Algorithm PA1 Report

B07901103 電機三 陳孟宏

Algorithm

Insertion sort	InsertionSort(*A*) **for** *j* = 2 **to** *A.length* *key* = *A*[*j*] // Insert *A*[*j*] into the **sorted** sequence *A*[1..*j*-1] *i* = *j* - 1 **while** *i* > 0 and *A*[*i*] > *key* *A*[*i*+1] = *A*[*i*] *i* = *i* - 1 *A*[*i*+1] = *key*
Quicksort	QUICKSORT(*A*, *p*, *r*) // Call QUICKSORT(*A*, 1, *A.length*) to sort an entire array **if** *p* < *r* *q* = PARTITION(*A*, *p*, *r*) QUICKSORT(*A*, *p*, *q*) QUICKSORT(*A*, *q*+1, *r*) PARTITION(*A*, *p*, *r*) *x* = *A*[*p*] // break up *A* wrt *x* *i* = *p* -1 *j* = *r* +1 **while** TRUE **repeat** *j* = *j* -1 **until** *A*[*j*] ≤ *x* **repeat** *i* = *i* +1 **until** *A*[*i*] ≥ *x* **if** *i* < *j* exchange *A*[*i*] with *A*[*j*] **else** **return** *j*

Merge sort	MergeSort(A, p, r) **if** $p < r$ $q = \lfloor (p+r)/2 \rfloor$ MergeSort (A, p, q) MergeSort ($A, q+1, r$) Merge(A, p, q, r) Merge (A, p, q, r) $n_1 = q - p + 1$ $n_2 = r - q$ let $L[1..n_1+1]$ and $R[1..n_2+1]$ **for** $i = 1$ **to** n_1 $L[i] = A[p + i - 1]$ **for** $j = 1$ **to** n_2 $R[j] = A[q + j]$ $L[n_1+1] = \infty$ $R[n_2+1] = \infty$ $i = 1$ $j = 1$ **for** $k = p$ **to** r **if** $L[i] \le R[j]$ $A[k] = L[i]$ $i = i + 1$ **else** $A[k] = R[j]$ $j = j + 1$
Heap sort	BUILD-MAX-HEAP(A) $A.heap\text{-}size = A.length$ **for** $i = \lfloor A.length/2 \rfloor$ **downto** 1 MAX-HEAPIFY(A, i)

