

COMP3600 - Algorithms

Mathematical Tools: Review and Applications

August 3, 2009

1 Why Analyse Algorithms?

1.1 Appreciating the Running Times of Algorithms

Suppose n is the size of the problem and $f(n)$ s are the following functions. Order them in order of growth from the slowest to the fastest:

$$n, n^3, n!, n \log n, 2^n, n^2, \sqrt{n}, \log n$$

1.2 Counting Sort

Counting sort is a sorting algorithm that takes advantage of the knowledge of the range of the integers to be sorted. The algorithm has an input array $A[1..n]$, an output array $B[1..n]$, and a temporary array $C[1..k]$. k is the range of integers being sorted, i.e., $0 \leq A[i] \leq k$ for any index i with $1 \leq i \leq n$.

```
COUNTING-SORT( $A, B, k$ )
1  for  $i \leftarrow 0$  to  $k$ 
2    do  $C[i] \leftarrow 0$ 
3  for  $j \leftarrow 1$  to  $\text{length}[A]$ 
4    do  $C[A[j]] \leftarrow C[A[j]] + 1$ 
5  ▷  $C[i]$  now contains the number of elements equal to  $i$ .
6  for  $i \leftarrow 1$  to  $k$ 
7    do  $C[i] \leftarrow C[i] + C[i - 1]$ 
8  ▷  $C[i]$  now contains the number of elements less than or equal to  $i$ .
9  for  $j \leftarrow \text{length}[A]$  downto 1
10   do  $B[C[A[j]]] \leftarrow A[j]$ 
11      $C[A[j]] \leftarrow C[A[j]] - 1$ 
```

How much time does counting sort require in Θ notation? (Analyse each “for” loop. Don’t bother proving it formally.)

1

2 Asymptotics

Prove or Disprove:

1. $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$
2. $n! = \omega(2^n)$

3 Summations

Give the O bound for

1. $\sum_{k=1}^n k^{9/4}$

4 Recurrences

Give the O bound for

1. $T(n) = 8T(n/2) + n^3$
2. $T(n) = T(\sqrt{n}) + 1$

5 Extra Practice

These are some harder problems that will require a bit more time. I encourage you to do these if you have time. Otherwise, they will make good exam revision!

Prove or Disprove

1. $f(n) = O(g(n)) \rightarrow a^{f(n)} = O(a^{g(n)})$

Give the Θ bound for

1. $\sum_{k=1}^n k^3 / 2^k$
2. $\sum_{k=2}^n k^{1/2} / \log k$
3. $T(n) = \sqrt{n}T(\sqrt{n}) + n$

2

5.1 Efficiency

Searching for the minimum and maximum elements in a list of n elements is a straightforward task. Doing this naïvely would require $2(n-1)$ comparisons (scan the n elements twice) to find the minimum and maximum elements. This task can actually be done very efficiently with only $3n/2 - 2$ comparisons. Devise such an algorithm.

3