# Analysis of the Time Complexity of Quick Sort Algorithm

Wang Xiang
School of Information and Electronic Engineering
Tianjin Vocational Institute
Tianjin, China
e-mail: pingfan6699@yahoo.com.cn

***Abstract*—Quick sort algorithm has been widely used in data processing systems, because of its high efficiency, fast speed, scientific structure. Therefore, thorough study based on time complexity of quick sort algorithm is of great significance. Especially on time complexity aspect, the comparison of quick sort algorithm and other algorithm is particularly important. This paper talks about time complexity of quick sort algorithm and makes a comparison between the improved bubble sort and quick sort through analyzing the first order derivative of the function that is founded to correlate quick sort with other sorting algorithm. The comparison can promote programmers make the right decision when they face the choice of sort algorithms in a variety of circumstances so as to reduce the code size and improve efficiency of application program.**

***Keywords*- quick sort algorithm; time complexity; partition**

Sorting algorithms are widely used in many aspects of data processing, information searches, business finance, computer encryption, etc. Quick sort algorithm, which is invented by famous computer scientist C.A.R Hoare, is considered as one of the fastest and best sorting algorithms including bubble sort, selection sorting and insertion sorting. Quick sort algorithm is also one of the good examples of the use of strategy of divide and conquer.

The basic thoughts of Quick sort algorithm are:

- Pick an element, which is called a pivot, from the list waiting to be sorted.
- Perform partition operation to realize that all elements in the list with values smaller than the pivot come before the pivot, while all elements in the list with values bigger than the pivot come after it (elements, which is equal to pivot, can go either way). After this partition operation, the pivot is in its final position of the list.
- Recursively sort the sub-list of smaller elements and the sub-list of bigger elements.

## I. The Content and Process of Partition Program in Quick Sort Algorithm

Let L[1…n] be an array waiting to be sorted. To make it easier for us to express the array, it can be expressed as L[first…last]. The function of quick sort is as follows:

```
void QuickSort(int L[], int first, int last){
      if(first<last){
           int split=part(L,first,last);
           QuickSort (L,first,split-1);
           QuickSort (L,split+1,last);
      }
}
```

From the program above, we can see that the key idea of quick sorting is the partition process. Elements move drastically in the array while running the partition program, which is one of the internal reasons for quick sort run faster than any other sorting method.

To begin with, two pointers, which are named first and last, should be set up, so as to run the partition program. Initial values of the two pointers are the upper bound and lower bound of the list waiting to be sorted respectively, i.e., LP=first, RP=last. If there are n elements in the list, then first=0, last=n-1.We pick the first element in the list as the pivot, i.e. pivot= L[first]. The partition program scanned the whole list from the pointer named RP to left, until it find an element that is smaller than pivot, then put the element in the position pointed out by the pointer named LP, and make the LR pointer shift right one position. After that, the partition program scanned the rest of the list from the pointer named LP to right, until it find an element that is not smaller than pivot, then put the element in the position pointed out by the pointer named RP, and make the RP pointer shift left one position. In fact, partition program finish its one cycle through the operation above. The function of partition is as follows:

```
int partition(int L[], int first, int last){
     int LP=first;
     int RP=last;
     int pivot=L[first];
     while(LP<RP){
             while((RP > LP) && (L[RP]>=pivot))  RP --;
             if(RP > LP) L[LP]=L[RP];
             while((LP < RP) && (L[LP]<=pivot))  LP ++;
             if(LP < RP) L[RP --]=L[LP];
}
     L[LP]=pivot;
     return LP;
}
```

## II. Calculation of the Time Complexity of Quick Sort Algorithm

The amount of time needed to finish quick sort is consumed mainly by the processes of partition program. Quick sort have to perform operations N-1 times to finish the comparison between different data in the array consisting of N elements (N≤n). In the worst case of quick sort algorithm, sort must run partition program N-1 times every time, if the key words values of pivot which is picked from the unordered region of the list waiting to be sorted are always the smallest (or the biggest). The length of the array is n-I+1 at the beginning of the Ith partition and quick sort must run partition program n-I times (1≤i≤n−1). At this time, the comparison comes to the maximum number of times. Let W(n) be time complexity of quick sort algorithm in the worst case, we can obtain the following equation very easily: $W(n)=n(n-1)/2=O(n^2)$. In the best case of quick sort algorithm, the key words values of pivot which is picked from the unordered region of the list in the array waiting to be sorted are always in a middle position. The partition result is that the left and right intervals of the pivot have the approximately equal lengths. Let B(n) be time complexity of quick sort algorithm in the best case, we can also obtain the following equation very easily: $B(n)\approx n-1+2B(n/2) =(n-1)+2(n/2-1)+4B(n/4)= \cdots =O(n\log_2 n)$. Let A(n) be time complexity of quick sort algorithm in the average case, generally we can explain the A(n) through the following recursive expression, i.e.,

$$A(n)=\begin{cases} 0 & n=1 \\ n-1+\sum_{i=1}^{n}\frac{1}{n}[A(i-1)+A(n-i)] & n>1 \end{cases}$$

the expression may boil down to:

