INT102 Algorithmic Foundations
Problem Session 2, Week 4

Suggested Solutions

Question 1

Given the Bubble sort algorithm as below:

```
ALGORITHM BubbleSort(A[0..n − 1])
//Sorts a given array by bubble sort
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in ascending order
for i=0 to n − 2 do
    for j = n-1 downto i+1 do
        if A[j ]<A[j-1 ] swap A[j ] and A[j - 1]
```

1. What is the number of swapping operations needed to sort the numbers A[0..5]=[6，1，2，3，4，5] in ascending order using the Bubble sort algorithm? **(5 marks)**
2. What is the number of key comparisons needed to sort the numbers A[0..5]= [6，1，2，3，4，5] in ascending order using the Bubble sort algorithm? **(5 marks)**

Answer

1. The number of swapping operations is 5.
2. The number of key comparisons is 15.

Question 2

Given the Merge sort algorithm as below:

```
Algorithm Mergesort(A[0..n-1])
 if n > 1 then begin
  copy A[0..⌊n/2⌋-1] to B[0..⌊n/2⌋-1]
  copy A[⌊n/2⌋..n-1] to C[0..⌈n/2⌉-1]
  Mergesort(B[0..⌊n/2⌋-1])
  Mergesort(C[0..⌈n/2⌉-1])
  Merge(B, C, A)
 End

Algorithm Merge(B[0..p-1], C[0..q-1], A[0..p+q-1])
        Set i=0, j=0, k=0
        while i<p and j<q do
        begin
                if B[i]≤C[j] then set A[k]=B[i] and increase i
                else set A[k] = C[j] and increase j
                k = k+1
        end
        if i==p then copy C[j..q-1] to A[k..p+q-1]
        else copy B[i..p-1] to A[k..p+q-1]
```

What is the number of key comparisons needed to sort the numbers A[0..5]= [6，1，2，3，4，5] in ascending order using the Mergesort algorithm?

Answer: 10 times

Question 3:

The time complexity of the merge sort algorithm can be described by the following recurrence for T(n).

$$T(n)=\begin{cases}1 & if \quad n=1\\ 2T(n/2)+n & if \quad n>1\end{cases}$$

In the lecture we have proved that $T(n) = O(n \log n)$ using the substitution method (i.e., using mathematical induction). Now prove that $T(n) = O(n \log n)$ using the iterative method (unfolding the recurrence). Assume that $n= 2^k$

$$\begin{aligned}
T(n) &= 2T(n/2) + n\\
&= 2\,(2(T(n/2^2)+n/2) + n\\
&= 2^2T(n/2^2) + 2n\\
&= 2^2(2T(n/2^3) + n/2^2) + 2n\\
&= 2^3T(n/2^3) + 3n\\
&\dots\dots\\
&= 2^kT(n/2^k) + kn\\
&= 2^kT(1) + kn\\
&= 2^k+kn\\
&= n + n \log n < n \log n \quad for\ n>1
\end{aligned}$$

Therefore, $T(n) = O(n\log n)$

Question 4

1. Write a pseudocode for a divide-and-conquer algorithm for finding a **position** of the largest element in an array of n numbers.

2. Set up and solve a recurrence relation for the number of key comparisons made by your algorithm.

Answer:

1.

Algorithm: LP(A[l..r])
//find a position of the largest element in an array
//Input: an array A[l, r]
//Output: a position of the largest element in the array

```
IF l=r RETURN l
ELSE
  ll = LP(A[l.. ⌊r+l/2⌋ ])
  rr = LP(A[ ⌊r+l/2⌋+1.. r])
  IF A[ll] > A[rr] RETURN ll
  ELSE RETURN rr
```

2. For an array of size n we have the following recurrence relation

$$T(n)=\begin{cases} 1 & if \quad n=1 \\ 2\times T(n/2)+1 & if \quad n>1 \end{cases}$$

$T(n)=2T(n/2) + 1$

$= 2\,(2(T(n/2^2)+1) + 1 = 2^2T(n/2^2) + 2 + 1$

$=2^2(2T(n/2^3) +1) + 2 + 1=2^3T(n/2^3) + 2^2+ 2 + 1$

……

$=2^kT(n/2^k) + 2^{k-1}+ \ldots+ 2^2+ 2 + 1$

……

$=2^{\log(n)} + 2^{\log(n)-1} + \ldots+ 2^2+ 2 + 1=2^{\log(n)+1}-1=2n-1$

(Note that $2^0 + 2^1 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$)

OR

$$T(n)=\begin{cases} 0 & if \quad n=1 \\ 2\times T(n/2)+1 & if \quad n>1 \end{cases}$$

$T(n)=2T(n/2) + 1$

$= 2\,(2(T(n/2^2)+1) + 1 = 2^2T(n/2^2) + 2 + 1$

$=2^2(2T(n/2^3) +1) + 2 + 1=2^3T(n/2^3) + 2^2+ 2 + 1$

……

$=2^kT(n/2^k) + 2^{k-1}+ \ldots+ 2^2+ 2 + 1$

……

$=2^{\log(n)-1} + \ldots+ 2^2+ 2 + 1=2^{\log(n)}-1=n-1$

Question 5

1. Design a divide-and-conquer algorithm for finding values of both the largest and smallest elements in an array of *n* numbers.

2. Set up and solve (for $n = 2^k$) a recurrence relation for the number of key comparisons made by your algorithm.

Answser

1. Algorithm smallest-largest(*A, first, last*)
   //Input: An array *A*[first..*last*] of orderable elements
   //Output: A pair contains smallest and largest in the range [first,last]

```
if first =last return (A[first], A[last] )
  else
    (s1, l1)= smallest-largest (A, first, ⌊(first+last)/2⌋)
    (s2, l2)= smallest-largest (A, ⌊(first+last)/2⌋+1, last)
    If l1 < l2 then
        largest= l2
    else
        largest=l1
    If s1 < s2 then
        smallest= s1
    else
        smallest= s2
  return (smallest, largest)
```

2. $$T(n) = \begin{cases} 0 & if \quad n = 1 \\ 2T(n/2) + 2 & if \quad n > 2 \end{cases}$$

$$\begin{aligned}
T(n) &= 2T(n/2) + 2 \\
&= 2\,(2(T(n/2^2)+2) + 2 = 2^2T(n/2^2) + 2^2 + 2 \\
&= 2^2(2T(n/2^3) + 2) + 2^2 + 2 = 2^3T(n/2^3) + 2^3 + 2^2 + 2 \\
&\ldots\ldots \\
&= 2^kT(n/2^k) + 2^k + \ldots + 2^3 + 2^2 + 2 \\
&= 2^kT(1) + 2^k + \ldots + 2^3 + 2^2 + 2 \\
&= 2^k + \ldots + 2^3 + 2^2 + 2 = 2(2^k - 1) = 2(n-1)
\end{aligned}$$