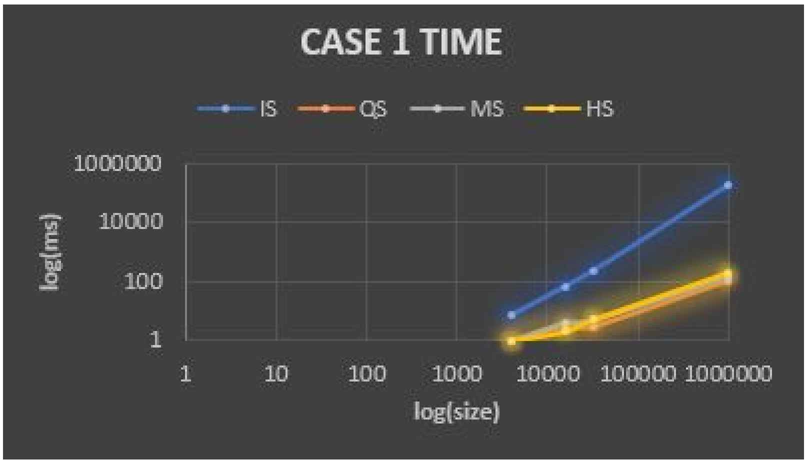
```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

Time Complexity Analysis

演算法	時間複雜度			穩定性
	Best	Average	Worst	
Insertion	O(n)	O(n^2)	O(n^2)	stable
Quick	O(nlogn)	O(nlogn)	O(n^2)	unstable
Merge	O(nlogn)	O(nlogn)	O(nlogn)	stable
Heap	O(nlogn)	O(nlogn)	O(nlogn)	unstable

<Case 1> Random Order (Average Case)

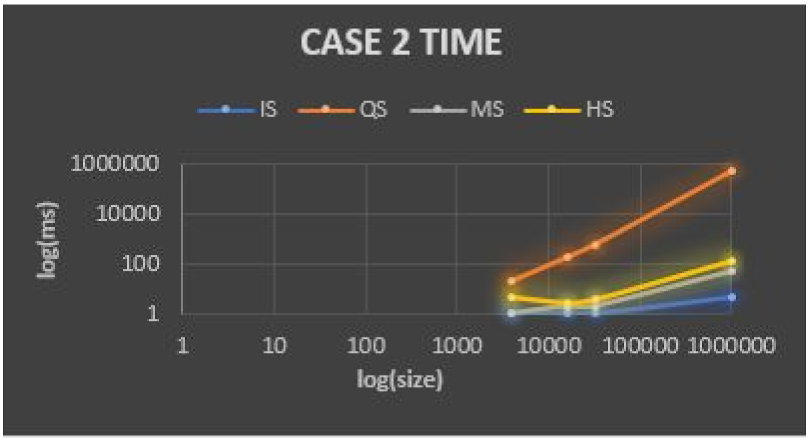
Case_1				
Input Size	IS time(ms)	QS time(ms)	MS time(ms)	HS time(ms)
4000	6.999	1	1	1
16000	68.989	3	4	1.999
32000	230.965	3	4.999	4.999
1000000	201976	99.985	146.978	185.971



1. Average case : Insertion > Heap ~ Merge ~ Quick
2. Insertion sort : O(n^2)
3. Quicksort = Merge sort = Heap sort = O(nlogn)

<Case 2> Sorted (Best Case, non-decreasing)

Case_2				
Input Size	IS time(ms)	QS time(ms)	MS time(ms)	HS time(ms)
4000	1	20.996	1	5
16000	1	190.971	1.999	3
32000	1	626.905	2	4
1000000	5	605011	52.991	145.978

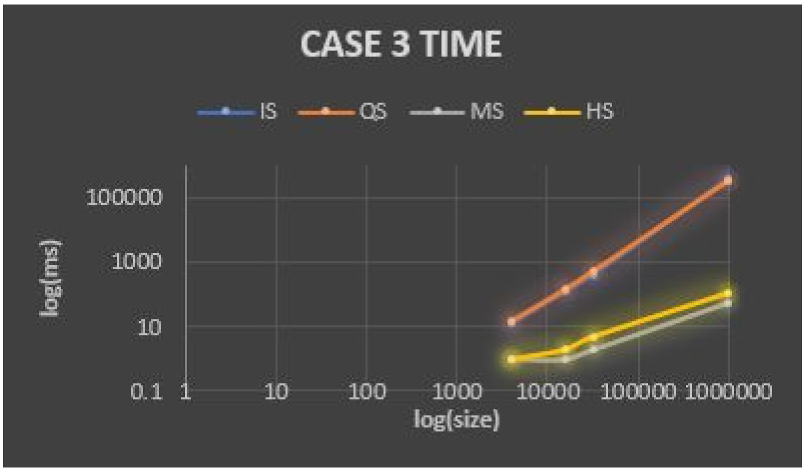


1. Best Case : Quick > Heap ~ Merge > Insertion
2. Insertion sort : O(n), the blue line (lowest)
3. Heap sort : O(nlogn)
4. Merge sort : O(nlogn)
5. Quick sort : O(nlogn), at this sorted case, the quick sort is the slowest sort way, because I use the last element

(data[-1]) to be my pivot, which is the largest number at this case.

<Case 3> Sorted (Worst Case, non-increasing)

Case_3				
Input Size	IS time(ms)	QS time(ms)	MS time(ms)	HS time(ms)
4000	12.998	15.997	1	0.999
16000	130.98	155.977	1	1.999
32000	432.934	557.915	2	4.999
1000000	404658	350380	57.99	114.983



1. Worst case : Insertion ~ Quick > Heap ~ Merge
2. Insertion sort : O(n^2)
3. Quick sort : O(n^2)
4. Heap sort : O(nlogn)
5. Merge sort : O(nlogn)