MATH 2112/CSCI 2112, Discrete Structures I

Winter 2007

Toby Kenney

Homework Sheet 8

Due: Wednesday 21st March: 1:30 PM

Compulsory questions

1 (a) Consider the following algorithm for finding the nth fibonacci number:

Input: natural number n
Output: nth Fibonacci number
  if n=0 **then**
    return 0
  end if
  if n=1 **then**
    return 1
  end if
  Find the $n-1$th Fibonacci number {using this algorithm}
  Find the $n-2$th Fibonacci number {using this algorithm}
  Add them together and
  return the result.

Find a recurrence relation for the number of additions required to calculate the nth fibonacci number using this algorithm and solve it.

(b) Now consider the following algorithm to find both F_n and F_{n+1}:

Input: natural number n
Output: nth and $n+1$th Fibonacci numbers
  if n=0 **then**
    return 0 and 1
  end if
  Find F_{n-1} and F_n, the $n-1$th and nth fibonacci numbers {using this algorithm}
  return F_n and $F_{n-1}+F_n$.

How many additions does this algorithm need to calculate F_n and F_{n+1}?

2 Which of the following functions are $\Theta(n^a)$ for some $0 < a < \infty$. For functions which are $\Theta(n^a)$ for some a, give the value of a. For function which are not, are they $O(n^a)$ for all a? are they $\Omega(n^a)$ for all a? Justify your answers. You may use any of the results about O, Ω and Θ proved in the lectures.

(a) $f(n) = n^7 - 3n^{3.6} + 4$

(b) $f(n) = e^{2n}$

(c) $f(n) = 6$

(d) $f(n) = (n+3)\log(n)$

(e) $f(n) = n^3 + n(\log(n))^2$

(f) $f(n) = \sqrt{n} - \log(n^2+5)$

3 Consider the following algorithm for finding an element in a sorted list $a[1], a[2], \ldots, a[n]$ of length n.

Input: x item to search for
Output: index at which x occurs in the list (or **false** if it doesn't occur)
  if $n = 0$ **then**
    return **false**
  else
    Compare x to $a[n/2]$ {rounding $n/2$ up to the nearest integer}
    if $x = a[n/2]$ **then**
      return $n/2$
    else if $x < a[n/2]$ **then**
      use this algorithm to find x in the list $a[1], a[2], \ldots, a[n/2-1]$, and
      return the result.
    else if $x > a[n/2]$ **then**
      use this algorithm to find x in the list $a[n/2+1], a[n/2+2], \ldots, a[n]$, and
      return the result plus $n/2$.
    end if
  end if

(a) How many comparisons does this algorithm take to find x: [The order of magnitude is all that is required, e.g. $O(n^2)$ comparisons.]

(i) in the best case?

(ii) in the worst case?

(b) If the list is not sorted, the best search algorithm takes $O(n)$ comparisons to find x on average. How many searches must a program perform in order for it to be faster to sort the list with a merge-sort than to simply use an unsorted list? (Give the order of magnitude, i.e. something like "$\Omega(n^3(\log(n))^2)$ searches".) Justify your answer.

4 Recall the insertion sort: (This version is slightly different from the version in the textbook.)

Suppose the list $a[1], \ldots, a[n]$ is initially sorted, then 100 of its values are changed at random. How many comparisons and swaps will be needed for the insertion sort to sort the changed list? Explain your answer. [You only need to give the order of magnitude, e.g. $\Theta(n \log n)$.]

```
for i = 1 to n do
  for j = i − 1 to 0 do
    if j = 0 then
      move a[i] to the front of the list. {This requires i swaps.}
    else
      compare a[i] and a[j].
      if a[i] ⩾ a[j] then
        insert a[i] just after a[j] {This requires i − j − 1 swaps.}
        go to next i.
      end if
    end if
  end for
end for
```

Bonus question

5 Prove that any algorithm for sorting a list using only comparisons and swaps must use $\Omega(n \log n)$ comparisons in the worst case. [Hint: There are $n!$ possible orders the list can start in. The comparisons made must distinguish between all of these possibilities. You may use the fact that $\log(n!)$ is $\Theta(n \log n)$.]