Introduction to Algorithms HW4

109511207 蔡宗儒

Pseudo Code

```
PARTITION (A,p,r)
    x = A[r]
    i = p – 1
    for j = p to r-1
        if A[j] <= x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    return i+1

RANDOMIZED-PARTITION(A,p,r)
    i = RANDOM(p,r)
    exchange A[r] with A[i]
    return PARTITION (A,p,r)

RANDOMIZED-QUICKSORT(A,p,r)
    if p < r
        q = RANDOMIZED-PARTITION(A,p,r)
        RANDOMIZED-QUICKSORT(A,p,q-1)
        RANDOMIZED-QUICKSORT(A,q+1,r)
```

Introduction of Code

```
void swap(int &a,int &b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

Swap function

```cpp
int RM_Partition(vector<int> &v,int p,int r)
{
    int i = rand()%(r-p+1)+p;
    swap(v[r],v[i]);
    int pivot = v[r];
    i = p - 1;
    for(int j=p;j<=r-1;j++)
    {
        if(v[j] <= pivot)
        {
            i++;
            swap(v[i],v[j]);
        }
    }
    swap(v[i+1],v[r]);
    return (i+1);
}
```

RM_Partition 參考了課本的 pseudo code。與 Partition 不同的是，RM_Partition 的 pivot 並不是選擇陣列的最後一個值，而是用亂數選取的方式隨機選擇陣列內的某一值作為 pivot，如此一來可以降低選到最差的 pivot 的機率。

```cpp
void RM_Quicksort(vector<int> &v,int p,int r)
{
    if(p<r)
    {
        int q = RM_Partition(v,p,r);
        RM_Quicksort(v,p,q-1);
        RM_Quicksort(v,q+1,r);
    }
}
```

RM_Quicksort 也參考了課本的 pseudo code，用了 Divide and Conquer 的方法去 recursively 排序。

Randomized Quicksort 執行結果

```
Enter an integer for data size or enter CTRL+Z to terminate the program: 10
Enter 1 to show generated data otherwise hide it: 0
Enter 1 to show data after sorted otherwise hide it: 0
--------------------
Quick Sort...


Quick sort time used: 0.0033 ms
--------------------

Enter an integer for data size or enter CTRL+Z to terminate the program: 100
Enter 1 to show generated data otherwise hide it: 0
Enter 1 to show data after sorted otherwise hide it: 0
--------------------
Quick Sort...


Quick sort time used: 0.0178 ms
--------------------

Enter an integer for data size or enter CTRL+Z to terminate the program: 1000
Enter 1 to show generated data otherwise hide it: 0
Enter 1 to show data after sorted otherwise hide it: 0
--------------------
Quick Sort...


Quick sort time used: 0.3229 ms
--------------------

Enter an integer for data size or enter CTRL+Z to terminate the program: 10000
Enter 1 to show generated data otherwise hide it: 0
Enter 1 to show data after sorted otherwise hide it: 0
--------------------
Quick Sort...


Quick sort time used: 2.9115 ms
--------------------

Enter an integer for data size or enter CTRL+Z to terminate the program: 25000
Enter 1 to show generated data otherwise hide it: 0
Enter 1 to show data after sorted otherwise hide it: 0
--------------------
Quick Sort...


Quick sort time used: 11.6962 ms
```

```
Enter an integer for data size or enter CTRL+Z to terminate the program: 50000
Enter 1 to show generated data otherwise hide it: 0
Enter 1 to show data after sorted otherwise hide it: 0
--------------------
Quick Sort...


Quick sort time used: 19.844 ms
--------------------

Enter an integer for data size or enter CTRL+Z to terminate the program: 75000
Enter 1 to show generated data otherwise hide it: 0
Enter 1 to show data after sorted otherwise hide it: 0
--------------------
Quick Sort...


Quick sort time used: 33.0302 ms
--------------------

Enter an integer for data size or enter CTRL+Z to terminate the program: 100000
Enter 1 to show generated data otherwise hide it: 0
Enter 1 to show data after sorted otherwise hide it: 0
--------------------
Quick Sort...