$$A(n)=n-1+\frac{2}{n}\sum_{i=1}^{n-1}A(i)$$

$$=n-1+\frac{2}{n}[A(1)+A(2)+\cdots+A(n-1)]$$

and

$$A(n-1)=n-2+\frac{2}{n-1}\sum_{i=1}^{n-2}A(i)$$

$$=n-2+\frac{2}{n-1}[A(1)+A(2)+\cdots+A(n-2)]$$

Thus $\frac{A(n)}{n+1}=\frac{A(n-1)}{n}+\frac{2(n-1)}{n(n+1)}$,since A(0)=0,it follows that

$$\frac{A(n)}{n+1}=\frac{A(n-1)}{n}+\frac{2(n-1)}{n(n+1)}$$

$$=\frac{A(n-2)}{n-1}+\frac{2(n-2)}{(n-1)n}+\frac{2(n-1)}{n(n+1)}$$

$$=\cdots=\frac{2}{2\times 3}+\frac{4}{3\times 4}+\cdots+\frac{2(n-2)}{(n-1)n}+\frac{2(n-1)}{n(n+1)}$$

$$=2\sum_{i=1}^{n}\frac{i-1}{i(i+1)}=2\sum_{i=1}^{n}[\frac{1}{i}-\frac{2}{i(i+1)}]$$

$$=2\sum_{i=1}^{n}\frac{1}{i}-4\sum_{i=1}^{n}\frac{1}{i(i+1)}$$

$$=2\sum_{i=1}^{n}\frac{1}{i}-4\sum_{i=1}^{n}(\frac{1}{i}-\frac{1}{i+1})$$

$$=2\sum_{i=1}^{n}\frac{1}{i}-4(1-\frac{1}{n+1})$$

$$=2\sum_{i=1}^{n}\frac{1}{i}-\frac{4n}{n+1}$$

If we adopt the approximate calculation of the Harmonic series and assume the value of n is large enough, then we have

$$\sum_{i=1}^{n}\frac{1}{i}\approx \ln n+\gamma=\ln 2\times\log_2 n$$

$$=0.693\log_2 n+\gamma,\gamma\approx 0.577 \quad (\gamma \text{ is Euler constant})$$

Thus $\frac{A(n)}{n+1}\approx 2\times(0.693\log_2 n,+0.577)-\frac{4n}{n+1}$

i.e., $A(n)\approx 1.386(n+1)\log_2 n-2.846n+1.154$

## III. Comparison of Time Complexity in the Average Case Between the Improved Bubble Sort and Quick Sort

The function of the improved bubble sort algorithm is as follows:

```
void newbubblesort(int L[], int n){
       int p=1; int i,j;
       for(i=1;(i<n)&&p;i++){
            for(j=n;j>i;j--){
                p=0;
                if(L[j]<L[j-1]) {p=1; swap(L[j],L[j-1]);}
            }
       }
}
```

Notice that the complexity of the improved bubble sort, selection sorting and insertion sorting in the average case can be expressed as n(n-1)/4,n(n-1)/2 and $n^2/4$ respectively. Obviously, bubble sort has the lowest complexity among the three algorithms above. We use the symbols "$A(n)_{bubble}$" and "$A(n)_{quick}$" to denote "time complexity of the improved bubble sort algorithm in the average case" and "time complexity of quick sort algorithm in the average case" respectively. If n=11, then $A(n)_{quick}<A(n)_{bubble}$. To explore the relationship between the two algorithms for every n>11, let us begin to consider the behavior of a new function: $f(n)= A(n)_{bubble}- A(n)_{quick}$.

i.e.,

$$f(n)=\frac{n(n-1)}{4}-1.386(n+1)\log_2 n+2.846n-1.154$$

we can easily get the first order derivative of the function, such that

$$f'(n) = 0.5n - 1.386\log_2 n - \frac{2}{n} + 0.596$$
$$= \frac{n - 2.772\log_2 n}{2} - \frac{2}{n} + 0.596$$
$$= \frac{\log_2 2^n - \log_2 n^{2.772}}{2} - \frac{2}{n} + 0.596$$

We can obtain that if n>11, then $2^n$-$n^{2.772}$>0,-2/n+0.596>0. Thus,if n>11, then $f'(n)$ >0.This implies that $A(n)_{quick}$<$A(n)_{bubble}$ for every n ⩾ 11. To sum up, we conclude that quick sort should be preferentially applied when the number of elements in the list waiting to be sorted is no less than eleven. In other cases, we can consider applying other sorting algorithms including the improved bubble sort algorithm to design programs.

## IV. Analysis for improving quick sort performance

Partition weighs with time complexity in quick sort. Therefore, it is very important for quick sort to pick a suitable element as pivot. The suitable element should most approximate the average of elements in the first, middle and last position of the list. Let V be the suitable element we have picked, then quick sort must exchange value between V and the element that is in the first position of the list before performance of the partition operations.

Quick sort adopts the method of recursive algorithms, which is helpful to the implementation and analysis of the algorithm. However, the method make the algorithm require a large stack so as to enhance the time complexity and space complexity. We can create stacks within the program scientifically to take the place of the method of recursive algorithms so that the quick sort algorithm can diminish frequency of push-down stack procedures.

## References

[1] Liu Jing. An Introduction to Computer Algorithm— Techniques of Design and Analysis[M]. Beijing: Science Press, 2003.

[2] Deng Xiangyang, WanTingting. Design and Analysis of Algorithms[M]. Beijing: Metallurgical Industry Press, 2006.