INT102 Algorithmic Foundations
Problem Session 2, Week 4
Group1: 9:00-11:00, 03/17/2023, Friday
Group2: 17:00-19:00, 03/17/2023, Friday
Location: SC176

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]
```

→ 最后一个不用处理了

→ j从n-1到i+1

Array中的每个元素都要进行一次从后往前两个两个比较交换的操作

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次

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?

   ~~5×5=25次~~ 15次

   1. i=0时 j从5到1

   6 1 2 3 4 5

   1 6 2 3 4 5

   i=1时

   $n+(n-1)+\cdots+1 = \frac{n(n-1)}{2} = \frac{6\times 5}{2}$

   [6 1 2 3 4 5] = 15次

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]
```

分到最小

[6 1 2 3 4 5]

[6 1 2] [3 4 5]

[6] [1,2] [3] [4 5]

[1] [2] 1次 1次 [4] [5]

[6] [1 2] [3] [4 5]

[1 2 6] 2次 [3 4 5] 1次

[1 2 3 4 5 6] 1次

6 1 2 3 4 5

6 1 2 | 3 | 4 5 2次

1 2 6

1 3 4 5

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?

10次

1 2 3 4 5 6 5次

Question 3:

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

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

$T(\frac{n}{4}) = 2T(n/2^3) + \frac{n}{2^2}$

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

$T(\frac{n}{2}) = 2T(n/2^2) + \frac{n}{2}$

$T(n) = 2(2T(n/2^2) + \frac{n}{2}) + n$

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

$= 2^2(2T(n/2^3) + \frac{n}{2^2}) + 2n$

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

⋮

$2^kT(n/2^k) + kn$

$T(n) = nT(n/2^k) + n\log n$

$= nT(n/n) + n\log n$

$= n + n\log n < n\log n \; (n>1)$

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

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.