排序演算法比較

408410111 林彥廷

Github 連結:

https://github.com/stanley408410111/C-program.git

一、 測試環境

OS: Ubuntu 20.04.2 LTS

CPU : Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz

RAM: 4000704 kB

二、 解釋各種排序演算法

1. Quicksort:

作法:

1. 選定一個基準值(Pivot)
2. 將比 Pivot 小的數值移到基準值左邊，形成左子串列
3. 將比 Pivot 大的數值移到基準值右邊，形成右子串列
4. 分別對左子串列、右子串列作上述三個步驟 ⇒ 遞迴直到左子串列或右子串列只剩一個數值或沒有數值

程式碼:

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

2. Heapsort:

作法:

1. 先建一個 Max Heap
2. 將樹根(最大值)與最後一個節點調換，將最後一個節點(原樹根)取出，並加入已排序數列
3. 相當於對 Max Heap Tree 作 Delete MaxNode
4. 對整棵樹重新調整為最大堆積樹 ⇒ 調整後樹根為 Max Node
5. 重複步驟 2、3

程式碼:

```
Heapsort(A) {
   BuildHeap(A)
   for i <- length(A) downto 2 {
      exchange A[1] <-> A[i]
      heapsize <- heapsize -1
      Heapify(A, 1)
}
```

```
BuildHeap(A) {
   heapsize <- length(A)
   for i <- floor( length/2 ) downto 1
      Heapify(A, i)
}
```

```
Heapify(A, i) {
   le <- left(i)
   ri <- right(i)
   if (le<=heapsize) and (A[le]>A[i])
      largest <- le
   else
      largest <- i
   if (ri<=heapsize) and (A[ri]>A[largest])
      largest <- ri
   if (largest != i) {
      exchange A[i] <-> A[largest]
      Heapify(A, largest)
   }
}
```

3. Mergesort:

作法:

1. 將數列對分成左子數列、右子數列

2. 分別對左子數列、右子數列作上一個步驟 ⇒ 遞迴

直到左子數列、右子數列被分割成只剩一個元素為止

將僅剩的一個元素作為遞迴的結果回傳

3. 對回傳的左子數列、右子數列依大小排列合併

4. 將合併的結果作為遞迴的結果回傳

合併作法：

5. 將左子數列及右子數列依大小合併成一個新的數列

6. 若左子數列的數值都已填到新的數列 ⇒ 將右子數列中未填過的最小值填入新數列

7. 若右子數列的數值都已填到新的數列 ⇒ 將左子數列中未填過的最小值填入新數列

8. 將左子數列及右子數列中，未填過的最小值填到新的數列

程式碼:

```
MergeSort(arr, left, right):
    if left > right
        return
    mid = (left+right)/2
    mergeSort(arr, left, mid)
    mergeSort(arr, mid+1, right)
```

```
    merge(arr, left, mid, right)

end
```

三 、建立 Data 方法及數量

方法:用 rand()函式產生 1000000 筆資料

數字:

```
int a[10000000];
```

```
srand( (unsigned)time( NULL ) );
for(int i=0; i<1000000; i++)
{
    a[i]=rand();
    fprintf(fp,"%d\n",a[i]);
}
printf("Data generated successfully\n");
```

字串:

```
char s[10000000][100];
```

```
srand( (unsigned)time( NULL ) );
for(int i=0; i<1000000; i++)
{
    for(int j=0; j<100; j++)
    {
        int c_ascii = rand()%26;
        char c = c_ascii + 'a';
        s[i][j]=c;
    }
    //printf("%s\n",s[i]);
    fprintf(fp,"%s\n",s[i]);
}
printf("Data generated successfully\n");
```

四 、測量排序時間方式

方法:

利用 gettimeofday()函式測量和 Linux 的 time 指令

```c
#include <sys/time.h>
#include <unistd.h>

 struct  timeval start;
 struct  timeval end;
 unsigned  long diff;

gettimeofday(&start,NULL);
```

...排序法...

```c
gettimeofday(&end,NULL);

diff = 1000000*(end.tv_sec-start.tv_sec)+ end.tv_usec-start.tv_usec;
float sec;
sec=(float)diff/1000000;
printf("第%d次所需時間為 %f (秒)\n",n,sec);
```

五、實驗結果

1. **Quicksort :**

```
The time required to sort 1,000,000 data is 3.764159 sec

real    0m3.766s
user    0m0.337s
sys     0m2.926s
```

2. **Heapsort :**

```
The time required to sort 1,000,000 data is 3.725332 sec

real    0m3.727s
user    0m0.463s
sys     0m2.806s
```

3. **Mergesort :**

```
The time required to sort 1,000,000 data is 3.470276 sec

real    0m3.508s
user    0m0.296s
sys     0m2.807s
```

4. **Quicksort (string) :**

```
The time required to sort 1,000,000 data is 5.579803 sec

real    0m5.584s
user    0m1.540s
sys     0m3.245s
```