Quick sort time used: 48.1851 ms
--------------------

Enter an integer for data size or enter CTRL+Z to terminate the program: 250000
Enter 1 to show generated data otherwise hide it: 0
Enter 1 to show data after sorted otherwise hide it: 0
--------------------
Quick Sort...


Quick sort time used: 131.142 ms
--------------------

Enter an integer for data size or enter CTRL+Z to terminate the program: 500000
Enter 1 to show generated data otherwise hide it: 0
Enter 1 to show data after sorted otherwise hide it: 0
--------------------
Quick Sort...


Quick sort time used: 229.299 ms
--------------------
```

上二圖為 randomized quicksort 在 input size 分別為 10, 100, 1000, 10000, 25000, 50000, 75000, 100000, 250000, 500000 所花的時間。

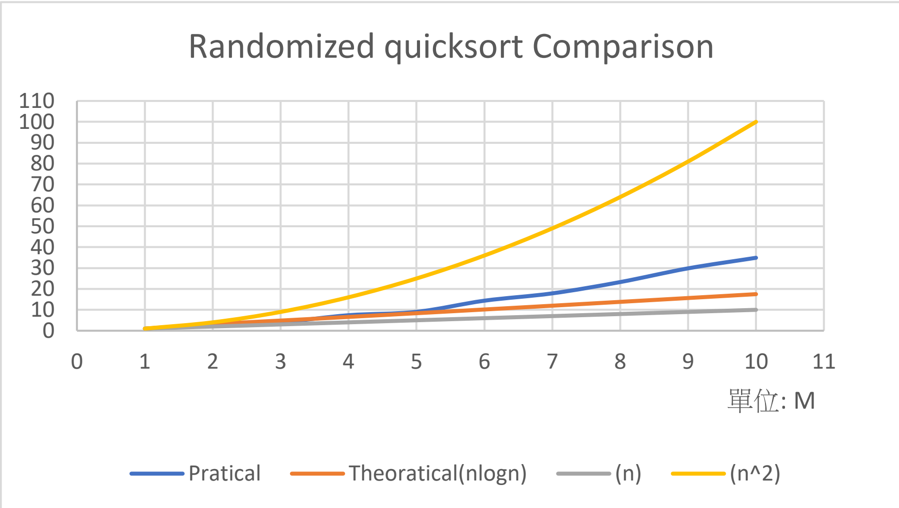
Run Time Comparison

Input Size	Randomized quicksort	Heapsort	Insertion Sort	Merge Sort
10	0.0033ms	0.0022ms	0.0008ms	0.0585ms
100	0.0178ms	0.025ms	0.0224ms	0.1942ms
1000	0.3229ms	0.3631ms	1.8397ms	2.0226ms
10000	2.9115ms	4.7702ms	154.844ms	16.856ms
25000	11.6962ms	12.3864ms	909.187ms	41.2024ms
50000	19.844ms	22.6932ms	3651.05ms	78.3804ms
75000	33.0302ms	40.1652ms	8113.59ms	113.191ms
100000	48.1851ms	45.5921ms	14399.6ms	151.412ms
250000	131.142ms	113.427ms	110209ms	401.183ms
500000	229.299ms	239.852ms	529504ms	816.087ms

以上為 randomized quicksort 和 HW3 做的 heapsort 以及 HW1 做的 insertion sort 跟 merge sort 的 run time 做比較，可以發現在 size 較小的 case 下(約 n = 100 以下)，insertion sort 依然是有著最快的 run time；而在 size 較大的 case 下，heapsort 和 randomized quicksort 則有較佳的 run time；而整體來看，randomized quicksort 看起來有更好的 performance。

而因為 size 較小時可能會有比較大的誤差，於是我又每隔 1000000 跑了 n=1000000~10000000 的情況，根據執行後的結果，並以 n=1000000 做為 unit size，可以做出以下圖表。

Size	1000000	2000000	3000000	4000000	5000000	6000000	7000000	8000000	9000000	10000000
Practical Run Time	708.774	1870.68	2897.65	5258.14	6397.41	10183.1	12669.3	16508.7	21160.5	24746.5
Pratical	1	2.639318	4.088257	7.418641	9.026022	14.3672	17.87495	23.29191	29.85507	34.914514
Theoratical(nlogn)	1	3.150515	4.857841	6.60206	8.373713	10.16723	11.97892	13.80618	15.64705	17.5
(n)	1	2	3	4	5	6	7	8	9	10
(n^2)	1	4	9	16	25	36	49	64	81	100



可以發現執行的實際結果的數量級也較接近理論值 O(logn)。

Analysis randomized quicksort

For worst case

