Analysis of MergeSort and QuickSort

1. Analysis of MergeSort

Algorithm:

```
mergesort( int [] a, int left, int right)
{
        if (right > left)
        {
                middle = left + (right - left)/2;
                mergesort(a, left, middle);
                mergesort(a, middle+1, right);
                merge(a, left, middle, right);
        }
}
```

Assumption: N is a power of two.

For N = 1: time is a constant (denoted by 1)

Otherwise: time to mergesort N elements = time to mergesort N/2 elements plus time to merge two arrays each N/2 elements.

Time to merge two arrays each N/2 elements is linear, i.e. N

Thus we have:

```
(1) T(1) = 1

(2) T(N) = 2T(N/2) + N
```

Next we will solve this recurrence relation. First we divide (2) by N:

```
(3) T(N) / N = T(N/2) / (N/2) + 1
```

N is a power of two, so we can write

```
(4) T(N/2) / (N/2) = T(N/4) / (N/4) +1

(5) T(N/4) / (N/4) = T(N/8) / (N/8) +1

(6) T(N/8) / (N/8) = T(N/16) / (N/16) +1

(7) ……
(8) T(2) / 2 = T(1) / 1 + 1
```

Now we add equations (3) through (8) : the sum of their left-hand sides will be equal to the sum of their right-hand sides:

```
T(N) / N + T(N/2) / (N/2) + T(N/4) / (N/4) + … + T(2)/2 =

T(N/2) / (N/2) + T(N/4) / (N/4) + ….+ T(2) / 2 + T(1) / 1 + LogN
```

(LogN is the sum of 1s in the right-hand sides)

After crossing the equal term, we get

```
(9) T(N)/N = T(1)/1 + LogN
```

T(1) is 1, hence we obtain

```
(10) T(N) = N + NlogN = O(NlogN)
```

Hence the complexity of the MergeSort algorithm is **O(NlogN).**

2. Analysis of QuickSort

```
if( left + 10 <= right)
        {
        int i = left, j = right - 1;
        for ( ; ; )
          {
            while (a[++i] < pivot) {}
            while (pivot < a[--j] ) {}

                if (i < j)
                                swap (a[i],a[j]);
            else  break;
        }
       swap (a[i], a[right-1);
       quicksort ( a, left, i-1);
       quicksort (a, i+1, right);
     }
else  insertionsort (a, left, right);
```

Recurrence relation based on the code

1. the for loop stops when the indexes cross, hence there are N iterations
2. swap is one operation – disregarded
3. Two recursive calls:
    a. Best case: each call is on half the array, hence time is 2T(N/2)
    b. Worst case: one array is empty, the other is N-1 elements, hence time is T(N-1)

```
T(N) = T(i) + T(N - i -1) + cN
```

The time to sort the file is equal to

- the time to sort the left partition with **i** elements, plus
- the time to sort the right partition with **N-i-1** elements, plus
- the time to build the partitions

2. 1. Worst case analysis

The pivot is the smallest element

```
T(N) = T(N-1) + cN, N > 1
```

Telescoping:

```
T(N-1) = T(N-2) + c(N-1)

T(N-2) = T(N-3) + c(N-2)

T(N-3) = T(N-4) + c(N-3)

T(2) = T(1) + c.2
```

Add all equations:

```
T(N) + T(N-1) + T(N-2) + … + T(2) =

= T(N-1) + T(N-2) + … + T(2) + T(1) + c(N) + c(N-1) + c(N-2) + … + c.2
```

T(N) = T(1) + c times (the sum of 2 thru N) = T(1) + c(N(N+1)/2 -1) = **$O(N^2)$**

2. 2. Best-case analysis:

The pivot is in the middle

$$T(N) = 2T(N/2) + cN$$

Divide by N:

$$T(N) / N = T(N/2) / (N/2) + c$$

Telescoping:

$$T(N/2) / (N/2) = T(N/4) / (N/4) + c$$

$$T(N/4) / (N/4) = T(N/8) / (N/8) + c$$

……

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

Add all equations:

$$T(N) / N + T(N/2) / (N/2) + T(N/4) / (N/4) + \ldots. + T(2) / 2 =$$

$$= (N/2) / (N/2) + T(N/4) / (N/4) + \ldots + T(1) / (1) + c.\log N$$

After crossing the equal terms:

$$T(N)/N = T(1) + c\mathrm{Log}N$$

$$T(N) = N + Nc\mathrm{Log}N = \mathbf{O(N\log N)}$$

2. 3. Average case analysis

Similar computations, resulting in $T(N) = O(N\log N)$

The average value of $T(i)$ is $1/N$ times the sum of $T(0)$ through $T(N-1)$

$$1/N \ \Sigma \ T(j), \ j = 0 \text{ thru } N-1$$

$$T(N) = 2/N \ (\Sigma \ T(j)) + cN$$

Multiply by N

$$NT(N) = 2(\Sigma \ T(j)) + cN*N$$

To remove the summation, we rewrite the equation for N-1:

$$(N-1)T(N-1) = 2(\Sigma \ T(j)) + c(N-1)^2, \ j = 0 \text{ thru } N-2$$

and subtract:

$$NT(N) - (N-1)T(N-1) = 2T(N-1) + 2cN - c$$

Prepare for telescoping. Rearrange terms, drop the insignificant c:

$$NT(N) = (N+1)T(N-1) + 2cN$$

Divide by N(N+1):

$$T(N)/(N+1) = T(N-1)/N + 2c/(N+1)$$

Telescope:

$$T(N)/(N+1) = T(N-1)/N + 2c/(N+1)$$

$$T(N-1)/(N) = T(N-2)/(N-1) + 2c/(N)$$

```
T(N-2)/(N-1) = T(N-3)/(N-2) + 2c/(N-1)

….

T(2)/3 = T(1)/2 + 2c /3
```

Add the equations and cross equal terms:

```
T(N)/(N+1) = T(1)/2 +2c  Σ (1/j), j = 3 to N+1
```

The sum Σ (1/j), j =3 to N-1, is about LogN

Thus **T(N) = O(NlogN)**

Learning Goals

- Be able to analyze the complexity of mersesort and quicksort algorithms using recurrence relations

Exam-like questions

1. Show that the complexity of mergesort algorithm is O(NlogN) by using recurrence relations
2. Analyze the worst-case complexity of quick sort solving the recurrence relation.
3. Analyze the best-case complexity of quick sort solving the recurrence relation.

Back to Contents page

Created by Lydia Sinapova