}$ and $1 \neq 0$, $f(n) \in \Theta(n^k)$ because Big Theta is defined as $f(n) = \Theta(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \in \mathbb{N} \neq 0$

$$k > k', \lim_{k,k'\to\infty} \frac{f(n)}{n^{k'}} = \lim_{k,k'\to\infty} \frac{a_0 + a_1 n + a_2 n^2 + \ldots + a_k n^k}{n^{k'}} = \infty$$

Since $\infty \not< \infty$, by the Big O Definition $f(n) \notin O(n^{k'})$.

Question 4

Prove that $\log_2 n = O(n^{1/10})$, but $\log_2 n$ is not in $\Omega(n^{1/10})$. Is $\log_2 n = \Theta(n^{1/10})$? Why or why not?

By L'Hopital's Rule

$$\lim_{n\to\infty} \frac{\log_2 n}{n^{\frac{1}{10}}} =$$

$$\lim_{n\to\infty} \frac{10}{\ln(2) n^{\frac{1}{10}}} = \frac{\lim_{n\to\infty} 10}{\lim_{n\to\infty} \ln(2) n^{\frac{1}{10}}} = \frac{10}{\infty} = 0$$

By the *Big O* defintion, a function$f(n) = O(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$, $\log_2 n = O(n^{1/10})$ since $0 < \infty$. By the *Big Omega* defintion, a function$f(n) = \Omega(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$, $\log_2 n \notin \Omega(n^{1/10})$ since $0 \not< 0$. Therefore, $\log_2 n \neq \Theta(n^{1/10})$ because by the *Big Theta* definition $f(n) = \Theta(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \in \mathbb{N} \neq 0$ and $O(f(n)) = \Omega(f(n))$. In this case neither of these conditions are satisfied.

Question 5

Suppose the input array A is in sorted order, *except* for k elements. In other words, there are $n - k$ elements of A that are already in sorted order, and the remaining k elements are out of order. Prove that Insertion-Sort on A runs in $O(nk)$ time.

Proof: Consider the case where the Array A is sorted up to index $A[n-k]$ and elements from index $A[n-k+1]$ to $A[n-1]$ are not sorted. In this case Insertion Sort will only loop over the first $n-k$ elements in the array because they are already sorted. Hence, Insertion sort runs in a linear $O(n-k)$ on the first $n-k$ elements. Next consider the remaining k elements. We know that the first $n-k$ elements are sorted, meaning that the maximum number of indices in the remaining Array that $\exists j \in A[n-k,\ldots,n-1]$ is away from it's sorted index is k. Therefore Insertion sort would have to swap at most k elements to sort elements $A[n-k-1]$ through $A[n-1]$. To sort the remaining k elements the algorithm needs k $O(n)$ operations.

Next consider the scenario where the sorted $n-k$ elements are not in consecutive order and rather the array looks something like $\{n,\ldots,8,6,5,3,\ldots,n+1\}$ in this case Insertion Sort may have to swap as many n elements and compare as many as $n \cdot k$ elements, leaving us with a combined runtime $O(nk)$.

Acknowledgements

For parts of Question 5 I consulted the TA as well as the section on Insertion Sort in *Introduction to Algorithms*, *3rd Edition*.