5. **Heapsort(string) :**

```
The time required to sort 1,000,000 data is 33.763847 sec

real    0m33.768s
user    0m21.501s
sys     0m2.871s
```

6. **Mergesort(string) :**

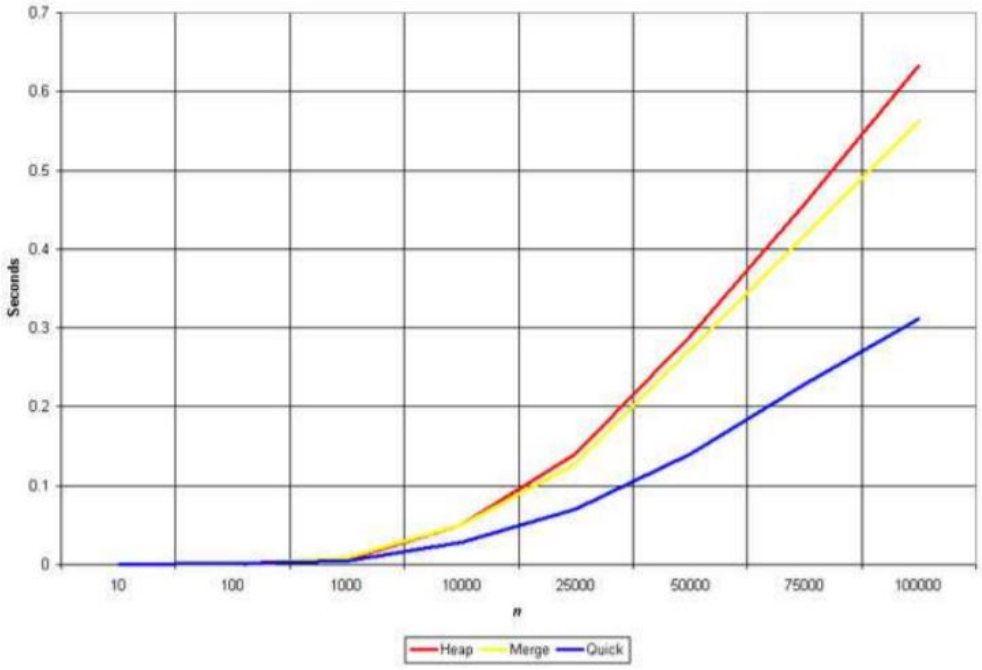
```
The time required to sort 1,000,000 data is 26.516296 sec

real    0m26.530s
user    0m1.735s
sys     0m7.845s
```

六、總結

由實驗可得知 heapsort 執行速度最慢

而 merge sort 略優於 heapsort

但 merge sort 和 heap sort 都相較 quicksort 慢上許多



(Efficiency for $\mathbf{O}(n \log n)$ **Sorts-** http://linux.wku.edu/~lamonml/algor/sort/sort.html)

1. quicksort (unstable)

Time Complexity:

Best Case：O(nlogn)

Worst Case：O(n^2)

Average Case：O(nlogn)

2. heapsort (unstable)

Time Complexity:

Best Case：O(nlogn)

Worst Case：O(nlogn)

Average Case：O(nlogn)

3. mergesort (stable)

Time Complexity:

Best Case：O(nlogn)

Worst Case：O(nlogn)

Average Case：O(nlogn)

七、參考資料

1. https://stackoverflow.com/questions/20106531/mergesort-an-array-of-strings-in-c
2. https://cs50.stackexchange.com/questions/9066/try-to-sort-strings-with-merge-sort
3. https://www.geeksforgeeks.org/swap-strings-in-c/
4. https://stackoverflow.com/questions/46723450/string-sort-with-merge-sort
5. https://stackoverflow.com/questions/5935933/dynamically-create-an-array-of-strings-with-malloc
6. https://groangao.pixnet.net/blog/post/25443935
7. https://stackoverflow.com/questions/50782373/how-to-read-large-text-file-in-c
8. https://stackoverflow.com/questions/20516824/c-fgets-what-if-the-string-is-longer
9. http://www.cplusplus.com/forum/beginner/232570/

10. https://www.quora.com/What-are-the-advantages-and-disadvantages-of-quicksort
11. http://www-cs-students.stanford.edu/~rashmi/projects/Sorting
12. https://medium.com/appworks-school/%E5%88%9D%E5%AD%B8%E8%80%85%E5%AD%B8%E6%BC%94%E7%AE%97%E6%B3%95-%E6%8E%92%E5%BA%8F%E6%B3%95%E9%80%B2%E9%9A%8E-%E5%90%88%E4%BD%B5%E6%8E%92%E5%BA%8F%E6%B3%95-6252651c6f7e
13. https://kopu.chat/2017/08/10/%E5%90%88%E4%BD%B5%E6%8E%92%E5%BA%8F-merge-sort/