Let q be an integer with $0 <= q <= n-1$.

$T(n) = \max(T(q) + T(n-q-1) + \Theta(n))$

We guess that $T(n) <= cn^2$, then we obtain

$T(n) <= \max(cq^2 + c(n-q-1)^2 + \Theta(n))$

$\quad = c*\max(q^2 + (n-q-1)^2 + \Theta(n))$

We obtain the maximum value of $q^2 + (n-q-1)^2$ at q=0 or n-1, then

$T(n) <= c(n^2-2n-1) + \Theta(n))$

$\quad <= cn^2$

so worst case is $\Theta(n^2)$.

For average case

We can bound the running time of Quicksort by O(n+X), where n is the largest number of calls to Partition, and X denotes the total number of comparisons over an entire execution of Quicksort.

Since each call to Partition only takes a constant time, X will dominate the overall running time. We denote the element of A by Z_1, Z_2, …, Z_n and define Z_{ij} ={ Z_i, Z_{i+1}, …, Z_j } to be the set of elements between Z_i and Z_j. Define X_{ij} =I { Z_i is compared to Z_j }

Since each pair is compared at most once, the total number of comparisons is:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}
\end{aligned}$$

$$\begin{aligned}
\Pr\{z_i \text{ is compared to } z_j\} &= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\} \\
&\qquad + \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\
&= \frac{2}{j-i+1} .
\end{aligned}$$

$$\mathrm{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n) .
\end{aligned}$$

Thus the expected running time of quicksort is O(nlogn).

Conclusion

Randomized quicksort
Randomizes quicksort 的時間複雜度為 O(nlogn)。優點是在普遍的情況下都有較佳的 performance，且屬於 in-place；缺點是他是不穩定的排序法，且會因為 pivot 的位置不同而有不同的效率，如果 pivot 接近極值時就會很差，worst case 的時間複雜度為 $O(n^2)$。

Heap sort
Heap sort 的時間複雜度為 O(nlogn)，空間複雜度為 O(1)。優點是它在 worst case 下的 performance 仍然很好，且屬於 in-place，不需要花額外的空間來進行排序，更重要的是它很好實現。但缺點是它需要頻繁地 swap，所以實際 performance 可能會較差，且 heap sort 不是一種穩定的排序法。

Insertion sort
Insertion sort 時間複雜度為 $O(n^2)$。優點是它很好實現，適合用在 size 較小或是已幾乎排序好的序列，且空間複雜度小，屬於 in-place，不需要花額外的空間來進行排序。缺點則是在 average 和 worst case 下的時間複雜度較其他兩者差。

Merge sort
Merge Sort 的時間複雜度為 O(nlogn)，空間複雜度為 O(n)。它的優點是較適合處理 size 較大的序列，它也是穩定的排序法，可以確保排序後相同 data 的相對位置不會改變。但缺點是需要額外的空間來存中間過程，在空間有限的情況下可能較不適用。

總結來說，randomized quicksort 在普遍情況下都有很優秀的 performance；heap sort 比較適合用於空間有限的情況下；insertion sort 比較適合用於 size 較小或是已大致排好序的序列；merge sort 比較適合用於 size 較大的序列。